CURVE-SKELETONS: PROPERTIES, COMPUTATION AND APPLICATIONS

by

NICU DANIEL CORNEA

A Dissertation submitted to the
Graduate School-New Brunswick
Rutgers, The State University of New Jersey
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
Graduate Program in Electrical and Computer Engineering
written under the direction of
Professor Deborah Silver
and approved by

________________________
________________________
________________________
________________________

New Brunswick, New Jersey
May, 2007
ABSTRACT OF THE DISSERTATION

Curve-skeletons: Properties, Computation and Applications

by NICU DANIEL CORNEA

Dissertation Director:
Professor Deborah Silver

The curve-skeleton is a one-dimensional (1D) abstraction of a three-dimensional (3D) object, consisting of a set of curves embedded in 3D space. As a reduced representation, it is useful in many visualization, computer graphics and computer vision tasks, such as: virtual navigation, reduced-model formulation, visualization improvement, mesh repair, animation, matching, etc. In spite its simplicity and proven usefulness, there is still no rigorous definition of the curve-skeleton. Recent attempts at providing such a definition are based on specific algorithms that extract some curve-skeleton.

In the first part of this thesis, we attempt to bring formalism to the concept of a curve-skeleton. First, we survey numerous applications and algorithms designed to compute curve-skeletons and distill the desirable properties of a curve-skeleton from these applications. Our analysis suggests that a rigorous definition cannot be developed without compromising some of the desirable properties. Instead, we adopt a general definition and we strive to provide the end
user with the flexibility of choosing from a variety of possible curve-skeletons, the one that best fits a given application requirements. To this end, we provide an extensive qualitative discussion of each of the surveyed algorithm classes with respect to the properties, and develop algorithms that quantitatively evaluate these properties for a given curve-skeleton.

We also present a robust vector field based hierarchical curve-skeletonization framework and we investigate two types of vector fields: one based on the repulsive force generated by a generalized Newtonian potential function, the other based on a normal front propagation technique, we term a normal diffusion field. We provide a comparative study of two types of vector fields, with respect to the properties of the resulting curve-skeleton.

In the second part of the thesis, we exemplify the utility of the curve-skeleton representation in general as shape abstraction and of our curve-skeletonization framework in particular, in several areas of visualization, computer graphics and computer vision. We investigate volume decomposition and several applications derived from it such as volume animation, texture mapping, focus and context rendering, and compression/packing of volumetric data for hardware accelerated volume rendering and manipulation. We present a robust method for 3D object retrieval based on matching curve-skeletons, and we investigate an extension of our matching and curve-skeletonization framework to four dimensions (4D) and its application in indexing and retrieval of fMRI time series by content.
I would like to thank my advisor, Professor Deborah Silver, for her guidance and support throughout the entire course of my studies at Rutgers University.

I would also like to thank Professor Paul Kantor and all the members of the Novel Indexing and Retrieval of Dynamic Brain Images Project: Dr. Ulukbek Ibraev and Bing Bai (Rutgers University), Professor Ali Shokoufandeh, Dr. Fatih Demirci and John Novatnack (Drexel University), Professor Sven Dickinson (University of Toronto). I thank our collaborators from the Department of Psychology at Rutgers Newark (Professor Stephen Hanson, Dr. Catherine Hanson and Adi Zaimi) for providing the oddball and event perception fMRI datasets, and our collaborators from Princeton University (Professor Jonathan Cohen, Dr. Sean Polyn and Dr. Joshua Greene) for providing the recall and morality datasets. I also thank Professor Lucy Brown from Albert Einstein College of Medicine of Yeshiva University for providing the romantic dataset.

I thank my other committee members, Professor Grigore Burdea and Professor Peter, for their helpful suggestions and comments. I would also like to thank my Vizlab colleagues at Rutgers University: Kristina Bennett, Carlos Correa and Maria Velez for their collaboration.

Part of this work is based upon work supported by the National Science Foundation under Grant Number EIA-0205178. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.
Dedication

To my brilliant wife.
# Table of Contents

ABSTRACT ................................................................................................................................. ii

ACKNOWLEDGEMENT ............................................................................................................... iv

DEDICATION ............................................................................................................................ v

TABLE OF CONTENTS ............................................................................................................. vi

LIST OF TABLES ...................................................................................................................... xiii

LIST OF ILLUSTRATIONS ....................................................................................................... xv

CHAPTER 1 – INTRODUCTION ............................................................................................... 1

1.1 The medial surface .......................................................................................................... 1

1.2 The curve-skeleton ......................................................................................................... 4

PART I – CURVE-SKELETONIZATION THEORY .................................................................. 9

CHAPTER 2 - CURVE-SKELETON APPLICATIONS ............................................................... 10

CHAPTER 3 - CURVE-SKELETON PROPERTIES ................................................................. 19

3.1 Basic concepts of digital topology .................................................................................. 22

3.2 General properties ........................................................................................................... 23

3.2.1 Centeredness ............................................................................................................. 24

3.2.2 Topology preservation ............................................................................................... 27

3.2.3 Connectivity .............................................................................................................. 29

3.2.4 Invariance under isometric transformations ............................................................ 29

3.2.5 Robustness to noise ................................................................................................. 30

3.2.6 Thinness ................................................................................................................... 30

3.3 Application specific properties ..................................................................................... 31

3.3.1 Reconstructability (reconstruction) ........................................................................ 31

3.3.2 Visibility (reliability) ............................................................................................... 33
3.3.3 Smoothness ..............................................................................................................35

3.4 Properties of the curve-skeletonization process ..............................................................36

3.4.1 Efficiency .................................................................................................................36

3.4.2 Hierarchy ..................................................................................................................36

3.4.3 Junction detection and component-wise differentiation ...........................................37

3.4.4 Ability to handle different object representations ....................................................39

3.5 Discussion .........................................................................................................................39

3.6 Curve-skeleton and curve-skeleton set definition ............................................................43

CHAPTER 4 - METHODOLOGY AND ALGORITHMS FOR EVALUATING
CURVE-SKELETON PROPERTIES .................................................................................46

4.1 Centeredness ......................................................................................................................48

4.1.1 Centeredness of an isolated point ............................................................................48

4.1.2 Centeredness of a curve segment .............................................................................50

4.1.3 Centeredness of the entire curve-skeleton ...............................................................52

4.1.4 Discrete approximation ............................................................................................53

4.2 Topology preservation .......................................................................................................54

4.2.1 Identifying and matching connected components ....................................................55

4.2.2 Identifying cavities and tunnels in a 3D object ........................................................56

4.2.3 Identifying loops in the curve-skeleton ....................................................................59

4.2.4 Matching curve-skeleton loops to object cavities and tunnels .................................63

4.3 Connectivity ......................................................................................................................64

4.4 Invariance under isometric transformations ......................................................................64

4.4.1 Computing distance between two curves .................................................................68

4.5 Robustness to noise ...........................................................................................................71

4.6 Thinness .............................................................................................................................72

4.7 Reconstructability ..............................................................................................................75
CHAPTER 5 - EXISTING METHODS FOR EXTRACTING THE CURVE-SKELETON

5.1 Thinning and boundary propagation .................................................................86
5.2 Using a distance field .......................................................................................91
5.3 Geometric methods .........................................................................................95
5.4 General-field functions ....................................................................................99
5.5 Discussion .......................................................................................................102
  5.5.1 Centeredness .............................................................................................103
  5.5.2 Topology preservation .............................................................................104
  5.5.3 Connectivity .............................................................................................104
  5.5.4 Invariance under isometric transformations ...........................................105
  5.5.5 Robustness to noise ...............................................................................106
  5.5.6 Thinness ..................................................................................................106
  5.5.7 Reconstructability ...................................................................................107
  5.5.8 Visibility ................................................................................................108
  5.5.9 Smoothness .............................................................................................108
  5.5.10 Efficiency ...............................................................................................109
  5.5.11 Hierarchy ...............................................................................................110
5.5.12 Junction detection and component-wise differentiation .........................111
5.5.13 Ability to handle different object representations ......................................111
5.5.14 Discussion summary ..................................................................................112
5.6 Concluding remarks ......................................................................................113

CHAPTER 6 – IMPLEMENTATION ........................................................................114

6.1 Evaluation of implemented algorithms ..........................................................120
  6.1.1 Centeredness ...............................................................................................121
  6.1.2 Topology preservation ...............................................................................122
  6.1.3 Connectivity ...............................................................................................123
  6.1.4 Invariance under isometric transformations ..............................................124
  6.1.5 Robustness to noise ..................................................................................126
  6.1.6 Thinness ......................................................................................................127
  6.1.7 Reconstructability .....................................................................................128
  6.1.8 Visibility .....................................................................................................130
  6.1.9 Smoothness ...............................................................................................131
  6.1.10 Efficiency ..................................................................................................132
  6.1.11 Hierarchy ..................................................................................................133
  6.1.12 Junction detection ....................................................................................134
  6.1.13 Ability to handle different object representations .....................................134

6.2 Concluding remarks ......................................................................................135

CHAPTER 7 - VECTOR FIELD-BASED HIERARCHICAL

SKELETONIZATION FRAMEWORK ....................................................................137

7.1 The framework algorithm .............................................................................139
7.2 Locating critical points ................................................................................141
7.3 Force-following ............................................................................................145
7.4 Extracting the curve-skeleton .......................................................................146
7.4.1 Level 0 – the “core” skeleton .................................................................147
7.4.2 Level 1 – the divergence skeleton .........................................................149
7.4.3 Level 2 – the user skeleton .................................................................150
7.5 Types of vector fields ..................................................................................152
  7.5.1 The potential field based repulsive function ......................................152
    7.5.1.1 Skeletons from scattered point sets using the repulsive force function ......159
  7.5.2 Normal diffusion field (NDF) ..............................................................161
7.6 Skeleton representations ............................................................................172
7.7 Discussion ....................................................................................................174
7.8 Comparison of the two types of vector fields ...........................................175
  7.8.1 Centeredness ......................................................................................175
  7.8.2 Topology preservation ......................................................................176
  7.8.3 Connectivity ......................................................................................176
  7.8.4 Invariance under isometric transformations ......................................177
  7.8.5 Robustness to noise ..........................................................................178
  7.8.6 Thinness ............................................................................................178
  7.8.7 Reconstructability ............................................................................179
  7.8.8 Visibility ...........................................................................................179
  7.8.9 Smoothness .......................................................................................180
  7.8.10 Efficiency .......................................................................................181
  7.8.11 Hierarchy .......................................................................................181
  7.8.12 Junction detection and component-wise differentiation .................181
  7.8.13 Ability to handle different object representations .........................181
7.9 Concluding remarks ..................................................................................182

PART II – CURVE-SKELETON APPLICATIONS ........................................183

CHAPTER 8 - VOLUME DECOMPOSITION FOR VISUALIZATION .............184
8.1 Related work ....................................................................................................................185
8.2 Volume decomposition ....................................................................................................187
8.3 Curve-skeletons and volume decomposition .................................................................190
  8.3.1 Using curve-skeleton for volume decomposition ..................................................191
  8.3.2 Computing component boundaries .......................................................................194
8.4 Applications ...................................................................................................................197
  8.4.1 Volume texture mapping ........................................................................................197
  8.4.2 Volume compression for hardware accelerated rendering and manipulation ......198
  8.4.3 Component-based volume rendering .....................................................................200
  8.4.4 Visualizing volume decomposition ........................................................................202

CHAPTER 9 - 3D OBJECT MATCHING ..............................................................................205
  9.1 Previous work ...............................................................................................................207
  9.2 Approach ......................................................................................................................208
  9.3 Many-to-many matching using EMD ........................................................................209
    9.3.1 Visualizing the Similarity ....................................................................................211
  9.4 Experiments ................................................................................................................212
    9.4.1 Base Classification and Object Retrieval .........................................................212
    9.4.2 Part Matching ....................................................................................................214
  9.5 Future work ................................................................................................................216

CHAPTER 10 - INDEXING AND RETRIEVAL OF DYNAMIC BRAIN IMAGES BY CONTENT ......................................................218
  10.1 Background on fMRI and fMRI analysis methods ..................................................219
    10.1.1 Functional Magnetic Resonance Imaging (fMRI) ............................................219
      10.1.1.1 The data ....................................................................................................222
    10.1.2 fMRI analysis methods .....................................................................................223
      10.1.2.1 Normalization ............................................................................................226
### List of Tables

Table 1. Type of evaluation of the different properties. M[0…n] = measurable property, evaluated on a scale from 0 to n, with 0 defined as best and n defined as worst. Y/N = property evaluated as Yes or No. .................................................................47

Table 2. Summary of properties achievable by the various algorithm classes.

Y = guaranteed property, N = property not achievable, P = property achievable as post-processing, S = property can be achieved by some but not all the algorithms in a class, or the property is difficult but not impossible to achieve using this class of algorithms. .................................................................113

Table 3. Centeredness ..................................................................................................................121

Table 4. Topology preservation. ..................................................................................................122

Table 5. Connectivity ...................................................................................................................124

Table 6. Invariance under isometric transformations .................................................................126

Table 7. Robustness to noise ......................................................................................................127

Table 8. Thinness ........................................................................................................................128

Table 9. Reconstructability ........................................................................................................129

Table 10. Visibility .....................................................................................................................130

Table 11. Smoothness ..................................................................................................................132

Table 12. Summary of overall performance of each implementation with respect to the curve-skeleton properties .............................................................................................135

Table 13. Average centeredness for potential field and normal diffusion field. ......................176

Table 14. Average topology preservation for potential field and normal diffusion field. ..........176

Table 15. Average connectivity for potential field and normal diffusion field. ......................177
Table 16. Average invariance under transformation for potential field and normal diffusion field. .................................................................177

Table 17. Average robustness to noise for potential field and normal diffusion field. ..........178

Table 18. Average thinness for potential field and normal diffusion field. ..........................179

Table 19. Average reconstructability for potential field and normal diffusion field. ..........179

Table 20. Average visibility for potential field and normal diffusion field. .......................180

Table 21. Average smoothness for potential field and normal diffusion field. ....................180

Table 22. fMRI database ........................................................................................................232

Table 23. Retrieval performance of the different methodologies presented in this chapter. ........................................................................................................251
List of Illustrations

Figure 1 Medial axis of a 2D shape (a), medial surface of a 3D box (b) and examples of inscribed disks and spheres .................................................................2

Figure 2 A section of the medial axis and the skeleton of an ellipse. The end points of the medial segment marked in red belong to the skeleton but not to the medial axis. ........................................................................................................3

Figure 3 A small change in the object can generate a large change in the medial axis. .........................................................................................................4

Figure 4 Examples of curve-skeletons of different 3D objects .........................................................5

Figure 5 Several curve-skeletons of the same object. A dog object (a); three different curve-skeletons (b-d). All are valid representations, highlighting the difficulty in defining an accurate curve-skeleton. .................................................................5

Figure 6 Virtual navigation applications: virtual colonoscopy (from [98]) (a); virtual bronchoscopy (from [180]) (b). ..................................................................11

Figure 7 IK skeleton of a 3D model used for animation in Character Studio [1] (a); automatically extracted IK skeleton using distance fields (from [239]) (b) or mesh decomposition (from [107]) (c). .................................................................................................12

Figure 8 Segmentation (from [69]) (a); colon straightening (from [215]) (b); curved planar reformation (from [102][103]) (c) .........................................................13

Figure 9 Registration of vascular structures (from [12]). Top row – two vascular structures that will be registered, Bottom row – the two registered vascular structures visualized in the same space, and their common subset. ..................................................14
Figure 10 Matching 3D objects. The algorithms automatically compute the correspondence between different object components, identified as skeleton graph nodes (from [223]) (a), or as skeleton regions (from [49]) (b).

Figure 11 Connectivity graphs of source and target objects of a morph (from [251]). The correspondence between shapes is specified on these graphs.

Figure 12 Mesh decomposition using the skeleton graph (from [38]).

Figure 13 Analysis of scientific data. Quantification of underwater plumes (from [255]) (a); vortex core extraction (from [14]) (b).

Figure 14 Centeredness of an isolated point in 2D. (a) a point perfectly centered within a symmetric figure is at equal distance from the boundary of the figure in all directions. (b) a point cannot be perfectly centered within a non-symmetric figure.

Figure 15 Centeredness of a curve segment within a 3D object. Centeredness of every point on the curve is measured in a plane normal to the curve at the point.

Figure 16 Motivation of centeredness measure of a curve segment. For a 3D rectangular box, the only point perfectly centered within this shape is the center of mass, C, shown in red. Segment AB (shown in blue) appears to be centered within the object.

Figure 17 Two possible curve-skeletons (b and c) for a hollow sphere (a).

Figure 18 Reconstructing objects from the curve-skeleton using the “ball-growing” approach. Original objects (a); curve-skeletons (b); reconstructed objects (c). Each curve-skeleton segment (and the corresponding reconstructed section) is shown in a different color.

Figure 19 Visibility on a 2D shape. The regions of the shape boundary shown in red are not visible from the curve-skeleton (in blue).

Figure 20 A hierarchy of increasingly complex curve-skeletons.
Figure 21 Reconstruction vs. thinness for a thin block object (upper left). Medial surface type of skeleton (a) and the reconstructed object (b); curve-skeleton (c) and the corresponding reconstructed object (d). .................................................................41

Figure 22 Centeredness vs. robustness and smoothness. A curve-skeleton (in red) as a subset of the medial axis/surface is perfectly centered within the figure (a). A smoother curve-skeleton, which is not perfectly centered in the “elbow” region (b). A perfectly centered skeleton cannot remain smooth in the presence of noise (c). ..........42

Figure 23 Visibility vs. robustness. The area on the boundary of the shape between points A and B (shown in red) is not visible from the curve-skeleton (shown in blue). To achieve visibility, the curve-skeleton has to be extended with an additional curve (shown in green), which starts at C and connects to the main curve-skeleton (in blue). This generates a large change in the curve-skeleton as a result of a small boundary irregularity. ..................................................................................42

Figure 24 Measuring centeredness of an isolated point P. Rays shot in all directions are intersected with the object (intersections shown as red dots). .......................49

Figure 25 Measuring centeredness along a curve. At point P on the curve shown in blue, radial rays restricted to a plane normal to the curve at P are intersected with the object. Intersection points are shown as red dots. .................................................50

Figure 26 Concavities, cavities and tunnels. An object with a visible concavity (marked with A), a visible tunnel (marked with B) and an invisible cavity, completely enclosed in the object (a). The convex deficiencies of the object, obtained by subtracting the object in (a) from its convex hull. The cavity, in the shape of a sphere (marked with C), is now visible (b). The descriptors associated with each convex deficiency, as defined in [228] (c). .................................................................57

Figure 27 The augmented spanning tree of a graph. .................................................................59

Figure 28 Invariance under transformation .............................................................................65
Figure 29 Junctions, shown as green dots, detected when two discrete curves are
too close to each other. .................................................................74

Figure 30 Smoothness. A curve which is not smooth (a); a smooth curve (b). .........................78

Figure 31 Visual smoothness of sampled curves. The continuous curve shown in
black is sampled using six samples (red polygonal line) or more than forty
samples (green polygonal line). The green curve appears smoother than the red
curve. .....................................................................................................79

Figure 32 The thinning process on an example 2D shape. Boundary points are
marked “B” at the beginning of each iteration and then removed if they are
simple. .....................................................................................................89

Figure 33 Color-mapped distance field of a chess piece. ..........................................................91

Figure 34 An embedding of the Reeb graph into the original image space. The
height function was used to define the level sets. Each node is taken to be the
centroid of its corresponding contour. Nodes from corresponding contours in
adjacent level sets are connected with a straight line. .........................................................98

Figure 35 The repulsive force field of a 2D shape. The vectors at each location
are shown as small arrows. The inset shows what region of the force field was
magnified. ...............................................................................................101

Figure 36 The candidate voxels selected from a distance field using [74] and the
Voronoi diagram based power shape [5] give a sense of how much additional
post-processing is required to get a 1D curve-skeleton. ...................................................116
Figure 37 Skeletons of various objects obtained using different curve-skeletonization algorithms. The comparisons are on the first steps of all of the algorithms. Note: our implementations are not fully featured and the resultant curve-skeletons may be improved. See text for details about each implementation. All of the objects and algorithm implementations are also available for download at [242].

Figure 38 Running time (log scale) vs. number of object voxels.

Figure 39 A cow model (a) and several levels of a curve-skeleton hierarchy (b), (c) and (d).

Figure 40 Critical points in 3D. Attracting node (a); repelling node (b); saddle with two attracting directions and one repelling direction (c); saddle with one attracting direction and two repelling directions (d).

Figure 41 Critical points and “core” skeleton of the cow object. Attracting nodes are shown as larger red spheres, saddle points are shown as smaller red spheres. The skeleton obtained by force-following is shown in blue. A slice of the underlying vector field is represented as light blue arrows.

Figure 42 Level-1 skeleton of the cow object. On the left, low divergence seeds are shown as larger light-blue spheres. The positions sampled by the force-following algorithm are shown as blue spheres. On the right, only the final curve-skeleton is shown, obtained using larger numbers of divergence seeds.

Figure 43 A 3D model of the letter K with varying curve-skeletons. (a) 3D model, (b) core skeleton using only critical points, (c) adding low divergence values (top 20%), (d) adding boundary seeds at high curvature boundary locations.
Figure 44 Construction of the repulsive force field. Charges C1 and C2 placed on
the boundary exert a repelling force on interior point P, inverse proportional with
the distance between the charge and point P. The resulting force at P (FP) is the
sum of the forces exerted by individual point charges. .........................................................154

Figure 45 A slice of the repulsive force field of a 3D cow model. The vectors at
each point are shown with arrows. ........................................................................................154

Figure 46 Effect of the field strength parameter on the resulting curve-skeleton of
an “L”-shaped object. m=3 (a); m=6 (b); m=10 (c); and medial axis (in red) (d). ..............155

Figure 47 Curve-skeletons of several objects. (a) core skeleton, (b) level 1
curve-skeleton. The parameters are given next to each curve-skeleton. .........................158

Figure 48 Curve-skeleton from scattered points: (a) scattered point set, (b) “core”
skeleton. (c) skeleton inside (in black) and outside the object (in gray) using top
20% divergence points as seeds. (d) “inside-only” skeleton obtained from (c) by
removing skeleton segments touching the bounding box. .....................................................160

Figure 49 Computing the normal diffusion field – Step1. (a) Inward surface
normals, computed as the gradient of the function defining the voxelized object.
(b) Smoothed surface normals. .........................................................................................162

Figure 50 Computing the normal diffusion field – Step2. Illustration of the front
propagation technique used to compute the vector field at the interior voxels. .............164

Figure 51 Normal diffusion field of the cow object .................................................................165

Figure 52 Illustration of the averaging effect of the normal diffusion field .........................166

Figure 53 Critical points computed from a thin plate object. (a) the thin plate
object, (b) using the gradient of the distance function as the vector function,
(c) using the normal diffusion field with no surface smoothing, (d) using NDF
with surface smoothing with a kernel of radius 2, (e) using NDF with surface
smoothing of radius 5, (f) using NDF with surface smoothing of radius 7. .........................167
Figure 54 Normal diffusion field based curve-skeletons. (a) core skeleton;
(b) level 1 curve-skeleton. The parameters are given next to each curve-skeleton. ............170

Figure 55 Two heuristics for automatically determining the sufficient amount of
surface normal smoothing. (a) no two surface normals have the same direction;
(b) no two critical points are 26-neighbors. .................................................................171

Figure 56 Skeleton representations for the cow model (a): curve skeleton (b);
voxel skeleton (c). .........................................................................................................173

Figure 57 Construction of the IK-skeleton of the cow model. Curve-skeleton with
automatically identified joints, shown as green spheres (a), IK-skeleton obtained
by connecting the joints with straight lines (b). .........................................................173

Figure 58 Illustration of hierarchical decomposition of a humanoid shape. ..................189

Figure 59 Curve-skeleton and decomposition of a cow object (a) and (b).
Curve-skeleton and decomposition after user intervention (c) and (d). .......................193

Figure 60 Hierarchical decomposition using the curve-skeleton. ..................................194

Figure 61 Shortest distance decomposition can create disconnected components. ..........195

Figure 62 Distance and vector field based decompositions. .........................................196

Figure 63 Texture mapping. (a) spherical mapping, (b) component based
cylindrical mapping. .....................................................................................................198

Figure 64 Cuboids oriented along the skeleton segments. ...........................................199

Figure 65 Component based selective rendering using cuboids without volume
decomposition (a) and with volume decomposition (b). .............................................202
Figure 66 Visualizing volume decomposition. (a) different components of the object are shown in different colors; a possible curve-skeleton is shown in gray and a line skeleton, obtained by connecting junctions (J0, J1, and J2) and end-points (J3, J4, J5 and J6) with straight lines, is shown in blue. (b) First transformation step: the children of the root junction (J0) are translated along the directions shown as yellow arrows in (a). (c) Second transformation step: the children of J1 and J2 are translated along the directions shown as yellow arrows in (b). 203

Figure 67 Curve-skeleton matching process. 208

Figure 68 The top ten best matches for a small set of objects. Models are sorted by similarity with the query object. 211

Figure 69 Computing similarity between two given objects. Step 1: compute skeletons. In Step 2 the correspondence is shown with color coded regions. 212

Figure 70 Precision/Recall for many-to-many matching algorithm in object retrieval experiment. 213

Figure 71 Part Matching Example: the query object in (a) is matched against each of the objects in (b). The correspondences between their skeletons are shown in red in (c). 215

Figure 72 Causal relations between brain activity and fMRI BOLD signal. 221

Figure 73 Hemodynamic response function modeled as a gamma function [218]. 222

Figure 74 A slice from an fMRI dataset (oddball). The intensities of three voxels over time are shown on the right. The times when the “oddballs” were shown are represented with vertical lines on the graph. 223

Figure 75 Registration of fMRI data to the standardized space. 227

Figure 76 fMRI matching process 230

Figure 77 ROC Curve. 231
Figure 78 The reduced representation based on GLM analysis. (a) t-map,  
(b) thresholded t-map at 1%. ...........................................................................................................235

Figure 79 ROC area histogram for simple overlap. .................................................................236

Figure 80 Normalization error affecting the outcome of simple overlap similarity. ...............237

Figure 81 Fuzzy overlap of radius two. Two reduced representations and their  
dilation with radius two (a) and (b). Overlap of the dilated regions (c). .....................238

Figure 82 ROC area histogram for fuzzy overlap, of radius two. ........................................239

Figure 83 ROC area histogram for EMD radius two. ...........................................................240

Figure 84 Visualization of a curve-skeleton from a real fMRI dataset. .................................247

Figure 85 ROC area histogram for matching skeleton maps thresholded at 1%. ..............249

Figure 86 Retrieval performance, measured as mean area under the ROC curves,  
versus the threshold used to select significant voxels from the skeleton maps. .............250
Chapter 1

Introduction

Three-dimensional (3D) models are now increasingly common in many disciplines ranging from computer aided design, medical imaging, computer graphics, scientific visualization, computational fluid dynamics, and remote sensing. While the 3D representation is invaluable, many applications require alternate “compact” representations of these models, in order to satisfy various requirements related to storage space or computational overhead at application run time. One such reduced representation is a line-like or stick-like one-dimensional (1D) representation which is sometimes referred to as centerline or curve-skeleton [226].

1.1 The medial surface

The need for a reduced representation of shape was recognized from the early beginnings of computer graphics and computer vision, motivated by the limited resources of computer systems. However, as computers became more sophisticated, more powerful and cheaper, similar progress in imaging technology, 3D scanning and computer graphics produced larger and even more complex shape models. As a result, the quest for efficient shape abstractions is still an active area of research. One such abstraction is the representation known as the medial axis in 2D and medial surface in 3D.

The medial axis [27] of a 2D shape is defined as the locus of points which are equidistant from at least two points on the boundary of the object [137]. Depending on the representation of the 2D shape, its medial axis consists of a set of curves or line segments. For example the medial
axis of a polygon consists of a set of straight line segments, while in the case of shapes with curved boundaries their medial axis contains curve segments. In the 3D case, the corresponding representation is called the medial surface [84] because in addition to curves, it can also contain surface patches.

A more illustrative definition of the medial axis/surface is given by the grass-fire analogy [27], where the boundary of an object made entirely of dry grass is set on fire and the medial axis/surface consists of the loci where the fire fronts meet and quench each other. Figure 1 shows the medial axis for a 2D shape and the medial surface of a 3D shape (a box). Note that in Figure 1(b) only the horizontal patch of the medial surface is shaded, but in fact the medial surface consists of nine different patches determined by the lines drawn in red and the edges of the box in black. The term medial is sometimes used to refer to the medial axis (in 2D) or the medial surface (in 3D).

![Fig 1](a) ![Fig 1](b)

**Figure 1** Medial axis of a 2D shape (a), medial surface of a 3D box (b) and examples of inscribed disks and spheres

The skeleton is defined as the locus of centers of maximal inscribed (open) balls (or disks in 2D) [137]. More formally, let $X \subseteq \mathbb{R}^3$ be a 3D object. An open ball of radius $r$ centered at $x \in X$ is defined as:

$$S_r(x) = \{ y \in \mathbb{R}^3, d(x, y) < r \},$$

where $d(x, y)$ is the Euclidean distance between two points $x$ and $y$ in $\mathbb{R}^3$. Note that the definition refers to “open” balls, thus the distance to the ball’s center must be strictly less than the radius $r$,
A ball $S_r(x) \subset X$ is maximal if it is not completely included in any other ball included in $X$ [112]. The skeleton is then the set of the centers of all maximal balls included in $X$. The process of obtaining a skeleton is called skeletonization.

Although the medial axis/surface and the skeleton are closely related, they are not exactly the same. Using the example given in [137], both the medial axis and the skeleton of a 2D ellipse are represented by a line segment but the segment’s end-points belong only to the skeleton and they do not belong to the medial axis. This case is exemplified in Figure 2 where the extremity of the skeleton segment, which does not belong to the medial axis, is shown in red.

![Figure 2](image.png)  
*A section of the medial axis and the skeleton of an ellipse. The end points of the medial segment marked in red belong to the skeleton but not to the medial axis.*

Given the small differences between the medial axis/surface and the skeleton, which only arise in the limit case, many authors use the terms medial axis/surface and skeleton interchangeably.

A major disadvantage of the medial surface (axis) is its intrinsic sensitivity to small changes in the object’s surface due to the way it is defined [7][42]. An illustrative example in 2D is shown in Figure 3(b) where it can be observed how a small change in the object’s shape can generate a large change in the medial axis. Stability of the medial axis has been extensively investigated in two and three dimensions [7][42][80][81] and several methods to prune and filter the noise-generated features of the medial axis have been proposed [9][8].
1.2 The curve-skeleton

The medial surface can be thought of as a 2D abstraction of a 3D shape since it contains isolated points, curves and surface patches which have dimensionality (at most) two. Note that they are still embedded in 3D space.

In many applications, however, a concise representation of 3D objects with curve arcs or straight lines is desirable because of its simplicity. For example animation traditionally uses an IK (inverse-kinematics) skeleton consisting of a small number of connected line segments representing, for example, the torso, arms and legs, in the case of a humanoid figure. Other applications, such as virtual navigation, also require a set of curve paths for traversal. This line-like representation of a 3D object is also known as the centerline or the curve-skeleton [226] and is a simplified one dimensional (1D) representation of the original object, consisting only of curves [56], which, ideally, capture the essential shape (topology and geometry) of the underlying 3D object in an easy to understand and very compact form. Figure 4 shows several sample curve-skeletons of 3D objects.
In Figure 5(b-d) we show three different curve-skeletons of the same object. Each of them is a valid representation, highlighting the difficulty in defining an “accurate” curve-skeleton.

Curve-skeletons have been successfully used in many applications in visualization, computer graphics, computer vision and science in general, such as: virtual navigation, registration, animation, morphing, scientific analysis, recognition, and retrieval. We will review many of these applications in Chapter 2. In spite of the numerous applications which use the 1D representation, until recently ([56]) there have been no attempts to provide a rigorous definition of the curve-skeleton. Each application defined the curve-skeleton according to its own set of requirements. For example, the IK skeleton used in animation is almost always manually specified by the
animator, although several attempts have been made to automate this process. In other applications, the curve-skeleton is chosen a subset of the medial axis but how to choose this subset is usually an application specific issue. This has led to a large number of algorithms, heuristics and hacks in the literature and many more constantly being proposed.

The lack of definition could be explained by the fact that everybody has an inherent sense of what a curve-skeleton is. These very simple line drawings representing various objects are some of the first representations we use when we draw things as children. Although simple, these representations are also very powerful, being able to convey overall shape and meaning with a few lines.

In this thesis, we review more than one hundred articles which describe various approaches to extract a curve-skeleton. The vast literature on the subject expands across many domains, such as computer graphics, computer vision, virtual reality, simulation of physical phenomena, biology, forest studies, oceanography, weather, etc., making it hard to follow all the new algorithm developments. Many of the algorithms in the literature use different definitions, parameters and thresholds and test the algorithm on a limited number of diverse 3D objects. Additionally, some are fine-tuned for a specific application and one may wonder how general and robust these algorithms actually are. Understandably, many of these algorithms can not be replicated and most major visualization/vision packages do not use them. It is hard to decide which algorithm to implement since there are no criteria for evaluation, thereby causing a further proliferation of new algorithms and methodologies.

In order to rigorously define the curve-skeleton, we must first understand the requirements specified by the different applications using curve-skeletons. Based upon a large number of surveyed curve-skeleton applications (Chapter 2), we compiled a list of desirable properties of curve-skeletons. These are discussed in detail in Chapter 3 and include: centeredness, topology preservation, connectivity, invariance under isometric transformations, robustness to noise, thinness, “reconstructability”, visibility, and smoothness. Our initial goal was to use these
properties to develop a more formal definition. However, the analysis of these properties will reveal that some of them are in fact conflicting (discussion at the end of Chapter 3). For example, a thin curve-skeleton cannot reconstruct the original object as well as a surface-like skeleton. Other examples include centeredness and robustness to noise, or centeredness and smoothness, indicating that in order to achieve a certain property others may need to be sacrificed. Therefore, a rigorous mathematical definition of the curve-skeleton may not be inclusive.

Our approach is to adopt a general definition of the curve-skeleton and to develop an evaluation framework, which will assist the end-user in evaluating the degree to which a curve-skeleton achieves each of the properties discussed here (and possibly others). In Chapter 4, we present methodologies and algorithms to quantitatively evaluate a curve-skeleton with respect to the desirable properties. In Chapter 5, we survey and classify a large number of algorithms devised to compute curve-skeletons from 3D objects. A detailed qualitative discussion of each of these classes with respect to the curve-skeleton properties provides a user with valuable information in choosing a methodology that can achieve the properties required by his/her application. In addition, we implement and evaluate one basic algorithm from each class on the same set of 3D shapes. We hope that the curve-skeleton evaluation methodology developed here will be adopted by the community as a standard way to evaluate new curve-skeletonization algorithms.

Next, we describe our work on curve-skeletonization algorithms. Given that the curve-skeleton is not uniquely defined for a 3D object, a desirable quality of a good skeletonization algorithm is the ability to produce curve-skeletons of various complexities from the same object (hierarchical, multi-scale). This property is essential if the curve-skeletons are to be used in multiple applications which require different curve-skeletons. We present a hierarchical curve-skeletonization framework based on vector fields (Chapter 7). We investigate two types of vector fields that can be used to generate curve-skeletons: a repulsive force field resulting from a generalized potential field, and a normal diffusion field, generated by a propagation of surface
normals. These algorithms are discussed in detail and are demonstrated on many 3D models, both synthetic and real-world. In addition, we also evaluate their performance with respect to all the properties of curve-skeletons defined in Chapter 3.

In Part II of this thesis, we demonstrate the utility of a curve-skeletonization framework from a practical perspective in scientific visualization, computer graphics and computer vision. In Chapter 8, we discuss volume decomposition and its applications in volume animation and volume rendering. The curve-skeleton is used to drive the volume decomposition process, taking advantage of our hierarchical skeletonization framework to automatically identify the different volume components and attach them to the skeleton. Once the decomposition is computed, it can be directly used to animate the volume in conjunction with the curve-skeleton, but also to improve rendering by assigning different transfer functions to different components in a focus and context approach. Next, we demonstrate the improvement brought by our curve-skeletonization approach to the 3D object matching problem, by using the extracted curve-skeletons as object abstractions and matching them in order to retrieve similar 3D objects from a large database (Chapter 9). The results encouraged us to attempt to use a curve-skeleton for indexing and retrieval of time-varying datasets as well, more specifically, for functional MRI data. In Chapter 10, we present initial results on using the curve-skeleton as a descriptor for retrieving four-dimensional fMRI data.

In the rest of this work, we may use the terms curve-skeleton and the short form, skeleton, interchangeably. The reader should note that the term skeleton was historically used to refer to the medial axis in 2D and the medial surface in the 3D case.
PART I – Curve-skeletonization Theory

In the first part of this thesis, we attempt to bring more formalism to the concept of a curve-skeleton. We start with a review of many applications using curve-skeletons (Chapter 2) and we compile a list of desirable properties of these one-dimensional abstractions (Chapter 3). Our analysis suggests that a formal definition of the curve-skeleton cannot accommodate all the desirable properties. Our approach is to adopt a very general definition of the curve-skeleton and develop algorithms for evaluating it against each of the desirable properties (Chapter 4). In Chapter 5, we present a comprehensive survey, classification and evaluation of existing curve-skeletonization algorithms. We give a qualitative discussion of how each methodology can achieve the desirable properties, and we provide a quantitative evaluation based on the implementation of a basic algorithm from each class, using a common set of test objects (Chapter 6). Finally, we present a novel hierarchical curve-skeletonization framework based on vector fields (Chapter 7). Our new skeletonization algorithm can produce curve-skeletons of various complexities, useful in different applications.
Chapter 2

Curve-skeleton Applications

One-dimensional abstractions are useful in many applications. Since they were first introduced, curve-skeletons have found uses in many areas of science in general, and in computer science related areas in particular. The applicability of such abstractions spans many domains with no apparent connection between them, making it a difficult task to follow the current developments in the field.

In this chapter we attempt to provide an overview of many applications involving curve-skeletons from various scientific fields, such as: medicine and medical imaging, computer graphics, computer vision, virtual reality, forestal studies, biology and marine biology, visualization and others. We reference numerous articles describing algorithms for computing curve-skeletons or applications using these one-dimensional structures. Although we do not claim that this survey is a comprehensive coverage of all existing curve-skeleton applications, it should give the reader a sense of the vast applicability and usefulness of curve-skeletons.

One of the first modern uses of the 3D curve-skeleton was in virtual navigation [180][240], exploiting its centeredness property to generate collision-free paths through a scene or through an object. Given a scene composed of 3D objects, the curve-skeleton of the background gives a collision-free path through the scene. In virtual endoscopy, curve-skeletons are used to specify collision free paths for navigation through human organs. Traditional endoscopic methods are invasive, dangerous and often uncomfortable to patients. A virtual endoscopy system can produce images similar to those obtained using the traditional technique, but in a non-invasive way and without the discomfort associated with the traditional procedure. After imaging, the organ is
“skeletonized” and a virtual camera is translated along this skeleton path allowing the inspection of the respective organ. Clinical applications include colonoscopy [98][108] [now available from GE, Viatronix, Philips, Siemens], bronchoscopy [180], angioscopy [15] and others. Clearly the skeleton must be a 1D curve and a reliable navigation path ensures the interior organ surface can be fully examined by the physician performing the virtual procedure [94][240]. Figure 6 shows a few snapshots from a virtual colonoscopy application (from [98]) (a) and a virtual bronchoscopy system (from [180]) (b).

![Figure 6](image)

**Figure 6** Virtual navigation applications: virtual colonoscopy (from [98]) (a); virtual bronchoscopy (from [180]) (b).

In traditional computer graphics, line-skeletons are used extensively to specify animation [1][26][155]. These skeletons are sometimes referred to as IK-skeletons (inverse-kinematic) and they control the polygonal representation of the character being animated. Surface polygons are attached to and manipulated through this simple stick-like figure. While most of the IK-skeletons are still manually specified by an animator, methods have been developed to compute the skeleton and the “skinning” (polygon correspondence) automatically [25][138][107][239]. A simplification of the curve-skeleton can be successfully used as an IK skeleton, by replacing curve arcs with straight lines. As demonstrated in [75], volumetric objects can also be animated and manipulated using the same type of paradigm. In [232] and [37] the IK skeleton is
automatically detected from a sequence of volume datasets. Figure 7(a) shows an example IK skeleton defined manually in Character Studio [1]. In Figure 7(b) and (c), we show several examples of IK skeletons detected automatically using a distance field and a discrete medial surface (from [239]) (b), or the results of mesh decomposition (from [107]) (c).

Surgical planning and radiation treatment require accurate extraction (segmentation) and quantification of specific anatomical structures from CT (computed tomography), MRI (magnetic resonance imaging), MRA (magnetic resonance angiogram) or ultrasound data. Such structures include blood vessels and nerve structures. Since these structures have a characteristic tubular shape, methods aimed specifically at extracting the centerline of such tubular objects from medical images have been developed [11][12][69][70] using field-specific knowledge (intensity variation of the blood vessels, connectivity), or simply the volumetric representation of the vessels [168]. The centerline can also be used to aid in other image processing operations such as edge detection and segmentation [69][183][184]. Other uses include curved planar reformation (flattening) [102][103], detection of stenosis [169][219], aneurisms or vessel wall calcifications [222], deforming volumes: unwinding convoluted objects to allow a more
efficient inspection of the overall structure or to remove occlusion (e.g., colon straightening [215]). Figure 8 shows several examples from [69][215][102] and [103]. In (a) segmentation of the blood vessel is performed with the aid of an automatically computed centerline initialized by the user clicking on two points in the 3D model. After the centerline is computed, the estimated surface of the vessel wall is adjusted using an energy minimization algorithm [69]. Figure 8(b) shows a colon straightening application, based on the computed centerline, which can simplify the inspection of the organ [215]. In Figure 8(c) we show a curved planar reformation (flattening of the 3D structure of a blood vessel tree) computed along the centerline of the vessel tree [102][103]. The sequence of images in (c) represents the specification of the resampling plane based on the centerline, a slice of the original data showing that the entire vascular tree is not visible by simply slicing the 3D dataset, and the final 2D image showing the flattened vascular tree.

Figure 8  Segmentation (from [69]) (a); colon straightening (from [215]) (b); curved planar reformation (from [102][103]) (c)
A common operation in medical imaging is the registration of two images from the same patient taken with different modalities (MRI, CT, MRA). Registration is performed by aligning some structures that are visible in both images. One approach is to reduce the dimensionality of the problem by extracting the skeleton of the structure from both images and then aligning the skeletons [12][73][183]. In Figure 9, we show an example of registration of two quite different vascular structures from [12]. The top row shows the two structures that will be registered; the bottom row shows the results of the registration: the first image is a visualization of the two registered structures in the same space, while the second image is a visualization of the common substructures in the two vessel trees.

Another application is matching of 3D objects: given a query object, the task is to find similar or identical objects in a database by using the curve-skeleton [35][49][96][223]. If the
curve-skeleton can differentiate the part structure of the original object, part matching is also possible, where only parts of the objects are matched against the query. In addition to matching, it directly provides registration of the part in the whole object [49][223]. In Figure 10, we show two visualizations of correspondence between curve-skeleton features from [223] and [49], automatically computed by the matching algorithms. Figure 10(a) shows the color-coded correspondence of the nodes in the two skeleton graphs computed from the above 3D objects [223]. In Figure 10(b), the correspondence between curve-skeleton regions is also color coded [49].

![Figure 10](image)

**Figure 10** Matching 3D objects. The algorithms automatically compute the correspondence between different object components, identified as skeleton graph nodes (from [223]) (a), or as skeleton regions (from [49]) (b).

Shape **metamorphosis** (morphing) is the process of generating smooth transitions between two shapes, creating the impression that one object is being smoothly transformed into another. One of the most difficult tasks in generating a successful morph is determining the correspondences between the two shapes used to drive the interpolation process. Various trade-offs are made between allowing the user full control over the process (and turning it into a mostly manual process) and completely automating the correspondence finding algorithm. The curve-skeleton can be used in this context for its simplicity, allowing the user to quickly specify correspondences on the skeletons or enabling matching algorithms to find correspondences more
Additionally, the interpolation process can be performed directly on the skeleton efficiently. Additionally, the interpolation process can be performed directly on the skeleton [23][121][251]. Figure 11 shows the connectivity graphs of the source and the target shapes in a morph (from [251]). The correspondence between these shapes can then be specified on these skeleton-like graphs.

**Decomposing a polygonal mesh** into components is desirable for applications which treat objects as a sum of components. Such a decomposition can be assisted by using the curve-skeleton if it has the ability to distinguish the components of the original object [38][134]. In [107] an inverse approach is taken, where a 1D skeleton is extracted using the mesh decomposition results. Related geometric uses of skeletons include surface reconstruction [5][235] and mesh repair [130]. In [85] and [247], the more challenging problem of segmenting and quantifying an unorganized point cloud is approached using a curve-skeleton. Figure 12 (from [38]) shows a mesh decomposition obtained from the skeleton graph. From left to right: the original mesh, the skeleton graph, and the final mesh decomposition.

In [225], the curve-skeleton is used to define a “skeletal dimensional reduction” for the CAD field. It is also shown how such a representation can be used to reduce boundary value problems over complex solids to lower-dimensional problems over the skeleton. In [123], a skeleton is used for defect analysis of objects produced by casting and forging.

**Figure 11** Connectivity graphs of source and target objects of a morph (from [251]). The correspondence between shapes is specified on these graphs.
Figure 12  Mesh decomposition using the skeleton graph (from [38]).

Skeletons have also been used to improve the efficiency of collision detection of volumetric objects [76] or in surgical simulations [243] and as a general data structure for graphical objects [185]. In [100], a curve-skeleton in combination with convolution surfaces is used for implicit modeling of 3D objects.

One of the biggest uses of skeletons is in analysis of scientific data where complex topologies can be easily explained using line-like drawings. Furthermore, skeletons can be used for reduced modeling and to explain simple physical phenomena. Examples include plume visualization [192][201], vortex core extraction [14], feature tracking [238], protein backbone modeling [88][131], in marine biology for analysis of coral structures [116][117], and many others. Figure 13(a)(from [255]) shows an example of underwater plume visualization and quantification using curve-skeletons. Figure 13(b) (from [14]), shows a curve-skeleton computation algorithm used to extract the core of a vortex.
The previous discussion is by no means exhaustive, but gives a sample of popular uses of skeletons in visualization. Some applications have extra data available to help in the curve-skeletonization process such as velocity fields in the case of vortex core extraction [14] or blood flow data in the case of vessel tracking (e.g., [11][12][69]), while others use only the 3D object. In this work, we concentrate on the more general problem where extra information is not available.
Chapter 3

Curve-skeleton Properties

Until recently there have been no attempts to develop a rigorous definition of the curve-skeleton. As a result, almost each one of these applications developed their own algorithm for computing the curve-skeleton, adhering to a set of specific requirements for that particular application. For example, a curve-skeleton used for virtual navigation has to be centered within the object and smooth in order to avoid sudden changes in the view. On the other hand, a skeleton used for animation (IK skeleton) need not even be completely inside the object that will be animated, but it has to consist of a hierarchy of straight line segments representing the different parts of the object which will move during the animation. The IK skeleton is almost always manually specified by the animator, although several attempts have been made to automate this process. Various other examples can be mentioned here, but the conclusion is that different applications have different requirements and the algorithms designed for different applications will produce curve-skeletons with different properties. The advantage here is that several different methodologies for computing curve-skeletons were developed, the main disadvantage being the increased difficulty in developing a rigorous definition for the curve-skeleton that would be immediately useful for all these applications.

Our goal is to bring more formalism into the concept of curve-skeleton. The first step of any attempt to provide a rigorous definition for the curve-skeleton should be to understand the requirements of all the applications that use a curve-skeleton and identify a common subset which can later be used as a basis for a more formal definition. During the survey of the numerous applications presented in the previous chapter, we compiled a list of desirable properties of the
curve-skeleton, which will be discussed in this chapter. Some of these properties were previously discussed in the literature [9][152][189][224][230][253], but we are not aware of any attempt to formalize them. However, the analysis we present at the end of the chapter suggests that some of these properties are conflicting. In order to develop a rigorous definition of the curve-skeleton, one must find a non-conflicting subset of these properties and completely ignore the rest of the properties. However, as each of these properties is essential to some class of applications, a curve-skeleton which cannot achieve a specific property will be completely unusable in applications that require that property. Furthermore, the selection of a subset of non-conflicting properties will be nothing but completely arbitrary since we cannot judge the relative importance of various properties and the applications requiring them.

In this chapter, we discuss in detail a set of desirable properties of the curve-skeleton. The goal is to provide the end-user with a set of criteria that can be used to evaluate various curve-skeletonization algorithms and implementations. In the absence of a precise definition of a curve-skeleton, this set of criteria (possibly extended with more application specific properties) can be used to evaluate the various algorithms with respect to the requirements of any given application. We group these properties into three classes: general properties, application specific properties, and properties of the curve-skeletonization process itself, as listed below. The curve-skeleton should have these desirable properties:

- General properties:
  - centeredness,
  - topology preservation,
  - connectivity,
  - invariance under isometric transformations,
  - robustness to noise, and
  - thinness.
• Application specific properties:
  o reconstructability (reconstruction),
  o visibility, and
  o smoothness.

Additionally, we have identified the following desirable properties of the curve-skeletonization process itself:
  o efficiency,
  o hierarchy,
  o junction detection and component-wise differentiation, and
  o ability to handle different object representations, including cases where the distinction between inside/outside is not known, such as point sets.

Before we go further, we need to explain why we include application specific properties in this discussion. During our survey of the different applications and algorithms, some of these properties are referenced over and over again and we classified them as general properties. The application specific properties are referenced in fewer application papers, but the following discussion will reveal that their importance goes beyond these applications. To exemplify, reconstructability refers to the ability to reconstruct the original object from the information maintained in the curve-skeleton. While this is useful in only a handful of applications requiring compression, reconstruction also gives a sense of how well the curve-skeleton represents the original object (see discussion below). Note, also, that many other application specific properties could be included in the discussion of a curve-skeleton, depending on the specific requirements of particular applications. For example, the properties relevant to quantification of coral structures in marine biology described in [117] include: maximum branch thickness, angle between branches, branching rate, etc., but we believe these properties are too application specific. In this work, we limit our discussion to the three application-specific properties mentioned above, which we consider to be more general.
We also discuss several properties of the curve-skeletonization algorithm itself. This is necessary because the decision about choosing a particular algorithm for computing the curve-skeleton for a given application cannot be based only on the final result of the algorithm (i.e., the resulting curve-skeleton). In many cases, other factors are equally important. For example, algorithm efficiency expressed in terms of algorithmic complexity or running time is essential to real-time applications, but it cannot be determined simply by inspecting the end result of the algorithm.

It is our hope that this set of criteria for evaluating a curve-skeleton (or a variant of it) will be used in the future to discuss and evaluate the results of new curve-skeletonization algorithms. Shortly after the initial publication of these properties [50], Dey and Sun attempted to provide a rigorous mathematical definition of the curve-skeleton [56] using a medial geodesic function. The paper also includes an evaluation of the developed algorithm with respect to the properties discussed in [50]. While the discussion in [50] and [56] is purely qualitative, in this work, we also develop algorithms for quantitative evaluation of some of these properties. These evaluation algorithms will be presented in the next chapter.

For the following discussion, we will consider the discrete 3D case unless otherwise specified. In this case, objects are represented by voxels on a three-dimensional discrete regular grid. We will use $Sk(O)$ to denote the curve-skeleton of a 3D object $O$. In order to facilitate a better understanding of the subsequent discussion, we start with a brief review of some basic concepts of digital topology. The interested reader is directed to [112][113] for a more detailed discussion of this topic.

### 3.1 Basic concepts of digital topology

Let us consider the discrete space $\mathbb{Z}^3$. Each point $p$ in this space is called a voxel (from volume element) defined by its three integer coordinates $(p_x, p_y, p_z)$. A voxel can be viewed as a cube,
having 6 faces, 12 edges and 8 corners. Two voxels \( p \) and \( q \in \mathbb{Z}^3 \) are 6-adjacent if they have a common face; they are 18-adjacent if they have a common face or edge, and 26-adjacent if they have a common face, edge or corner. The set of 6-adjacent voxels to a voxel \( p \) is also known as the 6-neighborhood of \( p \) denoted by \( N_6(p) \). Similarly, \( N_{18}(p) \) and \( N_{26}(p) \) are the 18- and 26-neighborhoods of \( p \). By \( N^*_6(p) \) we denote \( N_6(p) \setminus \{p\} \); \( N^*_{18}(p) \) and \( N^*_{26}(p) \) are similarly defined.

An \( n \)-path is a sequence of voxels \( p_1, \ldots, p_k \) with \( p_i \) \( n \)-adjacent to \( p_{i+1} \), where \( n \) could be 6, 18 or 26. An \( n \)-connected component is then a set of voxels such that any two such voxels are connected by an \( n \)-path included in that component.

A cavity is a background (white) connected component surrounded by an object (black) component (an empty space inside the object). It is somewhat more difficult and less intuitive to define a tunnel in a 3D object. Intuitively, typical examples of tunnels are: the hole of a donut, or the handle of a coffee cup. Formally, a 3D object has a tunnel if there exists a closed \( n \)-path of object voxels which cannot be deformed into a single voxel and still remain inside the object.

A 3D binary digital picture is described by the quadruple \( P = (\mathbb{Z}^3, m, n, B) \), where \( B \subseteq \mathbb{Z}^3 \) is the set of black voxels representing the object in the picture (also known as object voxels), while \( \mathbb{Z}^3 \setminus B \) represents the background (white) voxels. The pair \( (m, n) \) specifies the object and background connectivities respectively. In order to avoid topological paradoxes such as objects being both connected and disconnected [112], different values must be chosen for \( m \) and \( n \); common choices are (26,6) and (18,6).

### 3.2 General properties

The following properties are important independent of the application of the curve-skeleton: centeredness, topology preservation, connectivity, invariance under isometric transformations, robustness, and thinness.
3.2.1 Centeredness

An important characteristic of a curve-skeleton is its centeredness within the object. This property is probably one of the first discussed in relation to the curve-skeleton, and the curve-skeleton is sometime referred to as the “centerline”. To achieve perfect centeredness, the curve-skeleton must lie on the medial surface since the medial surface is centered within the object and the curves must also be centered within the medial surface patches they belong to [56][200]. Centeredness within the medial surface was previously defined by means of a function defined over the medial surface: the medial geodesic function in [56], or the distance transform (distance to the boundary of the medial surface) in [200]. The points on the medial surface corresponding to ridges of the function are considered centered.

In shape compression and some scientific applications such as vortex core extraction [14], exact centeredness of the curve-skeleton may be essential. However, in most cases, exact centeredness of the extracted curve-skeleton is not required or desired. Given the well known sensitivity of the medial surface to small perturbation on the boundary of the object [7][42], constraining the curve-skeleton to lie on the medial surface may make it sensitive to such changes as well. Instead of exact centeredness, an approximate centeredness (what we call relaxed centeredness) is probably better for many applications such as virtual navigation or animation. For example, in a virtual colonoscopy application, visibility (see below) and smoothness of the navigation path are more important than exact centeredness [108]. We still want the navigation path to be close to the center of the object, but being one or two voxels away from the exact center creates a better curve-skeleton for these applications.

When the curve-skeleton is a subset of the medial surface, centeredness can be defined as shown previously with the help of a function defined over the medial surface. However, once the curve-skeleton is no longer constrained to the medial surface, the definition of centeredness needs to be revised so that it does not refer to the medial surface.
Let us first define centeredness of an isolated point inside an object. The point is considered centered if it is placed at equal distance from the surface of the object, in any direction we look. In other words, if we shoot any number of rays in any orientation going through the isolated point and we compute their intersection with the object, our point equally distanced from the surface along any of these rays. Figure 14 shows the centeredness concept for an isolated point inside a 2D figure. The rays shot through the isolated point P, intersect the boundary of the 2D figure. In Figure 14(a), P is equally distanced from the boundary on each of these rays, since the figure is symmetric. However, if the figure is not symmetric, such as the figure shown in Figure 14(b), a point equally distanced from the boundary in all directions cannot be found. The symmetry in question is symmetry about the center point, a property achievable only by star-shaped polygons (a star-shaped polygon can be built by choosing equally distanced points on many radial rays shot through the center point and connecting the points on adjacent rays.

Figure 14  Centeredness of an isolated point in 2D. (a) a point perfectly centered within a symmetric figure is at equal distance from the boundary of the figure in all directions. (b) a point cannot be perfectly centered within a non-symmetric figure.

Curve-skeletons are composed of curve segments. We define centeredness of a curve-segment similarly to the centeredness of an isolated point, but instead of shooting rays in all directions from every point of the curve-segment, we restrict the rays to a plane normal to the curve-skeleton segment at the given point. Thus, a curve-skeleton segment is centered within the object, if every point on the curve segments is centered within the object in a plane normal to the curve segment at that point (Figure 15).
Centeredness of a curve segment within a 3D object. Centeredness of every point on the curve is measured in a plane normal to the curve at the point.

The above definition of centeredness of a curve segment is motivated by the following observation: in a 3D object, only a single point can be completely centered within the object, if the object is symmetrical. Figure 16 shows such an example: a 3D rectangular box. The only point completely centered within this object is C, shown in red, since it is equally distanced from the surface on any ray passing through it. Thus, any curve segment consisting of more than the point C cannot be perfectly centered within this object. Consider however, the segment AB shown in blue in Figure 16. The segment appears to be centered within this box because every point on it is equally distanced from the surface of the object in a plane normal to the segment.

Motivation of centeredness measure of a curve segment. For a 3D rectangular box, the only point perfectly centered within this shape is the center of mass, C, shown in red. Segment AB (shown in blue) appears to be centered within the object.
Finally, we say that a curve-skeleton is centered within the object if all component curve-segments and isolated points are centered within the object. Note that, from this definition, an object needs to have a certain degree of symmetry for a curve-skeleton to be perfectly centered: the object has to have symmetric cross sections normal to the curve-skeleton segments. This implies that we cannot find a perfectly centered curve-skeleton for a non-symmetric object. However, the center of mass cannot be used because it could lie outside of the object.

This is a limitation of this way of defining centeredness, but a definition in terms of center of mass of the cross-sections will be even worse since the center of mass may be outside the boundaries of the object.

### 3.2.2 Topology preservation

The curve-skeleton should be topologically equivalent to the original object [112][113][137][197]. Preservation of topology can be stated simply as follows:

*Two objects have the same topology if they have the same number of connected components, tunnels and cavities.*

From a strict topological point of view, the actual configuration (geometry) of the two objects is irrelevant. As the well-known joke about topologists goes, a topologist cannot distinguish between the donut he is eating and the coffee cup he is drinking from (because they have the same topology and, from a topological perspective, they are indistinguishable). However, the object and its curve-skeleton are not completely separate objects. They enjoy a special relation and this relation should be accounted for when talking about topology preservation in this context. As pointed out in [112], the above formulation of topology preservation, applied to an object $O_2$ derived from an object $O_1$: “object $O_2$ preserves the topology of object $O_1$, ” is meaningful only if an additional constraint is added: *object $O_2$ is obtained from $O_1$ by only removing object voxels (no adding).* Otherwise, object $O_2$ could end up having a completely new configuration, but still have the same topology. For example, by adding object
voxels to $O_2$, $O_2$ may grow limbs where $O_1$ did not have them. Because $O_2$ still has the same number of connected components, tunnels and cavities, it is topologically equivalent to object $O_1$, but we can no longer say that it represents the same object. With this observation, the above definition of topology preserving is meaningful in the context of skeletonization, where the skeleton $S$ is a subset of the original object $O$.

Of course, we cannot have cavities in a 1D curve, so in a strict sense, a curve-skeleton cannot preserve the topology of an object with cavities. To accommodate objects with cavities, a \textit{relaxed definition of topology preserving} can be formulated using the loops of a 1D curve [200]: the curve-skeleton should have at least one loop around each cavity of the original object. Think of a hollow sphere: the curve-skeleton can be just a circle – a single loop – or many circles in different orientations but all surrounding the same cavity. The latter version may better convey the true shape of the object as shown in Figure 17, but clearly, this is application dependent.

\begin{figure}[h]
\centering
\begin{subfigure}[h]{0.3\textwidth}
\includegraphics[width=\textwidth]{sphere}
\caption{(a)}
\end{subfigure}
\begin{subfigure}[h]{0.3\textwidth}
\includegraphics[width=\textwidth]{circle}
\caption{(b)}
\end{subfigure}
\begin{subfigure}[h]{0.3\textwidth}
\includegraphics[width=\textwidth]{multiple_circles}
\caption{(c)}
\end{subfigure}
\caption{Two possible curve-skeletons (b and c) for a hollow sphere (a).}
\end{figure}

However, tunnels in the original object also create loops in the curve-skeleton. Thus, we will reformulate the relaxed definition as follows:

\textit{The curve-skeleton $Sk(O)$ preserves the topology of the original object $O$ in a relaxed sense if it has the same number of connected components as the original object $O$ and at least one loop for each tunnel and cavity in $O$.}
This formulation has the same constraint as the previous one with respect to configuration. Of course if the object does not have any tunnels or cavities, the curve-skeleton should have no loops at all.

Such a definition could be useful for iconic/abstract representation of objects, where all topological and geometrical features must be represented by the curve-skeleton. In addition, it could be used to develop an algorithm that checks the topology preservation property of a curve-skeleton.

3.2.3 Connectivity

Connectivity is a subset of topology preservation. If the skeleton corresponds to a single connected object, then by maintaining the topology of that object the skeleton would have to consist of a single connected component itself. However, while topology preservation is also concerned about tunnels and cavities in the original object, connectivity only requires a one-to-one correspondence of the connected components in the skeleton and the original object. This is a weaker condition than topology preservation, but it may be sufficient in some applications which are not concerned with the tunnels of cavities of the original object.

A consequence of connectivity of the curve-skeleton is that one can traverse the entire curve-skeleton by starting at one of the endpoints and moving along the individual curve-segments, with some special handling of junctions. This is useful in virtual navigation applications, or for constructing a hierarchy of curve-segments for use in animation.

3.2.4 Invariance under isometric transformations

Given an isometric transformation $T$ (a transformation in which the distances between points are preserved), the curve-skeleton of the transformed object $T(O)$, denoted by $Sk(T(O))$, should be the same as the transformed curve-skeleton of the original object. Formally, the invariance criterion
is given by: \( T(Sk(O)) = Sk(T(O)) \). An isometric transformation must preserve distances, so it can only involve translations, rotations and reflections.

This property is important for matching applications where the curve-skeleton is used as a shape descriptor. In such applications it is common to have similar objects in different orientations that nevertheless still need to be matched and for this reason, the shape descriptor must be insensitive to object orientation.

3.2.5 Robustness to noise

As shown in Figure 3, the medial axis is very sensitive to noise; this is also true in 3D. A desirable property of the curve-skeleton is to exhibit weak sensitivity to noise on the boundary of the object, that is, the curve-skeletons of a noise-free object and the curve-skeleton of the same object with noise should be similar. Formally, let \( N : R^3 \to R^3 \), a noise function, which can be applied to a 3D object \( O \). The robustness to noise condition can be expressed as:

\[
Sk(O) \approx Sk(N(O)).
\]

In this work, we only consider noise which affects the boundary of the object. We do not consider the case where objects have internal structures, in which case noise can affect the internal structure as well.

Note: a robust curve-skeleton cannot be perfectly centered. Exact centeredness would constrain the curve-skeleton to the medial surface, which is extremely sensitive to boundary perturbations.

3.2.6 Thinness

According to the definition adopted in this work, curve-skeletons should be one-dimensional, that is, at most one voxel thick in all directions, except at joints where the skeleton might become thicker to ensure connectivity between the different branches.
We can distinguish three types of curve-skeleton points [28]: regular points on a 1D curve-arc that have exactly two neighbors, end-points of a curve that have exactly one neighbor, and junction points (where curves meet) which can have three or more neighbors. We extend the definition of end-points to include the points that have no neighbors at all. This way, we can accommodate isolated points as part of the curve-skeleton (see Section 4.6). The thinness property can be easily checked if the junction points are known in advance. Some curve-skeletonization methods directly identify junction points [43][130][226]. If junction points are not known in advance, they have to be identified with another method.

Note that some of the general properties discussed above are conflicting. For example, a perfectly centered curve-skeleton cannot be robust to noise at the same time, since in general, any change in the object’s surface will redefine where the center of the object is.

### 3.3 Application specific properties

Although the following properties are application specific, they are also important characteristics of the curve skeleton: reconstructability, visibility, and smoothness.

#### 3.3.1 Reconstructability (reconstruction)

Reconstructability, or reconstruction [74][165] refers to the ability to recover the original object from the curve-skeleton. Given the relation of curve-skeleton to the medial surface/skeleton and the definition of the skeleton as the set of centers of maximal inscribed balls, an obvious choice of reconstruction method is to compute the union of maximal inscribed balls centered at each curve-skeleton point [27][75]. The radius of each ball is given by the “distance transform value”, which specifies the distance to the closest point on the boundary of the object. If we denote the reconstruction operation by $\text{Rec}(\text{skeleton})$, then accurate reconstruction means that $\text{Rec}(\text{Sk}(O)) = O$. Figure 18 illustrates the reconstruction of two objects from their curve-skeletons using the ball growing approach. The original objects are shown in the first column (a), the corresponding
curve-skeletons are shown in the second column (b), and the third column show the reconstructed objects (c). Each curve-skeleton segment in (b) is shown in a different color. The same color is used in (c) to show the corresponding reconstructed section of the object.

A 3D object can be completely reconstructed from its medial surface/skeleton representation by computing the union of maximal inscribed balls. This property has an immediate application in shape compression and volume animation [75]. However, in general, when using the ball-growing approach, accurate reconstruction is not possible from the curve-skeleton alone since it is only a subset of the medial surface (see Figure 18 for two examples). That is, in general, $\text{Rec}(\text{Sk}(O)) \neq O$. To test the degree of reconstruction (accuracy) possible from a given curve-skeleton, every point must be equipped with the distance transform value determined in the original object. Then, the difference volume $O \setminus \text{Rec}(\text{Sk}(O))$ will provide a quantifiable measure of the ability to reconstruct the object.

Reconstruction can be improved by storing more information in each curve-skeleton point and/or by increasing the number of branches in the curve-skeleton. For instance, one could store the three radii of a maximal inscribed ellipsoid and replace ball-growing with an ellipsoid-
growing algorithm. Alternatively, one could simply extract a curve-skeleton with many more branches that should reconstruct more of the original object using the classic ball-growing approach.

Intuitively, the ability to reconstruct an object from an abstraction, such as the curve-skeleton, might seem to be an indication of the quality of that shape abstraction for shape analysis tasks. After all, if the degree of reconstruction is very low, it means the curve-skeleton does not capture much of the original object. However, recent work has shown this is not the case, at least for 3D object retrieval. In [213], some of the best performing shape descriptors for shape matching cannot reconstruct the object at all. Additionally, in [49], the curve-skeleton showed good results for retrieving similar shapes from a large database of general 3D objects.

Thinness and reconstruction are two conflicting properties. Even for objects whose medial surface actually contains only curves (like tubular objects), a one-voxel thick curve-skeleton may not contain all the necessary maximal balls to accurately reconstruct the object (remember that a discrete medial surface/skeleton is usually more than one voxel thick owing to the discrete nature of the object).

3.3.2 Visibility (reliability)

Visibility, or reliability [94][108], refers to the property that every boundary point (point on the object’s surface) is visible from at least one curve-skeleton location. In other words, for any boundary point, there exists a straight “visibility” line connecting it to a curve-skeleton point, which does not cross the boundary of the object. This property is useful for virtual navigation. The term “reliable” is used in relation to virtual endoscopy where it ensures that the interior organ surface can be fully (reliably) examined by the physician performing the virtual procedure. As reliability is a word more commonly associated with robustness, in this work we use the term visibility to refer to this property. Figure 19 illustrates the concept of visibility on a 2D shape. The boundary of the shape which can be seen from the curve-skeleton (shown in blue) is colored
Figure 19  Visibility on a 2D shape. The regions of the shape boundary shown in red are not visible from the curve-skeleton (in blue).

in green. The boundary regions that cannot be seen from any point on the curve-skeleton are drawn in red. The dotted lines are just visual aids, illustrating several limit lines of sight, which define the boundary regions that are not visible from the curve-skeleton.

As with reconstruction, visibility gives a measure of how well the curve-skeleton represents the original object. If the surface area corresponding to a feature of the object is not visible from the curve-skeleton, it can be concluded that the curve-skeleton does not extend to that part of the object and thus, the curve-skeleton is not representative for the original object, at least in the region around the feature. For example, in medical applications like virtual colonoscopy, visibility of every surface point is required for a thorough investigation of the imaged organ. On the other hand, for navigation of a robot in a virtual environment, visibility of every detail on the surface of the surrounding objects is not essential, the rough shape and position of these objects should be sufficient for determining a collision-free path.

A brute-force algorithm to test the visibility of the curve-skeleton checks the visibility of each boundary point with a straight line to every skeleton point. Boundary points which cannot be connected without intersecting the surface are not visible. Efficient visibility computation can be done following the solutions from [94].
Note that for convex objects, visibility does not affect the complexity of the curve-skeleton (more or fewer branches), since the entire convex boundary is visible from any interior point. The same may be true for some configurations of non-convex objects; for example, the entire boundary of a star-shaped object is visible from the center. However, in general, imposing a complete visibility constraint on the curve-skeleton of a non-convex object usually means that a more complex curve-skeleton needs to be computed.

### 3.3.3 Smoothness

Smoothness is not only an aesthetic property, but is useful in some applications. For example, in virtual navigation, which uses the curve-skeleton as a camera translation path, the path should be as smooth as possible to avoid abrupt changes in the displayed image.

Smoothness of a curve is defined as the existence and continuity of derivatives up to whatever order is necessary in an application. Generally, $C^2$ continuity is sufficient for navigation paths.

We can measure smoothness of a curve segment as the variation of the curve tangent direction as we move along the curve. More precisely, we can measure the angles between tangent directions at successive locations along the curve. To ensure smooth navigation, the variation in tangent directions as we move from one point to the next along a segment should be as small as possible. In other words, since the tangent is given by the first derivative, small variations in the tangent direction means small changes in the second derivative, which in turn means small absolute value of the third derivative.

Note: many skeletonization algorithms may smooth the curve-skeleton as a post-processing step.
3.4 Properties of the curve-skeletonization process

There are another four criteria that apply to the algorithm used to compute the skeleton: *efficiency, hierarchy, junction detection and component-wise differentiation,* and *ability to handle different object representations.* As opposed to the previous properties, these properties cannot be evaluated by looking only at the resulting curve-skeleton.

3.4.1 Efficiency

Efficiency of the curve-skeletonization algorithm is essential in applications that need real-time computation of curve-skeletons. Efficiency can be expressed either as asymptotic algorithmic complexity in \( O \) notation or in terms of running time measured on a set of test objects with a specific implementation. The second alternative is less precise since every implementation can be highly optimized to produce very short running times, or can be executed on supercomputers. However, if the application has a specific set of objects that need to be processed at run-time, the actual running time of the algorithm on that set of objects may be more useful than an analytic expression of the algorithmic complexity.

3.4.2 Hierarchy

Because the curve-skeleton is an approximation of the complex components of an object, the skeletonization process should reflect the natural hierarchy of these complexities [48][107]. A *hierarchical approach* is useful because it can generate a set of skeletons of different complexities (*multi-scale skeletons*) that could be used in many different applications. In a strict hierarchy, the curve-skeleton at a certain level in the hierarchy contains all curve-skeletons from the layers below as subsets. Such a strict hierarchy is useful in applications using different levels-of-detail during processing such as multi-resolution matching. Figure 20 illustrates a hierarchy of increasingly complex curve-skeletons of a dog model.
A different kind of hierarchy, mostly useful in animation, consists in defining the hierarchical relations between parts of the same object. For example, the torso is the root of the hierarchy with the limbs and head as its children. This kind of definition is useful for animation as it allows a whole tree of object components to be manipulated by manipulating the root of the tree. For example, the entire arm tree (arm, forearm, hand and all fingers) can be moved by manipulating the shoulder joint.

In this work, we only consider the strict hierarchy. In order to have this property, a curve-skeletonization algorithm has to take at least one user-defined parameter whose effect will be to produce skeletons of different complexities and levels-of-detail. In a strict hierarchy, if the complexity of the resulting curve-skeleton increases as the value of the parameter increases, then the curve-skeleton obtained for a particular value of the parameter has to be completely enclosed in any another curve-skeleton obtained with a higher value of the parameter.

3.4.3 Junction detection and component-wise differentiation

The curve-skeleton should be able to distinguish the different components of the original object, reflecting its part/component structure. This says that the logical components of the object should have a one-to-one correspondence with the logical components of the curve-skeleton (which are curve arcs).

There is no rigorous definition of logical components of a 3D shape, although several attempts have been made. For example, in [107], meaningful components are defined as
components that can be perceptually distinguished from the remaining object. In [105], the component structure of a 2D shape is defined using a combination of substance and connection measures computed around junction points of the medial axis using “visual conductance”. The Reeb graph can also be used to identify object components (see Section 5.3) but its definition is dependent on the choice of the generating function.

As long as the curve-skeleton has identifiable joints or junction points, a partitioning of the original object can be performed to produce a one-to-one correspondence between the different components in the object and the skeleton (for use in animation or mesh decomposition for example).

We would like to make a clear distinction between curve-skeletonization methods that can identify the joints or junction points before or during the extraction of the curve-skeleton and the methods that extract these joints after the curve-skeleton is produced. If the resulting curve-skeleton is only one voxel thick in all directions, detection of joints as a post-processing step is trivial: they are the points having more than two neighbors. In other algorithms, clustering is used to find junctions. It is much harder to identify these junction points before extracting the full curve-skeleton. When extracting joints as a post-processing step, the identified joints are as good as the underlying curve-skeleton and no claims can be made about their significance or stability with respect to the original shape. Joints identified as a first step of the curve-skeletonization process carry more significance simply because they must be related to some intrinsic property of the original object since the curve-skeleton is constructed afterwards. It is the difference between the joints being a by-product of the curve-skeleton or being its source.

Component-wise differentiation is different from topology preservation in that the first one deals with “logical” perceptual components of a single connected object while the latter is concerned with geometrical connected components forming different objects.

Checking whether a curve-skeleton satisfies this property is a difficult task because the definition of “object component” is not precise enough, involving human perception, which is
inherently subjective. However, application specific definitions could be used for such purpose: for example, a curve-skeleton suitable for animation tasks would be one that has a separate branch for each of the limbs and/or parts of limbs of the model being animated. For simple models, the limbs can be easily defined manually or even automatically in special cases. A hierarchical skeleton (see below) could be useful here to produce levels of meaningful components.

In this work, we consider the junction detection property to be a property of the curve-skeletonization algorithm itself. We consider that an algorithm satisfies this property if junctions are determined before or during the skeleton extraction.

### 3.4.4 Ability to handle different object representations

Another important aspect of a curve-skeletonization algorithm is the ability to handle different object representations: voxelized objects, polygonal objects, unorganized point sets (i.e., where the connectivity is not specified and there is no inside/outside information). Most surveyed algorithms operate on either polygonal or voxelized objects, and very few can handle point sets. While conversion between polygonal and voxelized representations is quite easy, working with unorganized point sets proves to be more challenging because of the lack of connectivity information, and the fact that the distinction between inside (the object) and outside is not known.

### 3.5 Discussion

The set of properties discussed here is not a comprehensive set. Many applications have additional requirements for a useful curve-skeleton. For example, one aspect that was not discussed here is scale. Scale is important for matching applications, as 3D datasets found in databases are usually normalized. For example, when matching an airplane model with the model of a human, scale is usually ignored and the two models are considered to be the same size, something that is not true in real life. While scale may not be as important when matching entire
objects, for part matching, scale must be accounted for in the matching algorithm. For example, when trying to identify 3D models that contain a human as a sub-part (one example is a scene containing a human next to an airplane, where the human may be twenty times smaller than the airplane), the matching algorithm must specifically solve for the scale parameters in order to match the original human shape to the human in the composed scene, as the two scenes may have the same scale in the database. In the case of part-matching, scale may be accounted for in the matching algorithm itself, but for other applications it may be useful to explicitly encode scale in the curve-skeleton (and the original objects). Scale is not addressed in this work.

Not all properties described above are essential to all types of applications. Indeed, centeredness is less important for animation than thinness, junction detection or invariance under transformation; reconstruction is crucial for compression applications, while centeredness and visibility are desirable in virtual navigation applications.

Analyzing the above properties, we note that some of them are in fact conflicting. A thin curve-skeleton cannot provide full and accurate reconstruction. In 3D, the medial surface is known to be a fully reversible transformation, i.e., the original object can be fully recovered from the medial surface augmented with the radius of the maximal inscribed balls at each position. However, in general, this is not true for a thin curve-skeleton because of the additional dimensionality reduction relative to the medial surface. The curve segments of a curve-skeleton can only reconstruct a certain class of objects, depending on the adopted reconstruction methodology (see the section about reconstruction above). Figure 21 shows an example for a thin plate object. The object is better reconstructed from a medial surface type of skeleton Figure 21(a) than from a one-dimensional curve-skeleton Figure 21(b).
Centeredness and robustness to noise are also two conflicting properties. A perfectly centered curve-skeleton will be constrained to the medial surface since the medial surface is centered within the object. However, a medial surface constrained curve-skeleton will be plagued by the well known sensitivity to noise of the medial surface. Smoothness could also be compromised by a perfectly centered curve-skeleton that will have to follow every little surface irregularity. In Figure 22(a) illustrates the medial axis of a 2D shape in blue. If the curve-skeleton is constrained to be a subset of the medial axis, a possible curve-skeleton is the portion highlighted with red in (a). This curve-skeleton is perfectly centered within the object, but not as smooth as the curve-skeleton shown in Figure 22(b). In turn, the curve-skeleton in (b) is not perfectly centered in the “elbow” region, as it does not pass through the middle of the dotted line, shown as a black dot. In the presence of noise, Figure 22(c), the curve-skeleton cannot remain perfectly centered without compromising smoothness.
Figure 22  Centeredness vs. robustness and smoothness. A curve-skeleton (in red) as a subset of the medial axis/surface is perfectly centered within the figure (a). A smoother curve-skeleton, which is not perfectly centered in the “elbow” region (b). A perfectly centered skeleton cannot remain smooth in the presence of noise (c).

Another example of conflicting properties is visibility and robustness. A visible curve-skeleton should have a line of sight to every point on the surface of the object. However, small perturbations on the surface could generate such structures that are not entirely visible from the curve-skeleton. To include these, the curve-skeleton will have to include extra branches that extend closer to the respective structures, but this is exactly what a robust curve-skeleton should not have: large changes as a result of small perturbations on the surface. This is illustrated in Figure 23, where the object boundary shown in red between points $A$ and $B$ cannot be “seen” from the curve-skeleton shown in blue. In order to make this area visible to the curve-skeleton, a

Figure 23  Visibility vs. robustness. The area on the boundary of the shape between points $A$ and $B$ (shown in red) is not visible from the curve-skeleton (shown in blue). To achieve visibility, the curve-skeleton has to be extended with an additional curve (shown in green), which starts at $C$ and connects to the main curve-skeleton (in blue). This generates a large change in the curve-skeleton as a result of a small boundary irregularity.
new branch (in green), starting at point \( C \) and which connects to the main curve-skeleton (in blue), needs to be added to the original curve-skeleton.

Due to the conflicting nature of some of the desirable properties of the curve-skeleton, a rigorous mathematical definition that achieves all these properties is not possible. One possible approach is to select a subset of non-conflicting properties and develop a mathematical formulation that satisfies these properties. However, any attempt to select such a subset will be completely arbitrary since we have no reasonable way to assess the relative importance of these properties or of the applications that require them. Such a choice is better left to the individual application developer, which should filter the list of desirable properties to only include those properties required by the specific application.

3.6 Curve-skeleton and curve-skeleton set definition

In this work, we adopt the following generic definition for the curve-skeleton:

*The curve-skeleton of a 3D object is an abstraction consisting of a finite set of curves in 3D space.*

The proposed definition is not surprising; it defines the curve-skeleton as a set of curves embedded in 3D space. A curve \( C \) is a continuous map from a one-dimensional space \( I \) to an \( n \)-dimensional space \( X \): \( C : I \rightarrow X \). In 3D Euclidean space, a curve is defined in parametric form by a set of equations which give the coordinates \((x, y, z)\) of the points on the curve as functions \((f_x, f_y, f_z : R \rightarrow R)\) of a single parameter \( t \): \( x = f_x(t); \quad y = f_y(t); \quad z = f_z(t) \). Because the domain of these 3 functions is one-dimensional \((R)\), we refer to the curve-skeleton as a one-dimensional abstraction of a three-dimensional object.

The definition clearly differentiates the curve-skeleton from any representation that could contain surface patches by stating the finiteness of the curve set (a surface patch can be described as an infinite set of curves; for example by sweeping a curve along a given trajectory).
The curves forming the curve-skeleton can be open or closed (also known as loops). In
general, curves have infinite arc length, since their domain is usually infinite. However, for real
3D objects, the curves forming the curve-skeleton usually have finite arc length (curve segments).
This means that the domain of the functions defining the curve is not the full set of real number
\((\mathbb{R})\), but a bounded interval \([a, b]\). A curve segment \(L\) is defined in parametric form as:
\[ L : l(x(t), y(t), z(t)), \text{or simply } l(t) \text{ with } t \in [a, b]. \]

If \(l(a) = l(b)\), and \(a \neq b\), the curve is said to be a **loop**. The points on the curve corresponding
to \(l(a)\) and \(l(b)\) are called **end points**, while the points corresponding to \(l(t)\), with \(t \in (a, b)\) is
called a **regular point** or interior point of the curve. A special kind of curve segment is obtained
for \(t \in [a, a]\), which maps to an **isolated point** in space. An isolated point, as any other curve, can
be part of the curve-skeleton. For example, the curve-skeleton of a ball could be an isolated point
at the center.

An alternative parameterization is the arc length parameterization, where the parameter \(s\)
indicates the length of the arc measured along the curve from some fixed reference point. If we
take the reference point to be \(t_0\), then \(s = \int_{t_0}^{t} \sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2 + \left(\frac{dz}{dt}\right)^2} \, dt\) and each point on the
curve is defined in terms of the parameter \(s\): \(L : l(x(s), y(s), z(s))\), or simply \(l(s)\) with \(s \in [0, \text{arclength}(L)]\).

Note that the above definition does not fully describe the curve-skeleton, i.e., no algorithm
that uniquely associates a curve-skeleton to a 3D object can be derived from it. In fact, under the
above definition, any 1D abstraction of a 3D object is a curve-skeleton. But not all 1D
abstractions are equal from an application developer’s point of view. For example, under the
above definition, an isolated point is a curve-skeleton of any 3D object, but clearly such a
skeleton is not very useful for animating the original object or for describing its topology. To
account for the multi-scale aspect, we define the curve-skeleton set as:
The **curve-skeleton set** of a 3D object is a family of multi-scale curve-skeletons of that 3D object.

The important aspect to note in the above definition is that the elements in a curve-skeleton set must form a strict hierarchy (multi-scale). In a strict hierarchy, any curve-skeleton at a lower level in the hierarchy (less complex) is completely included in any curve-skeleton at a higher level in the hierarchy (increased complexity). We define complexity of the curve-skeleton in terms of the number of elements that composes it: curves, points, or voxels.

Also note that the curve-skeleton set is not unique for a given object. As a result, multi-scale algorithms may extract different curve-skeleton sets, but the elements in a curve-skeleton set must form a strict hierarchy.

In the rest of this work, we will discuss and analyze the properties of individual elements of the curve-skeleton set (curve-skeletons), and we will not focus on the properties of the entire curve-skeleton set. However, we will discuss the hierarchical (multi-scale) aspect of a family of curve-skeletons, and in Chapter 7 we present a hierarchical framework which extracts curve-skeleton sets from general 3D objects.

In order to allow the user to choose the best curve-skeleton for a particular application, among all the possible (or available) curve-skeletons, we develop an evaluation framework which allows the user to evaluate the degree to which a curve-skeleton achieves the desirable properties. In this scenario, each application will have a well-defined set of required properties for a useful curve-skeleton; given a number of available curve-skeletonization algorithm implementations, one would be able to evaluate each one of them against the required set of properties and choose the best ranking one for the job.

In the next chapter, we present an evaluation framework for the properties discussed here. Where possible, we develop algorithms that provide a quantitative measure of how well a particular curve-skeleton achieves a given property.
Chapter 4

Methodology and Algorithms for Evaluating Curve-Skeleton Properties

The previous chapter discussed the desirable properties of the curve-skeleton we extracted from a survey of many applications and algorithms. In this chapter, we attempt to develop an evaluation framework, which will allow the end-user to quantify the degree to which a curve-skeleton achieves each of these desirable properties. In other words, the end-user will be able to evaluate the curve-skeletons resulting from different algorithms, with respect to each of properties discussed previously.

Obviously not all properties are essential to all applications, so we develop separate algorithms for evaluating each one of the curve-skeleton properties. This will allow an end-user or application developer to create an application specific curve-skeleton benchmark for selecting the most suitable curve-skeletonization algorithms for his/her specific application.

The properties were divided into three groups: general, application specific and properties of the curve-skeletonization process. The general and the application specific properties are properties of the actual curve-skeleton and they can be quantitatively evaluated by inspecting the curve-skeleton and the original object. For example, we can decide whether a curve-skeleton is robust to noise by inspecting the curve-skeletons of the object with and without noise and comparing them. On the other hand, properties of the curve-skeletonization process cannot be quantified only by inspecting the resulting curve-skeleton. For example, the resulting curve-skeleton does not give any information about how efficient the algorithm was.
Another aspect is whether a property is quantifiable by a continuous measure (i.e., it can be assigned a number) or we can only say that a curve-skeleton has or does not have that property (i.e., we can only say yes or no). For example, we can measure centeredness of the curve-skeleton and assign it a number from 0 to 100, but we cannot do the same for the ability of an algorithm to handle different object representations: this must be a yes or no answer. The following table shows the type of evaluation used in this work for each of the properties. The properties marked with M, are evaluated as numbers, and the scale is given next to M. For example, [0 … 1] means the evaluation is on a scale from 0 (best) to 1 (worst). The properties marked with Y/N will be evaluated a yes/no properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type of evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centeredness</td>
<td>M [0 … 1]</td>
</tr>
<tr>
<td>Topology preservation</td>
<td>Y/N</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Y/N</td>
</tr>
<tr>
<td>Invariance under isometric...</td>
<td>M [0 … ∞]</td>
</tr>
<tr>
<td>Robustness to noise</td>
<td>M [0 … ∞]</td>
</tr>
<tr>
<td>Thinness</td>
<td>M [0 … 1]</td>
</tr>
<tr>
<td>Reconstructability</td>
<td>M [0 … 1]</td>
</tr>
<tr>
<td>Visibility</td>
<td>M [0 … 1]</td>
</tr>
<tr>
<td>Smoothness</td>
<td>M [0 … ∞]</td>
</tr>
<tr>
<td>Efficiency</td>
<td>M [0 … ∞]</td>
</tr>
<tr>
<td>Hierarchy</td>
<td>Y/N</td>
</tr>
<tr>
<td>Junction detection and...</td>
<td>Y/N</td>
</tr>
<tr>
<td>Ability to handle different...</td>
<td>Y/N</td>
</tr>
</tbody>
</table>

Centeredness, robustness to noise, thinness, reconstructability and visibility will be evaluated on a scale from 0 to 1 because the measured value can be normalized. Invariance under transformation, smoothness and efficiency are also evaluated as a number, but since the values could not be normalized, the reported values can be any positive number. The rest of the properties will be evaluated as yes/no.
Below we describe the methodology and algorithms for evaluating each of the general and application specific properties of the curve-skeleton, since they can be quantified by simply looking at the resulting skeleton and the original object. Note that the algorithms given below are not the only way to interpret and quantify these properties.

In the following discussion, we consider 3D objects to have non-zero volume. For objects represented by polygonal surfaces, we require the surface to be closed (water tight), but not necessarily a 2D manifold. All that is required is to be able to precisely define the interior vs. exterior of the object. We say a point $P$ is inside a 3D object $O$ if any straight line that goes through $P$ intersects the surface of object $O$ in at least one location on each side of $P$.

## 4.1 Centeredness

Let us first define centeredness for isolated points, curve segments and curve-skeletons more precisely. Then, we will elaborate on the algorithm designed to quantify the centeredness of a curve-skeleton.

### 4.1.1 Centeredness of an isolated point

Centeredness of an isolated point inside a 3D object can be quantified by measuring the distances from the given position to the surface of the object along a set of uniformly distributed radial rays originating at the given position $P$, as shown in Figure 24 (ray intersections with the object are shown as red dots).

Given a position $P$, inside the 3D object $O$, we seed a number of rays $r_i$, uniformly distributed on the unit sphere centered at $P$. Each ray $r_i$ is uniquely identified by two angles in polar coordinates: the azimuthal angle in the x-y plane from the x axis, denoted by $\theta \in [0, 2\pi]$ and the polar angle from the z axis, denoted by $\varphi \in [0, \pi]$. If $P$ is inside the object $O$, each ray $r_i$ should intersect the surface of $O$ at least once on each side of $P$. We choose the closest boundary intersections on each side of $P$: $Q_{i,1}$ and $Q_{i,2}$.
We define centeredness of point $P$ along ray $r_i$ as the positive difference of the distances from $P$ to the two intersection points, normalized by the sum of the two distances, when the two intersection points are distinct. If $Q_{i,1}$ and $Q_{i,2}$ coincide, they both also coincide with $P$, since they should be on opposite sides of $P$ on $r_i$. Also, $P$ must be on the surface of the object. In this case, we take $c_i$ to be 1.

$$c_i = \frac{|d(P, Q_{i,1}) - d(P, Q_{i,2})|}{d(P, Q_{i,1}) + d(P, Q_{i,2})}, \text{ if } Q_{i,1} \neq Q_{i,2}, \text{ and}$$

$$c_i = 1, \text{ if } Q_{i,1} = Q_{i,2},$$

where $d(P, Q)$ is the Euclidean distance from $P$ to $Q$. Note that $c_i$ is a quantity between 0 and 1. When $P$ is on the surface of the object, one of the intersection points ($Q_{i,1}$ or $Q_{i,2}$) or both coincides with $P$ and $c_i$ is 1. When $P$ is exactly centered on $r_i$ with respect to the object surface, $c_i$ is 0.

We define centeredness of a point $P$ inside an object as the mean centeredness of $P$ along all the rays $r_i$ uniformly seeded on the unit sphere centered at $P$:

$$c_p = \frac{1}{\pi} \iint_{\theta=0}^{\pi} \iint_{\phi=0}^{\pi} c_i d\theta d\phi, \text{ or } c_p = \frac{1}{\pi} \iint_{\theta=0}^{\pi} \iint_{\phi=0}^{\pi} c_i d\theta d\phi$$
Since a ray seeded at $P$ is uniquely determined by the two angles in polar coordinates $\theta$ and $\varphi$, we integrate $c_i$ over the domain of these two angles. Note that we integrate only on half the domain of the azimuth angle, $\theta$ (0 to $\pi$, not 0 to $2\pi$) because each ray extends on both sides of $P$. The first part of the formula, which evaluates to $1 / \pi^2$ is simply a normalizing factor which maps $c_p$ to the interval $[0, 1]$.

### 4.1.2 Centeredness of a curve segment

To evaluate the centeredness of an entire curve segment $L$, one could simply compute the mean centeredness of every point on the curve as isolated points. However, such a simple approach considers the curve segment to be a collection of unrelated points. We also need to take into account the relationships between these points that form the curve segment. Let us consider the example depicted in Figure 25, where the curve-skeleton of the 3D box is considered to be a horizontal simple straight line segment. For a point on this line segment, it is not meaningful to measure centeredness along the horizontal direction, because other points can be chosen in that direction which may be better centered. The midpoint of the centerline segment is perfectly centered in all directions, but any other point on the centerline is not. If we compute centeredness in this way, the result will show that the segment is not centered, but direct observation indicates that it is.

![Figure 25](image)

**Figure 25** Measuring centeredness along a curve. At point $P$ on the curve shown in blue, radial rays restricted to a plane normal to the curve at $P$ are intersected with the object. Intersection points are shown as red dots.
In order to avoid situations like the one presented above, where the centeredness measure of a curve segment is not zero even though the segment appears centered, we only compute centeredness in the normal plane to the curve segment at every point on the curve. Let $L$ be a curve segment, parameterized by arc length $l = l(x(s), y(s), z(s)) = l(s)$ with $s \in [0, \text{arclength}(L)]$.

At a position $s = p$ along the curve, we compute the centeredness of the point $P = l(x(p), y(p), z(p)) = l(p)$ as follows: let $N_p$ be the normal plane to the curve at $P$, given by the following equation: $N_p(Q) : \nabla l(p) \cdot (Q - P) = 0$, with $Q \in \mathbb{R}^3$. $\nabla l(p)$ gives the direction of the curve tangent at point $P = l(p)$. Let $Q_0 \neq P$ be an arbitrary point in this plane, distinct from $P$. Let $r(\alpha, t)$ be the family of radial rays in the $N_p$ plane, originating at $P$, parameterized by the angle $\alpha$ they make with the $(P, Q_0)$ direction and by $t$ which gives the location on the ray. $r(\alpha, t)$ is given by the following equation:

$$r(\alpha, t) : Q = T^{-1}R_z(\alpha)T[P + t(Q_0 - P)],$$

where $T$ is the transformation that maps $N_p$ to the $x$-$y$ plane such that $P$ is at the origin and $Q_0$ is along the $x$ axis, and $R_z(\alpha)$ is a rotation matrix around the $z$ axis with angle $\alpha$. The last part of the above equation describes the usual parametric equation of the line determined by $P$ and $Q_0$. To obtain the equations for a ray rotated around $P$ with angle $\alpha$ in the plane $N_p$, we transform the plane $N_p$ into the $x$-$y$ plane such that $P$ is at the origin (transformation $T$), rotate the ray around the $z$ axis with angle $\alpha$ (transformation $R_z(\alpha)$), then transform the plane back to its original position (transformation $T^{-1}$). Practically, $T$ can be determined from the following system of equations:

$$\begin{bmatrix}
TP = 0 \\
TQ_0 = d(P, Q_0) \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T \\
T(\nabla l(p)) = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T
\end{bmatrix}$$

The three equations specify the three conditions described above: $P$ must be transformed into the origin, $Q_0$ must lie on the $x$ axis and the normal of the $N_p$ plane must be parallel to the $z$ axis.
Along each ray \( r(\alpha, t) \), we compute the centeredness measure described for centeredness of an isolated point along a ray. We identify the first two intersections with the object surface, that is, the smallest \( t_1 \) such that \( t_1 \geq 0 \) and the largest \( t_2 \), such that \( t_2 \leq 0 \) and \( r(\alpha, t_1) \) and \( r(\alpha, t_2) \) are on the surface of the object. Let these points be \( Q_{\alpha,1} = r(\alpha, t_1) \) and \( Q_{\alpha,2} = r(\alpha, t_2) \). Centeredness along the ray \( r(\alpha, t) \) is given as before:

\[
c_{\alpha} = \frac{d(P, Q_{\alpha,1}) - d(P, Q_{\alpha,2})}{d(P, Q_{\alpha,1}) + d(P, Q_{\alpha,2})}.
\]

Centeredness of a point \( P \) on curve \( L \) is then computed as the mean centeredness along all radial rays in the plane normal to the curve at \( P \):

\[
c_{p}^{(L)} = \frac{1}{\pi} \int_{0}^{\pi} c_{\alpha} d\alpha \quad \text{or} \quad c_{p}^{(L)} = \frac{1}{\pi} \int_{0}^{\pi} c_{\alpha} d\alpha ,
\]

where the first part of the formula is a normalization factor that maps \( c_{p}^{(L)} \) to the interval \([0, 1]\).

Now that we know how to compute centeredness of a point on a curve segment, we can compute centeredness of the entire segment \( L \) by taking the mean centeredness over the entire curve segment:

\[
c_{L} = \frac{1}{\int_{L} ds} \int_{L} c_{p}^{(L)} ds , \quad \text{or} \quad c_{L} = \frac{1}{\text{arclength}(L)} \int_{L} c_{p}^{(L)} ds .
\]

Again, the first part of the formula is a normalization factor that maps the centeredness value to the interval \([0, 1]\).

### 4.1.3 Centeredness of the entire curve-skeleton

To compute centeredness of the entire curve-skeleton, we will simply compute the mean of the centeredness measure of all the component curve segments and isolated points. Let \( S \) be the curve-skeleton of an object \( O \), a set of curve segments: \( S = \{ L_i \mid i = 1, n \} \).
Then, the centeredness of the entire skeleton can be computed with the following formula:

\[ C = \frac{1}{n} \sum_{i=1}^{n} c_{i} \]

### 4.1.4 Discrete approximation

The formulation for the centeredness of isolated points and curve segments was done in the continuous case, using integrals. A computer implementation will have to compute an approximation of the above quantities, using discrete sampling. For the case of an isolated point, centeredness could be computed by sampling the \([0, \pi]\) intervals of \(\theta\) and \(\varphi\) using \(n\) samples for each:

\[ \theta = \frac{\pi}{n} i, \quad \varphi = \frac{\pi}{n} j, \]

with \(i, j \in [0, n-1]\). For each radial ray \(r_{i,j}\), we compute the centeredness along the ray and we average them to obtain the final centeredness value:

\[ c_{p} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} c_{i,j}}{n^2}. \]

Note that the normalizing factor is \(1/n^2\).

Similarly, we compute centeredness of a point on a curve segment as the average of centeredness values along the radial rays in the normal plane to the curve:

\[ c^{(L)}_{p} = \frac{\sum_{i=1}^{n} c_{i}}{n}. \]

The radial rays are parameterized by the value of the angle \(\alpha = \frac{\pi}{n} i\). Then the centeredness of the entire curve segment is the average centeredness at the \(m\) samples taken along the segment:
In practice, we have taken $n$ (the number of radial rays intersected with the object) as 100.

### 4.2 Topology preservation

As defined in this work, the topological elements of a curve-skeleton are the connected components and the loops. In order to be topological equivalent to the original object, there should be a one-to-one correspondence between the connected components of the curve-skeleton and those of the original object and the curve-skeleton should have at least one loop around each cavity or tunnel in the original object. To quantify the topology preservation property, we have to detect the connected components, cavities and tunnels in the original object and match them to the connected components and the loops of the curve skeleton. We will describe the process of identifying and matching these topological features in detail below.

Once we matched the connected components of the curve-skeleton and the object, and the loops in the curve-skeleton with the cavities and tunnels of the object, we can now decide whether the curve-skeleton preserves the topology of the object by checking the conditions specified in the relaxed definition of topology preservation given in Section 3.2.2. We should have:

- a one-to-one correspondence of the connected components of the curve-skeleton to the connected components of the object and vice-versa,
- a one-to-many correspondence between the cavities and tunnels of the object to loops of the curve-skeleton, and
- a one-to-one correspondence between every loop of the curve-skeleton and cavities or tunnels of the object.

\[
C_L = \frac{\sum_{j=1}^{m} c_{p_j}^{(L)}}{m}.
\]
If all three conditions above are satisfied, the skeleton preserves the topology of the object. Otherwise, this property is not achieved.

In the rest of this section, we will detail the procedures of identifying cavities and tunnels in an object, loops in a curve-skeleton, and connected components in both the object and the curve-skeleton, as well as how we match these topological features. We take into account different representations of the 3D object: voxelized or polygonal. Adjacency relations between component elements of the 3D object are defined differently depending on the specific object representation. For a voxelized representation, the adjacency relations are defined in Section 3.1. For a polygonal representation, two vertices are adjacent if there is an edge connecting them; two faces are adjacent if they share an edge. Adjacency relation in the curve-skeleton can be defined similarly: Two curve-skeleton segments are adjacent if they share at least one of their end points.

4.2.1 Identifying and matching connected components

Connected components of a 3D object or curve-skeleton can be identified using a connected components labeling algorithm, detailed below:

1. Mark all elements (vertices, voxels, faces, curve segments) of the 3D object or curve-skeleton as un-processed.
2. Initialize stack $S$ to empty
3. While there are un-processed elements left
   4. Select the first un-processed element $e$
   5. Label $e$ as part of a new connected component
   6. Mark $e$ as processed
   7. Push $e$ into stack $S$
4. While stack $S$ is not empty
   5. Extract the first element, $v$, from $S$
   6. For each un-processed element $w$ adjacent to $v$
      7. Label $w$ with the same component label as $v$
      8. Push $w$ onto the stack $S$
The above algorithm can be used to build the set of object connected components \( \text{Occ} = \{Oc_1, Oc_2, ... Oc_n\} \), as well as the set of curve-skeleton connected components \( \text{Scc} = \{Sc_1, Sc_2, ... Sc_m\} \), using the appropriate adjacency definition for each case.

In order to preserve topology of the original object, \( \text{Occ} \) and \( \text{Scc} \) must have the same number of elements, and there must be a one-to-one correspondence between the elements in the two sets. To match a curve-skeleton component \( Sc_i \) with an object component \( Oc_j \), we need to perform an inclusion test: if \( Sc_i \) included in \( Oc_j \), then we match \( Sc_i \) with \( Oc_j \). The inclusion test is trivial to perform for a voxelized representation of the 3D object. For a polygonal representation however, the inclusion test will actually test whether the \( Sc_i \) is inside \( Oc_j \). For this, \( Oc_j \) would have to define an orientable surface.

### 4.2.2 Identifying cavities and tunnels in a 3D object

An algorithm to identify and quantify cavities and tunnels of a discrete 3D object was presented in [228]. The algorithm starts with identifying all convex deficiencies of a 3D object by subtracting the object from its convex hull. Let \( O \) be the object and \( CH(O) \) its convex hull, both subsets of \( \mathbb{Z}^3 \). The convex deficiencies are the connected components of the difference between the convex hull of the object and the object: \( CH(O) \setminus O \). Convex deficiencies can be classified into three groups: cavities, tunnels and concavities. According to [228], a cavity is formed exclusively by voxels belonging to \( CH(O) \setminus O \) which have no neighbors in the complement of the convex hull (\( \mathbb{Z}^3 \setminus CH(O) \)) (named c1 voxels in [228]), while a tunnel or concavity also contains voxels of \( CH(O) \setminus O \) that have at least a neighbor in \( \mathbb{Z}^3 \setminus CH(O) \) (named c2 voxels in [228]). A convex deficiency descriptor is computed in [228] using a distance field ordered constrained topological erosion, which reduces the convex deficiency to a set of curves and surfaces. Topological erosion is performed only on the voxels in \( CH(O) \setminus O \) which do not have a neighbor in \( \mathbb{Z}^3 \setminus CH(O) \) (c1 voxels). In other words, voxels on the surface of the convex hull are not removed. If the topological erosion is continued until no more voxels can be removed, we can use
the convex deficiency descriptor to distinguish between tunnels and concavities. Both the concavity and tunnel descriptor contain a set of \( c_2 \) voxels, voxels belonging to the surface of the convex hull, which form one or more surface patches or are isolated points. The difference is in the set of remaining \( c_1 \) voxels after the topological erosion. A single \( c_1 \) voxel remains for a concavity, while for a tunnel we have a set of curves formed of \( c_1 \) voxels connecting the surface patches formed of \( c_2 \) voxels (see Figure 6 in [228]). However, if the topology of the full set (\( c_1 \) and \( c_2 \) voxels) is preserved but only \( c_1 \) voxels are removed, and the topological erosion is continued until no more simple \( c_1 \) voxels exist, the concavity descriptor cannot have any \( c_1 \) voxels since the last \( c_1 \) voxel will be a simple voxel as well and will be removed. Figure 26 shows the concept of cavity, tunnel and concavity in a 3D object. In Figure 26(a), we show a 3D object having one cavity, one tunnel and one concavity. Since the cavity is completely enclosed in the object, it is not visible in (a). Figure 26(b) shows the convex deficiencies of the object, corresponding to the cavity, concavity and tunnel, as determined by subtracting the object from its convex hull. The \( c_2 \) voxels are colored in red, while \( c_1 \) voxels are shown in yellow. The corresponding descriptors are shown in Figure 26(c).

**Figure 26** Concavities, cavities and tunnels. An object with a visible concavity (marked with A), a visible tunnel (marked with B) and an invisible cavity, completely enclosed in the object (a). The convex deficiencies of the object, obtained by subtracting the object in (a) from its convex hull. The cavity, in the shape of a sphere (marked with C), is now visible (b). The descriptors associated with each convex deficiency, as defined in [228] (c).
The algorithm is outlined below:

1. Compute the convex hull \( CH(O) \) of the 3D object \( O \)
2. Compute convex deficiencies: \( CD := CH(O) \setminus O \)
3. Perform a connected component labeling of \( CD \). The result is a set of convex deficiencies \( \{CD_1, CD_2, \ldots, CD_n\} \).
4. For each convex deficiency \( CD_i \),
   a. \( nc2 := \) number of \( c2 \) voxels in \( CD_i \)
   b. If \( nc2 = 0 \)
      i. \( CD_i \) is a cavity
   c. Else
      i. Compute the convex deficiency descriptor
      ii. \( nc1 := \) number of \( c1 \) voxels in the descriptor
      iii. If \( nc1 = 0 \)
         a. \( CD_i \) is a concavity
      b. Else
         i. \( CD_i \) is a tunnel.

There are many algorithms that could be used to compute the convex hull of a 3D object. If the object is given as a voxelized representation, we can reduce the number of point samples given as input to the convex hull algorithm by taking only the surface voxels of the 3D object (object voxels having at least one background neighbor). For a polygonal representation, we could input only the vertices of the object to the convex hull algorithm.

The next step is to compute the volume difference between the convex hull and the original object. Convex hull algorithms usually return a polygonal representation of the convex hull. If the original object is in voxelized form, we can voxelize the convex hull and perform a simple voxel-by-voxel comparison to determine the intersection volume. If the original object is in polygonal form, one could use the same voxelization approach: voxelized both the object and the convex hull and compute the voxelized volume intersection. Since the topological erosion step requires a voxelized representation, it is best to keep all objects in voxelized form.
4.2.3 Identifying loops in the curve-skeleton

The curve-skeleton can be regarded as an undirected graph, with nodes represented by segment end-points and arcs represented by segments. All arcs are assigned the same constant weight (1 for example). As a pre-processing step, we will remove all two-way junctions from the curve-skeleton. A two-way junction is a curve-skeleton segment end point, common to only two segments. By removing these two-way junctions, the adjacent segments will be merged into one.

Cycles in this graph can be detected using a depth-first search. Starting from an arbitrary node, we traverse the graph in a depth-first manner and build an augmented spanning tree in the process. The augmented spanning tree contains those graph edges visited during the regular depth-first search (called forward edges) and is augmented with those graph edges that connect back to a previously visited node in the graph (called back edges), which are normally ignored during a regular depth-first search. Figure 27 shows an undirected graph (a) and the augmented spanning tree built during the depth-first search (b). The order in which the graph edges are traversed is shown on the graph as numbers next to each edge. The back-edges, shown in red are oriented from the current graph node to a previously visited node.

![Figure 27](image)

**Detected cycles:**

1) b, i, j, k, d, c, b  
2) j, h, i, j  
3) k, h, j, k  
4) b, g, d, c, b  
5) d, g, h, k, d

**Figure 27** The augmented spanning tree of a graph.
A cycle can be detected by starting at a back-edge and walking up the tree until we reach the destination node of the back edge. During the walk up the tree, we also take any back edge in order to shorten the path to the source node. The five cycles detected using this procedure on the example graph in Figure 27, are: \((b, i, j, k, d, c, b)\), \((j, h, i, j)\), \((k, h, j, k)\), \((b, g, d, c, b)\) and \((d, g, h, k, d)\).

In the above example, the graph has five cycles and the algorithm detected five cycles, but the first cycle, \((b, i, j, k, d, c, b)\), is not the shortest one and may enclose several smaller cycles. A better choice would be \((b, i, h, g, b)\). Since the final goal is to match each loop of the curve-skeleton to a cavity or tunnel in the original object, we require that the identified loops be the shortest ones in terms of number of nodes and edges included in the cycle. At this point, it is also obvious why the pre-processing phase requires the removal of two-way junctions: these junctions will artificially increase the length of a path between two nodes in terms of number of contained nodes and edges.

In order to detect the shortest cycles in the graph, instead of walking up the tree from a back edge to detect the full cycle, we look for the shortest path between the nodes of the back edge in the original graph, not including the back edge. This can be done by a breadth-first search on the original graph, starting from one node of the back edge and ending when the other node is reached. With this approach, for the graph in Figure 27, the detected cycles are: \((b, g, h, i, b)\), \((j, h, i, j)\) and \((j, h, k, j)\), \((k, h, j, k)\), \((g, d, c, b, g)\) and \((g, h, i, b, g)\), and \((g, b, c, d, g)\) and \((g, h, k d, g)\). Note that multiple shortest paths may exist between two nodes of the graph. It is important to retain all these shortest paths until all back edges are processed. After that, duplicate loops can be easily detected and discarded. The entire algorithm is outlined below:

```plaintext
1 Build a graph \(G\) from the curve-skeleton  
2 Mark all nodes as not visited  
3 Initialize the spanning tree \(ST\) to empty  
4 Select a node \(v\), as the initial node  
\hspace{1em} // perform a depth-first search starting at \(v\)
```
DFS(v, G, ST)

Initialize the loop set LS to empty

For each back edge (v, w) of ST

Find_All_Shortest_Paths(v, w, G \ (v, w), LS)

Remove duplicate loops from LS and report the remaining loops.

The function DFS(node v, graph G, spanning_tree ST), invoked in line 10 above, implements the depth-first search algorithm in a graph G, starting from node v and building the augmented spanning tree ST in the process. In line 13, the function Find_All_Shortest_Paths(node v, node w, graph G, loop_set LS) finds all shortest paths from v to w in the graph G and adds them to the loop set LS. As mentioned above, the algorithm may find the same loop multiple times, so in line 14, we remove duplicates and report the remaining loops as the curve-skeleton loops.

Function DFS(node v, graph G, spanning_tree ST)

Mark node v as visited.

For each neighbor w of v

If w is not marked as visited already

Add (v, w) to ST as a forward edge

Call DFS(w, G, ST)

Else

Add (v, w) to ST as a back edge

The function Find_All_Shortest_Paths(node v, node w, graph G, loop_set LS) uses a modified version of Djikstra’s shortest path algorithm. Instead of storing only one predecessor (as required for a simple shortest path), we store all predecessors that give the same path length to a node:

Function Find_All_Shortest_Paths(node s, node e, graph G, loop_set LS)

// Initialization

For each vertex v in G
\[d[v] := \text{infinity}\]
\[\text{previous}[v] := \text{undefined}\]
\[d[s] := 0\]
\[S := \text{empty set} \quad // \text{set of nodes that were already processed}\]
\[Q := V[G] \quad // \text{set of un-processed nodes}\]
\[\text{done} := \text{false}\]

\textbf{While} \ Q \ \text{is not an empty set AND not done}

\[u := \text{Extract\_Min}(Q) \quad // \text{get the node with the smallest value}\]
\[\text{of the d function}\]
\[S := S \cup \{u\} \quad // \text{add node u to the set of processed}\]
\[\text{// nodes S}\]
\[\text{// relax each edge adjacent to u}\]

\textbf{For} each edge \((u,v)\) adjacent to \(u\)

\textbf{If} \ v == e

\[\text{done} := \text{true}\]

\textbf{break;} \quad // \text{we reached the target node. Stop the search}\]

\textbf{If} \ d[u] + w(u,v) < d[v] \quad // \text{Relax (u,v)}

\[d[v] := d[u] + w(u,v)\]

\[\text{previous}[v] := \{u\}\]

\textbf{Else}

\[\text{previous}[v] := \text{previous}[v] \cup \{u\}\]

// At this point, all shortest paths from s to e were detected

// and we add them to the loop set LS

\[L = \{} \quad // \text{a loop}\]

\text{CollectLoops}(s, e, L, \text{previous}, LS)

The function CollectLoops(node s, node e, node[], previous, loop L, loop_set LS) extracts
every path from e to s stored in the previous list, appends it to the L loop and adds it to the loop
set LS.

\textbf{Function} \ CollectLoops(node s, node v, loop L, node[], previous,

\[\text{loop}\_set \text{LS})\]

\[L := L \cup \{v\}\]
If \( v == s \)

\[
LS := LS \cup \{L\}
\]

Else

For each \( v \) in previous[\( v \)]

CollectLoops(\( s, w, L, previous, LS \))

### 4.2.4 Matching curve-skeleton loops to object cavities and tunnels

Once all the cavities and tunnels in the original object as well as all the loops in the curve-skeleton are identified, we need to match these topological features. We want to match each curve-skeleton loop to a single cavity or tunnel in the original object. We use the following algorithm:

For each loop \( L = \{p_1, p_2, ..., p_n\} \)

Compute a triangulation \( TL \) of the points in \( L \)

For each cavity and tunnel \( C \) of the original object

If \( (TL \text{ intersect } C) \) is not empty

Record \( C \) and \( L \) as matched

Each loop is transformed into a set of sample points collected from all the component segments of the loop. We compute a triangulation of these points using a non-planar polygon triangulation algorithm and we intersect the resulting triangulation (the loop surface) with every cavity or tunnel of the original object. If the intersection is not empty, we can directly record the match. Additional constraints can be added after the intersection is computed, before deciding whether there is a match or not. One such constraint could be imposed on the relative volumes determined by the intersection with the loop triangulation. For example, after computing the intersection between the loop surface and a cavity, we can divide the volume on the cavity into two parts, one on each side of the loop surface. Then, in order to record a match, we can require that the two volumes are relatively equal. This would mean that in order to match a loop to a cavity, we require not only that the loop surface intersect the cavity, but divide the cavity into almost equal parts.
4.3 Connectivity

The number of connected components of the curve-skeleton and the original object can be easily determined using the connected component labeling algorithm presented in Section 4.2.1. The same section describes how we can match connected components from the curve-skeleton to connected components of the original object. If all connected components of the curve-skeleton are matched to exactly one connected component of the object and vice-versa, the curve-skeleton achieves the connectivity property.

Since connectivity is closely related to topology preservation, the two measures may overlap. But topology preservation also includes information about the correspondence between curve-skeleton loops and object cavities and tunnels, while connectivity is concerned only with the connected components.

Note that this definition of connectivity also includes the correspondence between the skeleton’s connected components and the original object’s connected components. In other words, it is not sufficient for the curve-skeleton to have the same number of connected components as the original object, there also has to be a one-to-one mapping between them, determined by the connected components matching algorithm described in Section 4.2.1.

4.4 Invariance under isometric transformations

To evaluate invariance of a curve-skeleton under an isometric transformation, we need to actually compute two curve-skeletons. Given an original object $O$ and an isometric transformation $T$, we compute the transformed version of the object $T(O)$ and two curve-skeletons: $S_O$, the curve-skeleton of the original object, and the curve-skeleton of the transformed object $S_{T(O)}$. Invariance of the curve-skeleton under transformation $T$ translates into $T(S_O) = S_{T(O)}$, that is, the transformed curve-skeleton of the original object is the same as the curve-skeleton of the transformed object. In order to quantify transformation invariance, we have to compute the
difference (distance) between two curve-skeletons: \( T(S_O) \) and \( S_{T(O)} \). Figure 28 illustrates the process.

In general, when comparing the two curve-skeleton, we cannot make any assumptions about their relative position, topology or geometry. Curve-skeletons can be regarded as graphs with the junctions representing the nodes in the graph and the curve-segments representing the edges. Many general graph matching algorithms have been proposed [110][111][179][206][214] and these could be used to match two arbitrary curve-skeletons. However, in this case, we have two curve-skeletons computed from two objects that are identical except for an isometric transformation applied to one of them. After we apply the same transformation to \( S_O \), we expect the two skeletons to be very close to each other (i.e., especially because of discretization the skeletons may not be identical to each other). Therefore, here we use a simpler matching algorithm, detailed below.

**Figure 28**  Invariance under transformation
Of course there are skeletonization algorithms that are dependent upon the orientation of the object. Such algorithms could produce very different skeletons for the original object than for the transformed object and they should be penalized by this measure.

To compute the difference (distance) between two curve-skeletons, \( S \) and \( S_2 \), we compute and sum the differences between corresponding individual curve segments in these two skeletons. This process involves several steps, detailed below:

- Match individual curve segments from \( S \) to individual curve segments from \( S_2 \). This has to be a one-to-one mapping.
- Compute the distance between every pair of matched segments and add them up.

In addition, since the two skeletons may have different number of curve segments, we also add up the length of the segments that are not matched to anything. Consider \( S \) as consisting of \( n \) curve segments: \( s_{1,1}, s_{1,2}, s_{1,3}, \ldots, s_{1,n} \), and \( S_2 \) consisting of \( m \) segments \( s_{2,1}, s_{2,2}, s_{2,3}, \ldots, s_{2,m} \). We match each segment in \( S \) with exactly one segment in \( S_2 \), and no segment in \( S_2 \) can be matched to more than one segment in \( S \). This matching process is based on the minimum distance between the two segments. We match segment \( s_{i,j} \) with segment \( s_{2,j} \) if the distance between \( s_{i,j} \) and \( s_{2,j} \) is minimum among all the distances between \( s_{i,j} \) and all the segments in \( S_2 \). We use the \( \leftrightarrow \) notation to express that two segments are matched.

\[
s_{i,j} \leftrightarrow s_{2,j} \quad (s_{i,j} \text{ matches } s_{2,j}), \quad \text{iff } \quad \text{dist}(s_{i,j}, s_{2,j}) = \min_{k=1}^{m} \{ \text{dist}(s_{1,j}, s_{2,k}) \},
\]

where \( \text{dist}(s_{1,j}, s_{2,k}) \) represents the distance between two skeleton segments. If a segment \( s_{i,j} \) is not matched to anything, we use the notation \( s_{i,j} \leftrightarrow \emptyset \).

Alternatively, we can attempt to match every segment in \( S_2 \) with a segment in \( S \). The details of how to compute the distance between two curve segments are given later in this section. Once the matching process is complete, we can now compute the distance between two curve skeletons, \( \text{SkelDist}(S_1, S_2) \):
The distance between two skeletons, \( \text{SkelDist}(S_1, S_2) \), is the sum of distances between matched curve segments plus the length of the unmatched segments in both skeletons. The distance between two curve segments can be computed as described in the following section, while the length of a curve \( c \), parameterized by arc length, is given by:

\[
\text{length}(c) = \int c ds
\]

Finally, we define the transformation invariance measure, for a specific isometric transformation \( T \), \( \text{TIM}(T) \), as the distance between the two curve-skeletons, \( T(S_O) \) and \( S_{T(O)} \):

\[
\text{TIM}(T) = \text{SkelDist}(T(S_O), S_{T(O)})
\]

Note that this measure of transformation invariance is dependent on the specific transformation used, and it is not normalized. One could use this measure in two different ways: when comparing different curve-skeletonization algorithms, one can use a single transformation \( T \), extract the curve-skeletons using every algorithm for both the original and the transformed object, compute the transformation invariance measure for each algorithm and compare them. Alternatively, one can compute some kind of transformation invariance index that characterizes the invariance of a particular algorithm under any isometric transformation by computing some sort of normalized measure of invariance over a large number of random transformations. While for the first scenario, the value of \( TIM(T) \) can be used directly, for the second scenario, we need a way to characterize an entire distribution of \( TIM(T) \) values, usually done by mean and standard deviation.

In general, there are algorithms which are sensitive to transformations and others that are not.

### 4.4.1 Computing distance between two curves
There are various ways to compute the distance between two curve segments. Let $C_1$ and $C_2$ be two curve segments. Since curve-skeleton segments, are most likely given as a set of samples, we can compute the symmetric Hausdorff distance of these sample distributions:

$$h(C_1, C_2) = \max \left\{ \max_{p_1 \in C_1} \min_{p_2 \in C_2} \{d(p_1, p_2)\}, \min_{p_1 \in C_1} \max_{p_2 \in C_2} \{d(p_1, p_2)\} \right\},$$

where $d(p_1, p_2)$ is the Euclidean distance between two points $p_1$ and $p_2$.

However, the Hausdorff distance does not take into consideration the sequence of these samples, and two very different curves may have a small Hausdorff distance.

Another option is the Fréchet distance, which takes into account not only the location of the sample points on the two curves, but also the order of these samples. The Fréchet distance is defined intuitively as follows: suppose a man walks his dog on a leash. The man and the dog are constrained to follow two different trajectories (the two curves we want to measure the distance between). They can vary their speed independently of each other, but they can only walk forward. The Fréchet distance is the length of the shortest leash the man needs for this walk. Mathematically, the Fréchet distance between two curves $C_1$ and $C_2$ is given by:

$$f(C_1, C_2) = \min_{\alpha, \beta} \max_{t \in [0,1]} \{d(C_1(\alpha(t)), C_2(\beta(t)))\},$$

where $\alpha(t)$ and $\beta(t)$ are two non-decreasing functions defining two parameterizations for the two curves. In other words, the Fréchet distance is the minimum of the largest distance between two corresponding samples on the two curves, over all the possible parameterizations of the two curves.

However, in the case of curve-skeletons undergoing isometric transformations, we do not need to consider all the possible parameterizations of the two curves. Differently from the Fréchet distance, we have to consider decreasing functions as parameterization for either of the segments, but they have to be monotone. In other words, we can walk forward or backward, but not both, on any of the two curves. The reason is that samples on a curve could be taken starting from any one of the two end-points. For example, a curve $C$ could be given as sample points
$p_1, p_2, p_3, \ldots, p_{n-1}, p_n$ or, as $p_n, p_{n-1}, \ldots, p_3, p_2, p_1$. The simplification comes from not having to consider all possible functions as valid parameterizations of the two curves if we are only considering isometric transformations. Since there is no scaling factor in an isometric transformation, we can limit the search to a very small number of parameterization functions $\alpha(t)$ and $\beta(t)$ for the two curve segments $C_1$ and $C_2$ respectively. Below, we present two ways of choosing these parameterizations. Once the set $W$ of parameterizations is chosen (given as a set of tuples $(\alpha, \beta)$), we can compute a pseudo-Fréchet distance (since we are not using all possible parameterizations) ($FDist$) using parameterizations in $W$:

$$FDist(C_1, C_2, W) = \min_{\alpha, \beta \in W} \{ \max_{t \in [0, 1]} d(C_1(\alpha(t)), C_2(\beta(t))) \},$$

or a more “integral” distance ($IDist$) which takes into account every difference between the two curves, not just the largest differences:

$$IDist(C_1, C_2, W) = \min_{\alpha, \beta \in W} \left\{ \int_0^1 d(C_1(\alpha(t)), C_2(\beta(t)))dt \right\},$$

in other words, the distance between the two curves is the minimum sum of distances between corresponding points on the two curves, over all parameterizations in $W$.

We now present the two possible sets of parameterization functions, $W_1$ and $W_2$, one could use to compute the distance between two curves (using either $FDist$ or $IDist$): one that matches full curves to each other, and a second one which also allows part of a curve to match the other curve.

In the first case, we only consider the two possible parameterizations:

$$W_1 = \{ (\alpha_1(t) = t, \beta_1(t) = t), (\alpha_2(t) = t, \beta_2(t) = 1-t) \}, \text{ where } t \in [0, 1].$$

Note that, for the two parameterization choices above we reach the end-points of the two curves at the same time (when $t = 1$). Therefore, as two curves may have different lengths, and we start traversing them at the same time (when $t = 0$), we have different speeds along the two curves. For the first choice $(\alpha_1, \beta_1)$, we walk forward on both curves, while for the second choice
\((\alpha_2, \beta_2)\) we walk forward on curve \(C_1\) and backward on curve \(C_2\). Note also that, the case where one would walk backward on \(C_1\) and forward on \(C_2\), and the case where one would walk backward on both curves are equivalent with the \((\alpha_2, \beta_2)\) and \((\alpha_1, \beta_1)\) respectively.

This first set of parameterization functions is somewhat restrictive, in the sense that we are assuming that the two curves have the same length, by forcing the two parameterizations to reach the end-points of the two curves at the same time (at \(t = 1\)). This penalizes nearly identical curves too much. Consider the case where two curves have different lengths and are such that the shortest one \(C_1\) is a subset of the longer one \(C_2\). As the two curves have different lengths, a point \(P\) common to both curves, will correspond to different values of parameter \(t\) along the two curves. Equivalently, the same value of \(t\) corresponds to different points on the two curves, even in the common section which spans the entire length of \(C_1\). Therefore, even though the two curves coincide over the entire length of \(C_1\), the distance between them (wither \(FDist\) or \(IDist\)) will be large, including a component from the section that is actually common to the two curves.

The second set of parameterizations, \(W_2\), attempts to eliminate this disadvantage by allowing a curve to match only part of the other curve.

\[
W_2 = \begin{cases}
(\alpha_1(t) = \gamma_1(t), \beta_1(t) = \gamma_2(t)) & (\alpha_2(t) = 1 - \gamma_1(t), \beta_2(t) = \gamma_2(t)), \\
(\alpha_3(t) = \gamma_1(t), \beta_3(t) = 1 - \gamma_2(t)) & (\alpha_4(t) = 1 - \gamma_1(t), \beta_4(t) = 1 - \gamma_2(t))
\end{cases}
\]

where

\[
\gamma_1(t) = t \cdot \frac{\text{arclength}(C_2)}{\max\{\text{arclength}(C_1), \text{arclength}(C_2)\}} \quad \text{and} \quad t \in [0, 1].
\]

\[
\gamma_2(t) = t \cdot \frac{\text{arclength}(C_1)}{\max\{\text{arclength}(C_1), \text{arclength}(C_2)\}}
\]

The functions \(\gamma_1(t)\) and \(\gamma_2(t)\) map the value of \(t\) such that we are traversing the two curves with the same speed. As we reach the end of the shortest curve, on the longer curve we have traversed a distance equal with the arc-length of the shorter curve. For example, assuming \(C_1\) is shorter than \(C_2\), as \(t\) goes from 0 to 1, \(\gamma_1(t)\) also goes from 0 to 1, while \(\gamma_2(t)\) goes from 0 to
The last three parameterizations in $W_2$ allow for all the other permutations of end-point correspondence between the two curves.

Using the parameterizations in $W_2$, the shorter curve can match a corresponding length on the longer curve. However, in this case, the remaining un-matched portion of the longer curve should be included in the distance between the two curves as a penalty, otherwise when one curve is a subset of another, their distance will be zero. Thus, when using the parameterizations in $W_2$, the distance between two curves, $C_1$ and $C_2$, should be computed as:

\[
\text{dist}(C_1, C_2) = \text{FDist}(C_1, C_2, W_2) + |\text{arclength}(C_1) - \text{arclength}(C_2)| \quad \text{or,}
\]

\[
\text{dist}(C_1, C_2) = \text{IDist}(C_1, C_2, W_2) + |\text{arclength}(C_1) - \text{arclength}(C_2)|
\]

Implementation note: When dealing with curves represented by discrete samples, finding the point corresponding to a given parameter value $t$ can be achieved by interpolation between adjacent sample points.

### 4.5 Robustness to noise

Robustness to noise of a particular curve-skeleton can be evaluated by comparing the curve-skeleton of an object and the curve-skeleton of the same object with added noise. Let $O$ be a 3D object and $fn(O)$ be the noisy version of the same object, where the noise function $fn$ can be chosen by the application developer according to the specific requirements. This can be uniform noise, Gaussian, Laplacian, or any other noise model. Let $S_O$ be the curve-skeleton of the noise-free object $O$, and $S_{fn(O)}$ be the curve-skeleton of the noisy object $fn(O)$. To quantify robustness to noise, we can simply use the distance between the two skeletons $S_O$ and $S_{fn(O)}$ (defined in Section 4.4) as a measure. The robustness to noise measure, $RM(fn(O))$ is defined for a particular noise model and parameters as:

\[
RM(fn(O)) = \text{SkelDist}(S_O, S_{fn(O)}),
\]
where $SkelDist(S_O, S_{fn(O)})$ is the distance between two curve-skeletons and was defined in Section 4.4.

Note that the robustness measure is related to a specific noise model $fn(O)$ and its parameters, for which the noisy curve-skeleton is constructed. As in the case of invariance under transformation, to quantify the general robustness to noise of a skeletonization algorithm (i.e., under various noise models and parameters), we have to characterize the distribution of these robustness measure values, over an entire class of noise functions. Such a robustness index could be defined as the mean and standard deviation pair of a distribution of robustness measures over a particular class of noise models, or over different models with comparable noise level, etc.

### 4.6 Thinness

Thinness quantifies the dimensionality of the curve-skeleton. According to the definition, a curve-skeleton is composed of a finite set of curves. Since by definition a curve is a sequence of points, we consider it to be as thin as possible. As a result, a curve-skeleton given as a set of curve segments is by definition thin. The only case where thinness of a curve-skeleton can be questioned is when it is represented as a discrete set of voxels. In the following, we will show how to quantify the thinness of a curve-skeleton starting from a voxelized representation.

Let $SV$ be a curve-skeleton given as a set of voxels $v_1, v_2, \ldots v_n$ on a discrete grid, with the usual meaning of adjacency as defined in Section 3.1. The voxels in the curve-skeleton can be grouped into three categories:

- **End voxels** are those voxels that have one or no neighbors
- **Regular voxels** are those voxels that have exactly two neighbors
- **Junction voxels** are those voxels that have more than two neighbors
End voxels correspond to curve end points, regular voxels correspond to curve regular or interior points and junction voxels correspond to end points which belong to more than two curve segments (junctions). The resolution of the voxel grid affects the classification of the curve-skeleton voxels. The problem is that if the resolution is not sufficient, regular voxels can be incorrectly classified as junction voxels. In another instance, if two curve-segments have two points that are closer to each other than the resolution of the grid, they may map to the same voxel and also be classified as a junction. In this case, we have an artificially created junction point, due to resolution, but the curve-skeleton can still be thin (one-dimensional). The real problems arise where entire segments of different curves are closer to each other than the voxel resolution. In that case, many regular voxels of the two curves will be classified as junctions. Consider the following algorithm that identifies the curves of the curve-skeleton from a set of voxels, using (6, 26) connectivity.

```
1 Classify each voxel as end, regular or junction voxel based on the number of face neighbors
2 Mark every regular voxel as not processed
3 Initialize the set of segments $S$ to the empty set
4 $i = 1$
5 For each end or junction voxel $v_e$
6     If $v_e$ has no neighbors
7         Segment $i$ starts at $v_e$ and ends at $v_e$
8     Else
9         For each un-processed neighbor $v_f$ of $v_e$
10            If $v_f$ is a regular voxel
11                Follow the connected path from $v_e$ through $v_f$ until reaching an end voxel or a junction voxel $v_h$.
12                Mark each visited voxel as processed and add it to segment $i$.
13            Else
14                Segment $i$ starts at $v_e$ and ends at $v_f$
15            Insert segment $i$ into $S$
16            $i = i + 1$
```
The above algorithm constructs one curve-skeleton segment starting from an end voxel or a junction voxel. A segment contains a connected path of regular voxels between two end or junction voxels. Let us consider the example in Figure 29, showing two curves in dark grey lines and their discrete representations. The end voxels are colored in blue, regular voxels are colored in grey and junction voxels are shown in red. Note the number of junction voxels (shown as green dots) detected for these two curves, whose continuous representations do not even intersect.

**Figure 29** Junctions, shown as green dots, detected when two discrete curves are too close to each other.

Due to reduced resolution of the voxel grid, a large number of junction voxels are detected close to one another and instead of two segments, the algorithm described above identifies 20 segments, shown as yellow lines connecting the middle of the voxels. Out of those, only 4 have some regular voxels, the rest of 16 start and immediately end at neighboring junction voxels. The 16 segments with no regular voxels create a region where the curve-skeleton is no longer one-dimensional, it has a width. Referring to Figure 29, the section colored in red could be seen as a single segment of width two, and then our skeleton has only 5 segments. However, this is no longer a curve-skeleton, because not all segments are one-dimensional.

As we’ve seen in the example in Figure 29, thinness of the voxelized curve-skeleton is dependent on the resolution of the voxel grid and is related to the number of identified segments
that have no regular voxels or, in other words, to the number of junctions that are face neighbors. Let \( N \) be the number of curve segments with no regular voxels and that have at least one junction voxel as one of their end-points, and let \( NS \) be the total number of segments of the curve-skeleton. We define the thinness index of a voxelized curve-skeleton \( TH \) as:

\[
TH = \frac{N}{NS}
\]

\( TH \) is a normalized measure, with values between zero and one, which measures the number of places where resolution of the voxel grid causes the skeleton to have non-zero width. If \( TH \) is zero, the curve-skeleton is thin; otherwise, it is not thin. The actual value of \( TH \) (when not zero) is an indication of the extent of the non-thin regions with respect to the full curve-skeleton.

Note that a special case is represented by isolated points in the curve-skeleton. These also generate segments that have no regular voxels (the segment start and ends at the isolated point), but these segments can be easily distinguished from the case of junction face-neighbors because their end-points are not junctions.

**4.7 Reconstructability**

Reconstructability (reconstruction) is a measure of how well the curve-skeleton represents the original object in terms of geometry and volume. A simple scheme for reconstruction is to construct a sphere at each curve-skeleton point with a radius equal to the distance transform value recorded at that point. The sphere was chosen because the radius is directly given by the distance transform, and it is easy to construct. However, as discussed in Section 3.3.1, reconstruction can use other primitives in order to recover more of the original object. Regardless of the primitive used in reconstruction, the reconstruction index gives the percentage of the object that was not reconstructed.
In general, the thinner the skeleton, the poorer the reconstruction, that is, from a denser skeleton we can reconstruct more of the original object. Full reconstruction of the original object (using the ball growing approach) is possible from the medial surface, but, in general, full reconstruction is not achievable from a curve-skeleton, which retains only part of the medial surface.

Let $S$ be a curve-skeleton of object $O$, consisting of $n$ curve segments $c_i$, $i = 1,n$. For each curve segment, we define a function along the curve $f_{ci}(s)$, parameterized by arc length $s \in [0, \text{arclength}(c_i)]$, which builds the reconstruction primitive at point $s$ on the curve. In the case of the sphere-based reconstruction, each $f_{ci}(s)$ constructs a sphere centered at point $c_i(s)$, with radius given by the distance transform at that point. For a curve $c_i$, the reconstruction operator $Rec(c_i)$ performs the union of all the reconstruction primitives constructed along the curve $c_i$:

$$\text{Rec}(c_i) = \bigcup_{s=0}^{\text{arclength}(c_i)} f_{ci}(s)$$

The reconstruction operator for the entire skeleton, $Rec(S)$ performs the union of the reconstructed primitives built for all the $n$ segments in the curve-skeleton:

$$\text{Rec}(S) = \bigcup_{i=0}^{n} \text{Rec}(c_i)$$

To quantify the amount of the original object reconstructed by this procedure, we take the following volume ratio:

$$REC = \frac{Volume(O \setminus \text{Rec}(S))}{Volume(O)},$$

where $O \setminus \text{Rec}(S)$ is the difference between the original object $O$ and the reconstructed object $\text{Rec}(S)$.

The reconstruction index, $REC$, gives the ratio of volume of the part of the object that could not be recovered to the total volume of the original object.
4.8 Visibility

Visibility refers to the property of the curve-skeleton that every boundary element (voxel, vertex, or polygon) is visible from at least one location on the curve-skeleton [94][108]. In order to evaluate this property, we need to be able to decide whether a line of sight from a boundary element to a curve-skeleton point is completely included in the object, or it crosses into the background, in which case the boundary element is not visible from that particular curve-skeleton location. If the object is represented by polygons on the surface, a simple line-polygon intersection will provide the answer. If the object is voxelized, we can test whether the line of sight between a boundary voxel and a curve-skeleton voxel crosses a non-object voxel, in which case the boundary voxel is not visible from that curve-skeleton position.

Let $B(O)$ be the set of boundary elements of object $O$ (voxels for a voxelized representation, vertices or polygons for a polygonal representation). Let $S$ be a curve-skeleton of object $O$. Let $\text{visible}$ be a function that tests whether a boundary element $b \in B(O)$ is visible from a curve-skeleton location $c$ i.e., a “visibility” line of sight from $b$ to $c$ does not intersect the boundary of object $O$ anywhere in between $b$ and $c$:

$$
\text{visible}(b, c) = \begin{cases} 
\text{true}, & \text{if } b \text{ is visible from } c \\
\text{false}, & \text{otherwise}
\end{cases}
$$

Let $VB(O)$ be the set of boundary elements visible from $S$ defined as follows:

$$
VB(O) = \{b \in B(O) | \exists c \in S, \text{ such that } \text{visible}(b, c) = \text{true}\}
$$

Then, the visibility index, $VIS$, is defined as:

$$
VIS = \frac{|B(O) \setminus VB(O)|}{|B(O)|},
$$

where $B(O) \setminus VB(O)$ represents the difference between the original boundary and the visible boundary of the object $O$. 

The visibility index measures the ratio of the original object that was not recovered by the reconstruction process.

4.9 Smoothness

A function is said to be smooth if it is continuous and has continuous derivatives up to whatever order is required by the application. For visual smoothness, it is enough if a function has a continuous second order derivative. This is what we mean by smoothness of a curve, rather than how close the curve is to a straight line segment. Figure 30 shows an example. Even though the curve in Figure 30(a) is closer to a straight line, it is not smooth. The sudden changes in direction (discontinuities of the first derivative) are apparent to the naked eye. The curve in Figure 30(b) has no such sudden changes in direction, and even though it is far from a straight line, it looks smooth.

Figure 30 Smoothness. A curve which is not smooth (a); a smooth curve (b).

However, in general curve segments of the curve-skeleton will be given as a set of sample points along the curve, not as a continuous function. As a result, all curve segments of a real curve-skeleton will be defined by a polygonal line determined by adjacent samples, like the curve in Figure 30(a). However, visual smoothness can still be achieved by an appropriate sampling frequency, as shown in Figure 31. The curve shown in black was sampled using two different sampling frequencies resulting in two polygonal line approximations. The red polygonal line uses
only six samples and it is not smooth, while the green polygonal line uses more than forty
samples and looks a lot smoother.

For a sampled curve, the derivatives up to any order can be computed at each sample point
using finite differences from the neighboring samples. For visual smoothness, we require
continuity of the second derivative, i.e. small changes in the second derivative (normal to the
curve) as we go from sample to sample along the curve or, equivalently, that the magnitude of the
third derivative be small enough (a threshold). Choosing the threshold for the magnitude of the
third derivative is application specific and we will not try to provide a way to chose it here. Once
the user decides on a threshold, we can compute the average smoothness of each curve \( C \) defined
as a set of \( n \) sample points \( p_1, p_2, \ldots, p_n \), as:

\[
\text{AverageSmoothness}(C) = \frac{1}{n} \sum_{i=1}^{n} \max \left \{ 0, \left \| C'''(p_i) \right \| - \text{thr} \right \},
\]

where \( \text{thr} \) is the user-defined smoothness threshold, and \( C'''(p_i) \) is the third derivative estimate at
sample point \( p_i \), computed by finite differences, and \( n \) is the number of sample points taken along
the curve.

---

**Figure 31** Visual smoothness of sampled curves. The continuous curve shown in black is
sampled using six samples (red polygonal line) or more than forty samples (green
polygonal line). The green curve appears smoother than the red curve.
However, average smoothness as defined above can be biased toward curves with long straight segments. If on a curve the number of samples where there is no change in the second derivative greatly outnumbers the number of samples where there is a change, even if the change is large in those few samples, taking the average over all the samples will reduce the effect of these few samples on the resulting average smoothness measure. An alternative is to only consider those samples where a change in the second derivative (normal to the curve) can be observed. Let $M$ be a subset of the $n$ sample points $p_1, p_2, \ldots, p_n$ taken along curve $C$, such that at each sample point in $M$ there is a non-zero change in the second derivative along the curve (when compared to the previous or the next sample point on the curve). We define minimum smoothness of a curve $C$ as:

$$MinimumSmoothness(C) = \frac{1}{|M|} \sum_{p \in M} \max \left(0, \left\| C''(p) \right\| - thr \right)$$

Although this measure should give higher values of the smoothness measure for a curve compared with average smoothness, we call it minimum smoothness because a higher value of the smoothness measure actually means more change in the second derivatives along the curve and thus less smoothness.

The smoothness $SM$ of the entire curve-skeleton $S$, consisting of $m$ curve segments $C_1, C_2, \ldots, C_m$ is then the average smoothness over all the individual segments, computed either as minimum or average smoothness:

$$SM_{\text{average}}(S) = \frac{1}{m} \sum_{i=1}^{m} AverageSmoothness(C_i)$$

$$SM_{\text{minimum}}(S) = \frac{1}{m} \sum_{i=1}^{m} MinimumSmoothness(C_i)$$

Implementation note: the second derivative along a curve (computed as variation of the first derivative) is a vector, which gives the normal direction to the curve. The third derivative can
then be computed as the change in the normal direction as we move from sample to sample along the curve.

To achieve smoothness, most algorithms smooth the points of their resulting skeletons by fitting a spline.

4.10 Properties of the curve-skeletonization process

The properties associated with the curve-skeletonization process cannot be quantitatively evaluated by simply inspecting the resulting curve-skeleton and the original object. Since the evaluation will have to include an analysis of the algorithm itself, we cannot provide algorithms that produce an evaluation of these properties, but we discuss how one should approach the evaluation of these properties.

4.10.1 Efficiency

Efficiency can either be measured as running time of a specific implementation of an algorithm, or as algorithmic complexity. While the second alternative is more desirable for the general case, the direct comparison of running times can be also be useful in a practical situation.

4.10.2 Hierarchy

A hierarchical (multi-scale) algorithm must have at least one parameter that can be changed. Its effect should be that the resulting curve-skeleton will have different complexity according to the parameter setting. We define complexity of the curve-skeleton in terms of the number of elements that composes it: curves, points, or voxels. If the algorithm has no parameters, or if the parameter does not affect the complexity of the resulting curve-skeleton, it cannot be hierarchic.

If curve-skeletons of different complexities can be generating by varying the parameters of the algorithm, hierarchy can be checked by a simple inclusion test. Let us assume that an algorithm \( A \) takes a parameter \( k \). Let us assume that for a small value of \( k \), it produces a curve-
skeleton with a certain number of curves \( A(k) \) (points, voxels, etc.) and as the value of \( k \) is increased, the resulting curve-skeleton has more and more curve segments (points, voxels, etc.). A strict hierarchy is achieved if for any \( k_2 > k_1 \), we have \( A(k_1) \subseteq A(k_2) \), in which case, the algorithm extracts a curve-skeleton set (see definition in Section 3.6). If the strict inclusion condition above is not achieved, the algorithm is not hierarchic, even though it can produce curve-skeletons of different complexities.

While this property can be checked using the above approach, it is usually easier to determine whether an algorithm has the hierarchy property by analyzing the algorithm itself.

### 4.10.3 Junction detection and component-wise differentiation

Junction detection is the ability to accurately detect junction points in a curve-skeleton. We define junction detection of an algorithm as the ability of the algorithm to detect the junctions before or during the skeleton extraction process (see Section 3.4.3). In conjunction with junction detection, component-wise differentiation is always possible once a set of junctions has been determined. Detecting junctions as a post-processing step, after the full skeleton is computed, may not even be possible in certain cases. For example, if the curve-skeleton is given as a set of curve-segments (sampled), detecting junctions is straight-forward: the end point of each segment is a junction if it is shared with other segments. However, when the curve-skeleton is given as a set of voxels, junctions may not be determined accurately, based on the resolution of the voxel grid, as discussed in Section 4.6. As it is the case for thinness, the existence of junctions that are direct neighbors to each other is not desirable because cannot distinguish the case where such junctions are legitimate junctions or are simply an artifact due to insufficient resolution of the voxel grid. Thus, for a voxelized curve-skeleton, if the curve-skeleton is not thin, junctions cannot be determined accurately as a post-processing step.

Given the subjective nature of object decomposition, an algorithm which validates the detected junctions as being “the correct junctions” is difficult or even impossible to develop.
However, one could approach this from the perspective of stability of the detected junctions, under transformations or noise. Such an approach would be different from the definition given here and it may even be possible to evaluate it directly on the resulting curve-skeletons.

4.10.4 Ability to handle different object representations

The ability to handle different object representations can only be evaluated by analyzing the algorithm itself. For example, an algorithm that pre-computes the values of a function everywhere inside an object clearly needs to work on a discrete grid, but may be able to handle the other object representations by simply mapping them onto the grid.

In evaluating this property, one should consider not only the polygonal and voxelized representations of 3D objects, but also the representation by unorganized point sets. While conversion from voxelized to polygonal and vice-versa is a relatively easy and consistent process, converting a point set to a polygonal model involves the discovery of the missing connectivity information.

4.11 Future work

Several directions for future investigations can be drawn. First, the evaluation framework presented here is by no means complete or final. It could certainly be extended with additional properties which could be essential to new applications. As part of the effort to increase awareness in the community about this evaluation framework, one could develop a skeleton algorithm benchmark and repository, where developers can submit their new algorithms and automated tests would be performed to produce evaluation scores for each of the properties discussed here (and possibly others). Then, users of curve-skeletons can select the appropriate implementation based on the test scores achieved for the properties relevant to their application.
Chapter 5

Existing Methods for Extracting the Curve-skeleton

In this chapter we will review previous work in curve-skeleton extraction. There are many different skeletonization algorithms for both 2D and 3D. Although some of the 2D algorithms reportedly scale to 3D, we restrict our discussion to algorithms explicitly designed for 3D. For skeletonization work in 2D, the reader is referred to the work of Lam and Lee [118] as a starting point. The discussion below reviews general 3D curve-skeletonization algorithms, i.e., the generation of a 1D curve-like representation from a 3D object. However, for completeness we do include some medial surface algorithms since these medial surfaces could be further reduced to a curve-skeleton by applying the methodology described in [200][226].

Object representation is of course an important issue. Some algorithms work on objects represented by voxels on a regular grid (e.g., thinning) while others operate exclusively on a polygonal representation (e.g., Voronoi-based methods). In general, every algorithm is restricted to a given representation in order to allow for an efficient implementation. Conversions from one representation to another can be performed relatively easy using standard algorithms such as “marching cubes” [142] used to extract the surface from a voxel representation, or voxelization [221] to convert a polygonal model into a voxel representation. In this work we are concerned mainly with discrete representations of 3D objects, specifically representation by voxels on a cubical regular grid. However, for completeness we will also discuss some algorithms that operate on continuous representations (polygonal meshes).

A commonly used classification scheme present in the literature divides the skeletonization algorithms into the following classes [150][230]: topological thinning (grassfire propagation),
distance transform based (ridge detection) and Voronoi diagram based. However, many of the surveyed methods that produce curve-skeletons use pieces from several classes listed above to obtain a curve-skeleton. For example, there are thinning algorithms which use the distance field information to determine the thinning order, or some distance field methods which use thinning to prune the skeleton. Instead, we categorize the algorithms based on the underlying implementation into the following classes:

- thinning and boundary propagation,
- distance field based,
- geometric, and
- general-field functions.

We will discuss each of these methodologies in detail in the following sections. In addition, we discuss their advantages and disadvantages with respect to the desirable curve-skeleton properties presented previously. To further illustrate our evaluation of curve-skeletons methodology, we implemented a “core” algorithm for each of these classes and we evaluate these implementations with respect to the properties. The purpose of the comparison between these methodologies is to give a sense of how close they get to each of the desirable properties after the first step of processing (the “core”).

5.1 Thinning and Boundary Propagation

Thinning methods attempt to produce a curve-skeleton by iteratively removing voxels from the boundary of the object. These methods operate on 3D objects represented by voxels on regular or other types of grids and rely on the concept of simple point, introduced by Morgenthaler in 1981 [160]. A simple point is an object point which can be removed without changing the topology of the picture [17][112][113]. Simple points can be locally characterized, that is, one can determine if a point is simple or not by just inspecting the local neighborhood of the point. A digital picture
P is given by specifying the connectivity used for the object (m) and the background (n) and the set of voxels (black voxels) that belong to the object (B): \( P = (\mathbb{Z}^3, m, n, B) \) (see Section 3.1).

Following the criterion given in [112], a black point \( p \) (voxel) of a 3D picture \( P = (\mathbb{Z}^3, m, n, B) \) is

**simple** if the following conditions hold:

1. \( p \) is m-adjacent to only one m-component of \( N_{26}(p) \setminus \{B \setminus \{p\}\} \)
2. \( p \) is n-adjacent to only one n-component of \( N_{26}(p) \setminus B \)
3. \( \chi((\mathbb{Z}^3, m, n, B \setminus N_{26}(p))) = \chi((\mathbb{Z}^3, m, n, (B \setminus \{p\}) \setminus N_{26}(p))) \)

where \( \chi(P) \) is the 3D Euler characteristic of an image \( P \) defined by:

\[
\chi(P) = (\text{number of object components in } P) - (\text{number of holes (tunnels) in } P) + (\text{number of cavities in } P).
\]

For (26, 6) pictures, the simple point criterion can be formulated as follows [145]:

A black point \( p \) is simple if:

1. \( p \) is 26-adjacent to only one object component in \( N_{26}(p) \).
2. \( p \) is 6-adjacent to only one background component in \( N_{18}(p) \).

Thinning algorithms attempt to iteratively remove simple points from the boundary of the object until no more points can be removed. Since simple points can be characterized by only inspecting the 26-neighborhood, the simple point test is usually implemented as a set of templates (or masks), of size 3x3x3 or larger. The center of a mask is placed on the point being tested and covers its entire 26-neighborhood. Each of the points in the mask has a value of “0”, “1” or “don’t care”. A value of “0” must match a background voxel, a value of “1” must match an object voxel, while a “don’t care” can match either a background or an object voxel. In order to reduce the number of templates, additional notations are introduced such as: “the points marked z must match different types of points” (one must be white and the other one black) [176].

Removing all simple points from an object might simplify the object too much. For example, any object consisting of a single connected component, with no holes and no tunnels can be
reduced to a single point by sequential removal of simple points. This is not a desirable result of a skeletonization algorithm. In order to maintain the geometrical properties of the object, additional conditions are used to prevent removal of certain simple points, named surface end points or curve end points. Many definitions of surface and curve-end points have been proposed in the literature [234][160][123][176]. The following definitions for curve and surface end-points for (26, 6) pictures are given in [175]:

A black point \( p \) is a curve-end point in a picture \( (\mathbb{Z}^3, 26, 6, B) \) if the set \( (N_{26}(p) \cap B) \setminus \{p\} \) is singleton (i.e., \( p \) is 26-adjacent to exactly one black point).

A black point \( p \) is a surface-end point in a picture \( (\mathbb{Z}^3, 26, 6, B) \) if the set \( N_6(p) \setminus B \) contains at least one opposite pair of points (i.e., two white points in \( N_6(p) \) are on opposite sides of \( p \)).

These conditions are included in the set of templates used to test whether a point can be removed or not during thinning. Some thinning methods produce a surface-skeleton in the first stage by preserving surface-end points and continue to thin until a one voxel thick skeleton is obtained [30][234]. Others directly produce a curve-skeleton by explicitly preserving curve-end points only.

Figure 32 shows the thinning process on a 2D shape. At each iteration, the boundary points are marked with “B” and if they are simple points, they are removed before the start of the new iteration. At the last iteration, no points can be removed.

The definition of a simple point is enough to guarantee that removing one point at a time will preserve the topology of the original picture. However, thinning algorithms usually attempt to remove more than one voxel in a single pass and in this case preservation of topology may be compromised. An example is the case of a 2x2x2 cube. Every point in this configuration is simple, but removing all points at the same time will completely delete the object, altering the topology of the image. In [151], Ma established and proved the sufficient conditions for a 3D parallel thinning algorithm to preserve topology of any 3D picture and showed that only a limited number of configurations need to be checked in order to prove this property. Several approaches
have been proposed to deal with the problem of simultaneously removing simple points:

**Directional or border sequential thinning** methods remove voxels only from one particular direction in each pass. Several approaches have been proposed here:

- using 6 directions: Up (U), Down (D), North (N), South (S), East (E), and West (W), corresponding to the six faces of a voxel [84][123][146][234].
- using 12 directions: UN, UE, US, UW, NE, NW, ND, ES, ED, SW, SD, and WD (UN stands for Up-North), associated with the 12 edges of a voxel [176][139].
- using 8 directions, corresponding to the 8 corners of a cube: USW, UWN, UNE, UES, DSW, DWN, DNE, and DES (USW stands for Up-South-West) [175].

These methods are sensitive to the order in which the different directions are processed and the resulting skeletons may not be centered within the object, in addition to being sensitive to object transformations.

**Subfield sequential thinning** methods divide the discrete space into several subsets named subfields and at each sub-iteration only voxels belonging to a one of the subfields are considered for deletion. Different number of subfields can be used in 3D: 2, 4 or 8 [147]. Two voxels \( p = (p_x, p_y, p_z) \) and \( q = (q_x, q_y, q_z) \) are in the same subfield if:
• \((p_x + p_y + p_z) \equiv (q_x + q_y + q_z) \pmod{2}\) – for the 2-subfield case (i.e., two voxels are in the same subfield if they share an edge) [148][92][149].

• \((p_x + p_y) \equiv (q_x + q_y) \pmod{2}\) and \((p_y + p_z) \equiv (q_y + q_z) \pmod{2}\) – for the 4-subfield case (i.e., two voxels are in the same subfield if they share a corner) [92][147][145].

• \(p_x \equiv q_x \pmod{2}, \ p_y \equiv q_y \pmod{2}\) and \(p_z \equiv q_z \pmod{2}\) – for the 8-subfield case (i.e., two voxels in the same subfield are not 26-adjacent) [17][196].

**Fully parallel** thinning methods [60][145][154][140] consider all boundary points for deletion in a single thinning iteration. In order to achieve topology preservation, the neighborhood that needs to be inspected when deciding whether a point is deletable or not needs to be extended past the immediate 26 neighbors.

Several other algorithms which combine elements from this classification exist. The algorithm presented in [30] is partly parallel, partly directional. It uses a parallel approach to reduce the object to an at most two-voxel thick medial surface, then uses a 6-directions thinning to further reduce this medial surface to one voxel thickness. In a third step, the one voxel thick surface is reduced to an at most two voxel thick curve-skeleton and in a final step to one voxel thickness using again a combination of parallel and sequential sub-steps.

A fourth class of algorithms use a **non-directional** approach. In [226], a thinning method aimed specifically at surface-like objects is presented. The algorithm classifies object voxels as edge, inner, curve and junction, then iteratively removes edge voxels and re-classifies inner and junction points. This algorithm provides a solution to further reducing medial surface representations to curve-skeletons.

Most thinning algorithms are designed and proven correct for a specific connectivity such as (26, 6) or (18, 6) and need extensive changes to accommodate other connectivities. The correctness proof deals with preservation of topology.
5.2 Using a distance field

The distance transform or distance field is defined for each interior point $p$ of a 3D object $O$ as the smallest distance from that point to the surface $\text{Surf}(O)$ of the object:

$$D(p) = \min_{q \in \text{Surf}(O)} (d(p, q)),$$

where $d$ is some distance metric.

Various distance functions can be used such as the Euclidean distance or an approximation such as the $<3,4,5>$ chamfer metric [31][189]. A distance field can also be approximated using fast marching methods [195][208][230][231]. Figure 33 shows the color-coded distance field values on a slice of a 3D chess piece shape. The color map ranges from blue for small distance field values to red for large values.

![Figure 33 Color-mapped distance field of a chess piece.](image)

The distance field-based methods can accurately extract the medial surface as the ridges of the distance field; however they cannot extract a curve-skeleton from arbitrary objects without employing additional techniques to prune the medial surface, techniques which cannot be based on the same distance field. For example, for the box in Figure 1(b) the voxels along the center plane (shaded) all have the same distance field value. As the distance field does not offer any additional information to help in pruning these candidate voxels, other techniques such as thinning [226] must be used to extract a line from this plane.

A first sub-class of algorithms contains the distance field ordered thinning approaches [32][53][189], where a thinning algorithm uses a priority function computed over the distance
field to select candidate voxels for removal. In [53], the priority function takes into consideration also the slope of the distance function in the vicinity of a dynamically detected skeleton point. In [32], the divergence of the distance field gradient is used in the priority function and a user-defined threshold blocks the removal of simple points with lower divergence value.

A second sub-class of algorithms attempt to find the local maxima and ridges in the distance field which correspond to voxels that are locally centered within the object (i.e., the centers of maximal inscribed balls). These act as potential candidates (from the larger pool of object voxels) for curve-skeleton points. The candidates must then be somehow “pruned” or thinned to produce a 1D skeleton. The result is almost invariably a disconnected set of voxels which then need to be connected using a path connection or minimum spanning tree approach [223][240][252]. Therefore, most of the algorithms have three steps:

1. find ridge points (local maxima, saddles),
2. prune and connect (not necessarily in this order).

Several methods can be used to find candidate voxels: local maxima in the distance field are selected as initial candidates in [94]. In a second step, another set of candidates is chosen as the voxels with maximum distance field value in a set visible from a boundary point that was not yet covered. In [19], gradient searching involves detecting neighborhoods of non-uniform gradient and flagging those points as candidate voxels. Non-uniform gradient regions are defined as regions where any of the gradient vectors differs by more than 90 degrees from the average vector. In an algorithm specifically designed for the tubular branching structure of the human trachea [180], candidate voxels are local maxima of the distance field detected on the geodesic front propagating from the root of a trachea. Another function computable from the distance field is the bisector angle [152][53] or separation angle [67]. This is defined as the maximum angle formed between the center of an inscribed ball and any pair of contact points on the surface of the object. In the continuous case, voxels that are on the medial surface are centers of maximal inscribed balls which should touch the boundary in at least two points, and so the corresponding
bisector angle would be non-zero. The discrete version of the bisector function includes contact points of discrete balls in the neighborhood of the considered voxel (extended downstream) as defined in [53]. Thresholding the bisector angle alone [209] or in combination with other quantities [152] can also produce a set of candidate voxels. A thinness parameter based on the difference from the average distance value in the 26-neighborhood of a point is used to select candidate voxels in [74] and [75]. In voxel coding approaches [253], the distance field is combined with a distance-from-a-source field. Every different value in the distance-from-a-source field defines a cluster of voxels; the centers of these clusters chosen to be the voxel with the maximum distance field value comprise the candidate skeleton voxels. In [195], the distance field is combined with several distance-from-a-source fields generated using fast marching fronts of various speeds. The boundary surface shrinking approach described in [203], uses the gradient of the distance field to propagate the polygonal boundary of an object towards the medial surface of the object. The resulting mesh is then simplified by combining small edges and clustering mesh vertices.

The resulting candidate voxels are usually disconnected, therefore the next step is to prune and connect them. Pruning is an optional step which is usually performed to eliminate small branches in the resulting skeleton. This can be a pre-, post- or during connection process. For example in distance field-based thinning algorithms, connectivity is explicitly maintained while removing voxels. In [239], sphere coverage of a path tree is used to “weed out insignificant extreme points” while building the path tree. In [94], visibility is used to reduce the set of candidate voxels before they are connected. Candidates that do not add any new voxels to the current visible set are discarded. In [19], pruning of candidate voxels is performed after connection by extracting shortest paths from the connected set based on a penalized-distance field. Thinning can also be used to prune connected components of candidate voxels before enforcing connectivity, as in [230]. In [223], clustering is used to reduce the number of candidate voxels before connecting them in a graph structure.
After the pruning step, the remaining voxels are usually disconnected and the final step involves re-connecting them in order to produce a set of 1D curves. For connectivity, most use minimum spanning trees [74][223][240], shortest paths [94][239][253], other graph algorithms, or even straight lines between connected skeleton components [230]. In [252] an “LMpath” defines the connectivity of local maxima clusters, while in [19], the first connection step is performed by following the gradient of the distance field starting at each candidate voxel until reaching the already connected set. In [74], connection is performed using a midpoint subdivision scheme. Given two end-points, the algorithm identifies the closest candidate to the midpoint of the segment connecting them, and then the same method is recursively applied to the two new pairs of end-points (old end-point one with the midpoint, and midpoint with the old end-point two). The two initial end-points are selected by the user from the initial set of candidates.

Some methods may first connect the set of candidate voxels and then prune it by extracting shortest paths from the connected set [19], others explicitly maintain the connectivity while pruning, combining the two steps. An alternative is provided by the fixed topology skeleton which is a set of a fixed number of connected active contours driven by the underlying distance field [83].

A 2.5D method presented in [200] starts from a medial surface representation of 3D objects and, for each surface point, it computes the distance to the boundary of the surface. The candidate voxels (centers of maximal geodesic discs) are connected using gradient guided path growing.

The main advantage of these methods is that computation of the distance field is very fast [199] and it is usually needed by the underlying application for further processing. Furthermore, for tubular objects, the distance field approach works very well.
5.3 Geometric methods

Geometric methods usually apply to objects represented by polygonal meshes or scattered point sets in continuous space. A popular approach is to use the Voronoi diagram [34][170][171][172] generated by the vertices of the 3D polygonal representation or by a set of unorganized points [5][8][9][164] sampled from the true object surface. The Voronoi diagram represents a subdivision of the space into regions that are closer to a generator element (a surface sample in the case of a 3D model) than to any other such element. As the density of surface samples increases, the circumscribed spheres associated with the Delaunay tessellation (the dual of the Voronoi diagram) become tangent to the surface of the object, exactly as in the definition of the medial surface. As a result, the interior faces of the Voronoi diagram approach the medial axis as the density of surface samples increases. Note that the Voronoi-based methods can approximate the medial surface of a 3D object and that further processing is necessary to extract a curve-skeleton from this representation. For example, in [9], a criterion based on the perimeter of the boundary faces (i.e., faces that are shared with the boundary of the object) of the Delaunay tetrahedron corresponding to a Voronoi vertex and the radius of the Delaunay sphere centered at the vertex is used to remove vertices from the Voronoi medial surface until a curve-skeleton representation remains (named “wireframe skeleton” in [9]). In [244], a number of “domain balls” (non-intersecting maximal inscribed balls) are detected, whose centers are later connected into a curve-skeleton using surface connectivity information. In [56], a curve-skeleton is computed by eroding the Voronoi diagram representation using the medial geodesic function as a priority function.

Other methods attempt to directly identify the medial surface of a 3D polyhedral by tracing the seams of the medial surface: [55][190][204][212][248].

Cores and M-reps [40][185][182][161] are multi-scale medial-axis/surface approaches. In [40], boundariness detectors of different scales connect to others at the same scale via cores, over
a distance which is proportional to their scale, and in a direction perpendicular to the detected boundary direction. The boundariness detectors vote for each position in scale space, creating a “medialness excitation” field. A core is a ridge in this scale space excitation field. The location of the core represents the middle of the figure and the spread of the core represents the width of the figure [40]. M-reps were designed as an alternative shape representation to b-reps (boundary representations). The M-rep models the medial surface using a “web” of connected medial atoms. Each medial atom has a position, and additional information describing the shape locally, such as: width (with a certain tolerance), local figural frame which implies the figural directions, and an object angle between opposing, corresponding positions on the implied boundary [185]. The information associated with each medial atom implies a boundary of the object (within a given tolerance). However, extracting the medial atom based representation from already existing objects (b-reps) involves computing a medial surface and then manually pruning it and defining the m-rep mesh.

A similar structure is the shock scaffold, which relies on the concept of contact spheres [129][130] and represents the medial axis/surface by a set of shock curves (a graph), defined as the intersections and edges of medial surface sheets. Although this representation consists of a set of curves, it is not a curve skeleton. While some of the curves describing intersections of medial axis sheets should be part of the true curve-skeleton, the representation will extract the edges of a medial surface patch rather than a centered line inside this patch (the red edges in Figure 1(b) are the components of the shock scaffold [130]).

The methods described above can be labeled as medial-axis/surface based. The main disadvantage of medial-axis based geometric methods is their sensitivity to noise. For example, Amenta's power shape [5] contains a large number of unwanted branches that need to be pruned to extract a simple skeleton [8]. Measures such as the bisector angle (see Section 5.2) or the radius of the maximal inscribed ball centered at a medial axis point are used to assess its stability [5][8][9] and filter the noisy branches of the Voronoi-based skeleton. Additionally, these methods
are more computationally intensive than the thinning/distance field based methods. Furthermore, most of them produce something closer to the medial surface not a 1D curve-skeleton. Nevertheless, we mention them here because reducing a surface skeleton to a curve-skeleton is possible using the curve thinning algorithms presented in [200][226].

There are other geometric methods that avoid the medial axis altogether. **Reeb graph** based shape descriptors, with roots in Morse theory, are 1D structures encoding the topology and geometry of the original shape. The Reeb graph captures the topology of a compact manifold by following the evolution of the level sets of a real-valued function defined on the respective manifold. More formally [20]:

Given a real-valued function \( f \), defined on a compact manifold \( M, f: M \rightarrow \mathbb{R} \), the Reeb graph is the quotient space of \( M \times \mathbb{R} \) defined by the equivalence relation \( \sim \) given by:

- \( (x_1, f(x_1)) \sim (x_2, f(x_2)) \) iff \( f(x_1) = f(x_2) \) and
- \( x_1 \) and \( x_2 \) are in the same connected component of \( f^{-1}(f(x_1)) \) (or \( f^{-1}(f(x_2)) \)).

In other words, the equivalence classes defined by \( \sim \) consist of the connected components of the level sets of \( f \). Nodes in the Reeb graph correspond to the critical points of the function \( f \), (i.e., points where the gradient of \( f \) is zero) and edges of the graph represent connections between critical points. According to Morse theory, the object changes its topology only in connection to the critical points of \( f \), so the edges of the Reeb graph can be thought of as representing different components of the object and the nodes can be regarded as connections between these different components.

The Reeb graph is not a curve-skeleton: it is not even defined in the same space as the original object. However, an embedding of the Reeb graph into 3D space can be attempted by mapping each edge into a sequence of 3D points defined for example as the centers of the successive level-set contours associated with the respective edge. This defines a curve-skeleton for the original object [122]. Figure 34 shows an example for a 2D shape. The height function was used in the figure to define the level sets. The centroids of the connected components of each
Figure 34  An embedding of the Reeb graph into the original image space. The height function was used to define the level sets. Each node is taken to be the centroid of its corresponding contour. Nodes from corresponding contours in adjacent level sets are connected with a straight line.

level set are taken to be the nodes of the graph (marked with black dots on the image), and are connected using straight lines (the red lines).

Extensions of the Reeb graph to polygonal meshes have been proposed such as the Extended Reeb Graph [10][20] or the Discrete Reeb Graph [247]. The choice of the real-valued function $f$ distinguishes the various algorithms. The height function used in [10][247] is sensitive to the object’s orientation. The Euclidean distance from a point in space, usually the barycenter of the mesh (used in [21][22]) does not depend on the orientation of the object, but it is affected by changes in the underlying mesh as a result of articulated motion. Using the geodesic distance instead of the Euclidean distance solves this problem but the source point must now be a vertex of the mesh. The shortest path distance to a given source point [122][235] is sensitive to the selection of the source point. Alternatively, the integral of the geodesic distance to all the points on the surface [96] is less sensitive to small perturbations due to noise. Using the shortest geodesic distance to a set of curvature maxima on the surface [162] introduces the problem of robustly and automatically detecting relevant curvature maxima on the surface. A similar approach is given in [37], where curve-skeletons generated from multiple extremities are combined into a final one.

Other geometric methods do not rely on a function defined on the manifold to produce a curve-skeleton. Li et al. [134] construct a line segment skeleton by collapsing edges in length
order (shortest first). This method is sensitive to the mesh tessellation. Katz and Tal [107] first decompose a mesh surface into segments using clustering, and then use this segmentation to construct a skeleton (joints are represented by centered vertices at the boundary between different patches). In [136], the curve-skeleton and the mesh decomposition processes are interrelated. The curve-skeleton of each component is obtained by connecting the principal axis with the centroid of each opening (connection to other components). In turn, the mesh decomposition process decides whether or not to further decompose an existing component by evaluating the quality of its curve-skeleton.

5.4 General-field functions

The distance field based methods described in Section 5.2 use a distance function (distance to the closest point on the boundary) to compute a scalar or a vector field which is then used to extract the curve-skeleton. However, various other types of functions can be used to generate a field and extract a curve-skeleton.

The generalized potential function [3][43][48] uses a generalization of the Newtonian (gravitational) potential field. At each point interior to the object, the potential is determined as a sum of potentials generated by charges placed on the boundary of the object. In [43], the polygons of the mesh representation are the source of potential ([3] is the 2D version of [43]). Seed points are selected near the convex corners of the object and the repulsive force (gradient of the potential field) is analytically derived only along a path determined using a force-following algorithm. Each path started at a seed point ends at a potential minimum detected by a major change in the repulsive force vector direction. The resulting skeleton segments are usually disconnected pieces and a separate re-connection step is necessary to assure connectivity. In [48], each boundary voxel is a source of potential for discrete objects. Seed points in this case are chosen as critical points and low divergence points of the repulsive force vector field.
The **visible repulsive force function** used in [245] and [138] is a special case of the potential field function used in [3][43][48]: the Newtonian repulsive force. The same force-following approach starting at each mesh vertex is used to identify the local minima in the field which are then connected with straight lines considering the adjacency relations of the objects polygon mesh. In addition to [43] and [48], the visibility of boundary elements from an interior point is taken into considerations by computing the intersection with a number of sampling rays originating at the current position on the path. In [138], the same visible repulsive force field is computed over a discretization of the object and is used to define a priority function for a thinning step. The method needs additional clustering and re-connection steps to produce a curve-skeleton for animation.

In [2] and [90] an **electrostatic field function** is used for skeletonization of 2D shapes. The electric potential is generated by the surface of the object. In [2], significant corners are detected on equipotential contours by looking for local minima and maxima of the field along the contour. Next, corners of different contours are connected by tracing field lines.

**Radial basis functions** are used in [150]. The vertices of the objects mesh are centers for radial basis function level sets. Next, a gradient descent algorithm is used to trace each vertex to a local maximum in the field. The local maxima are then connected by minimizing the energy of an active contour initialized using the adjacency relations of the original mesh.

The curve-skeleton is extracted by detecting the local extremes of the field and connecting them. A force following algorithm can be used for connectivity, using the mesh vertices [245], the convex corners of the mesh [3][43], the significant corners along equipotential contours [2], or the critical points of the vector field [48] as “seed” points. Another possibility is the use of an active contour to detect the final location of the curves connecting the extremes in the field [150][244]. Detection of local extremes can be achieved explicitly by looking for critical points over the entire underlying vector field [48] or detecting the local maxima along equipotential contours [2]. The other methods directly use the force-following algorithms starting at other seed
points, using the fact that the force-following algorithm stops when it reaches an extreme. Figure 35 shows the vectors of the repulsive force field of a 2D shape.

![The repulsive force field of a 2D shape. The vectors at each location are shown as small arrows. The inset shows what region of the force field was magnified.](image)

**Figure 35** The repulsive force field of a 2D shape. The vectors at each location are shown as small arrows. The inset shows what region of the force field was magnified.

The main advantage of these functions over the distance field is that they can produce nice curves on medial sheets where the distance field is constant. This is because the underlying field takes into account larger boundary areas, not just the closest point as in the case of distance fields. This also creates an averaging effect that makes these algorithms less sensitive to boundary noise. However, this comes with a price. Since the field value at each point is now influenced by more points on the boundary, these methods are computationally more expensive. For methods operating in discrete space such as [48] and [138], resolution of the voxel grid also affects the field functions which tend to be more sensitive to noise in thin regions of the object. In such thin regions, the final field value at an interior point is primarily determined by fewer boundary voxels, the rest of the object boundary being too far away to make a significant contribution. Given that the field value is now actually determined by fewer boundary points in the local neighborhood, any perturbation of this boundary will have a non-negligible effect on the resulting field value, in spite the averaging effect specific to such field functions. Another disadvantage of
the general field functions is their numerical instability since the computations usually involve
first or even second order derivatives.

5.5 Discussion

The classification proposed in the previous section was based on the first steps of the
implementation, but we would like to emphasize that many methods use concepts from two or
more different classes to finally produce a curve-skeleton. The distance field-based thinning
procedures described in [53][75][189] initially compute a distance field over the object, but then
employ thinning to extract a curve-skeleton, using the distance field as a priority function to
select voxels that will be removed next. In [244], a control skeleton named the domain connected
graph (DCG) is extracted in several steps involving Voronoi diagrams, Reeb graphs and repulsive
force fields.

Additionally, concepts from different classes are actually closely related. For example, we
could relate the single-source distance field (voxel-coding) algorithms [195][253] with the Reeb-
graph approaches [122][235] since the minimum distance search employed in single-source
distance field methods actually involves the traversal of the level sets of a distance function
defined on the object (in this case, on the full solid 3D object rather than only its surface). Surface
shrinking along the gradient of the distance field [203] is also similar to surface thinning
(removing layers from the boundary of the object until no more layers can be removed).

Next, we discuss the described curve-skeletonization methods in terms of the properties
presented in the previous chapters. Some of these properties can be achieved by a post-processing
step, after the curve-skeleton is first computed. Such is the case for connectivity, visibility,
smoothness or junction detection. The discussion here focuses on the quality of the curve-
skeleton obtained without the use of post-processing.
5.5.1 Centeredness

Thinning and general field methods do not guarantee centeredness. For example, for a directional thinning algorithm, this would depend on the order in which the different directions are processed. In the case of general field methods, centeredness depends on additional parameters, but since they take into account a larger surface area that the two closest points, centeredness is usually compromised. For example, for the general potential function \([43][48]\), the strength of the field influences the centeredness of the resulting skeleton.

Methods using a distance field can better achieve centeredness because of the centeredness information included in the distance field. However, once clustering and spanning trees are used, centeredness may be lost (see for example [239]). Distance field methods using a distance to source (for example geodesic field propagation) do not generate centered curve-skeletons. The problem is described in detail in [180] and the solution provided there combines the distance-to-source field with a global distance-to-boundary field to achieve centeredness.

Geometric methods directly computing contact points (points where the maximal inscribed spheres touch the surface) [130] can also achieve centeredness since these points can be incorporated more easily into the pruning steps. Voronoi-based methods are dependent on the sampling density of the object’s surface: a denser sampling produces a more centered curve-skeleton [5] but the running time increases. Centeredness of level-set [235] and mesh decomposition-based [107] curve-skeletons is poor because centroids are directly connected with straight-line segments, regardless of the configuration of the object between these points. Centeredness problems arise especially in regions where the topology of the object changes between successive level sets and it is also influenced by the resolution (the distance between two successive level sets).

Resolution affects any centeredness measurement in the discrete domain. Using the same example of a shape whose width is an even number of voxels, if this shape is reduced to a 1D
skeleton, at the grid’s resolution, the curve-skeleton must be one voxel closer to one of the sides than to the other.

### 5.5.2 Topology preservation

Topology preservation is ensured by the thinning methods because only voxels that do not change the object topology (the simple points) are removed.

Since distance field methods do not produce a curve-skeleton directly, topology preservation depends on the subsequent pruning and connectivity steps. Clearly, connectivity algorithms based on minimum spanning trees do not preserve topology because they are not able to create loops.

The power shape [5] represents a topologically correct approximation of the medial axis. However, care must be taken in the subsequent simplification steps so the final curve-skeleton maintains the topology of the power shape. The Reeb graph is an accurate representation of the object topology. However, in the discrete case, an important factor is the sampling frequency used to construct the level sets. If the sampling is too sparse, small topologically relevant features may be lost, while a too dense sampling may take too long to compute. A solution for adaptive sampling is proposed in [10]. The edge-contraction methods described in [134] specifically maintains the topology of the intermediate shapes. The curve-skeleton extraction method based on mesh decomposition described in [107] cannot preserve topology, because the extracted curve-skeleton is always a tree. This approach was taken because the curve-skeleton was used for animation where loops are generally not desirable.

Due to numerical errors and resolution of the voxel grid, potential field algorithms can create loops in the curve-skeleton that do not correspond to a tunnel or hole in the original object [48].

### 5.5.3 Connectivity

Connectivity is usually checked by all the algorithms to ensure at least a minimal degree of topology preservation with the original object. Some algorithms (e.g., thinning) explicitly
maintain connectivity during computation, while other methods check and enforce connectivity in a post-processing step (e.g., distance field). Note that re-connection as a post-processing step is always possible, but not necessarily correct.

Initial curve-skeletons produced by algorithms employing force-following on generalized fields could be disconnected even for objects with a single connected component. Theoretically, these algorithms should produce connected skeletons. However, due to numerical errors and/or insufficient resolution affecting the integration steps [48], in practice the resulting curve-skeletons may be disconnected. Using an active contour approach [43] can preserve connectivity of curve-skeleton segments, but the initial connectivity of the sinks based on minimum distance seems arbitrary and may not correspond to the topology of the original object.

5.5.4 Invariance under isometric transformations

Directional thinning methods are sensitive to the object orientation and thus are not invariant under transformation. The final result (end points, number of branches and their location) depends on the order in which the different directions are processed. Parallel and non-directional thinning methods do not have this disadvantage.

Reeb graph based methods can be sensitive to object orientation depending on the function chosen to extract the level sets. For example, the height function [10][247] is dependent on orientation, while the distance to the barycenter of the mesh [21][22] is not. Other geometric methods are usually invariant under transformation because they operate on continuous representations. Distance field, and general field function usually do not depend on object orientation.

In all cases involving discrete representations of objects, the finite resolution of the voxel grid produces small errors when objects are transformed. As a result, even though curve-skeletonization algorithms themselves are not sensitive to object orientation, the input data itself is already adversely affected by the transformation. Such small discretization errors show up on
the boundary of the transformed object and their effect on the resulting curve-skeleton is similar to that of surface noise.

5.5.5 Robustness to noise

Thinning, distance field and Voronoi-based geometric methods are sensitive to noise, generating unnecessary branches in the skeleton as a result. Several methods have been proposed to filter the resulting skeletons [8][53].

Level-set based geometric methods are affected by noise in different ways. While the location of the level-set centroids should not be affected by noise because of the averaging effect involved in computing the centroid, noise can adversely affect the number of contours in a level set, thus generating undesirable branches in the resulting skeleton.

General field approaches are less susceptible to noise because of the large amount of averaging included in the underlying computation. These methods are more sensitive to resolution because thin regions in the objects can cause numerical instabilities in the computations.

Many of the algorithms described in the literature are usually illustrated with only a few examples and are not tested on a large database of general 3D objects (for example the Princeton Shape Benchmark [213], a database of 1814 3D models). Thus it is unclear how robust and general these algorithms are with respect to the choice of their parameters.

5.5.6 Thinness

Thinning algorithms can either directly produce a curve-skeleton (by using curve-thinning templates) or further thin a surface skeleton to a 1D representation. Parallel thinning algorithms, which remove all simple points at once, may not be able to achieve 1D skeletons due to topological constraints. An illustrative 2D example is the case of a rectangle whose width is an even number of voxels: in the last step of the thinning process, the middle section of the skeleton
will be a curve two voxels thick. Although all points of this curve are simple points, removing them would completely remove the middle section. At this stage, no other simple points can be removed and the skeleton is not 1D. Directional thinning methods do not have this disadvantage: one row of voxels in the middle section will be removed when processing the up-down direction (for example), and the second row will be preserved in subsequent steps.

Distance field methods and Voronoi-based geometric methods do not produce a 1D skeleton directly. Both require significant post-processing to reduce the candidate voxels (distance field) or the medial surface (Voronoi-based methods) to a curve-skeleton.

The Reeb graph based geometric methods and the mesh decomposition-based method of [107] directly produce 1D straight-line skeleton segments by connecting level-set or component junction centroids, although the representation is no longer voxel-based.

Thinness is an implicit property of the general-field methods that use force-following or active contours to generate 1D skeleton branches. However, both general-field methods and thinning algorithms are resolution dependent.

5.5.7 Reconstructability

The medial surface of a 3D object captures local symmetries present in the object through different types of elements: surface patches in the medial surface represent symmetric plate-like regions of the original shape, while individual curves in the medial correspond to cylinder-like (tubular) shape regions.

It should be obvious that regardless of the method used to compute it, a complete and accurate reconstruction of the original object is not possible from the information retained in a curve-skeleton alone when using the simple ball-growing approach. Since the curve-skeleton contains only curve-segments, flat object parts cannot be reconstructed from it. Cylindrical shapes, (i.e., shapes that can be accurately represented by generalized cylinders), represent a special class of objects that can be accurately reconstructed from the curve-skeleton alone.
General shapes however, can only be approximated by a generalized cylinder reconstruction. Clearly, a denser curve-skeleton will generate a better reconstruction [74]. Furthermore, in the discrete domain, even perfectly cylindrical shapes may not be fully reconstructed from a thin curve-skeleton if their width is an even number of voxels (see Section 3.3.1).

Reconstruction using the ball-growing approach [74] needs distance field information in order to determine the radius of the ball that will be grown from each curve-skeleton point. In this respect, the distance field based methods have an advantage over the other methods because this information is already available. However, if other primitives are used for reconstruction (such as ellipsoids), this advantage disappears.

### 5.5.8 Visibility

Visibility is an application specific property important for virtual navigation (each boundary point must be visible from at least one curve-skeleton location). For this reason, this property is only guaranteed by those algorithms developed specifically for that particular purpose (see for example [94][108]) and is enforced during a post-processing step. So, regardless of the algorithm used to compute the curve-skeleton, visibility can be checked and, if necessary, more branches can be added to the curve-skeleton to achieve complete surface coverage.

### 5.5.9 Smoothness

Due to their discrete nature, thinning algorithms do not produce smooth curve-skeletons. Boundary irregularities propagate all the way to the curve-skeleton during the thinning process.

Distance field methods have the same disadvantage because there is no averaging involved in the computation of the distance field values. However, the subsequent pruning and reconnection steps can incorporate some smoothing constraints. Voronoi-based geometric methods behave in a similar way.
Level-set geometric methods introduce smoothing in the computation of level-set centroids, but topology changes in the evolution of level sets are not handled in a way that preserves smoothness. The same is true for methods based on mesh decomposition.

In the case of general field methods, extensive averaging is employed during the computation of the vector field and the effect is improved smoothness of the extracted curve-skeleton.

Although some algorithms may produce smoother curve-skeletons than others, smoothing can be performed in a post-processing step, regardless of the extraction algorithm used to compute the initial curve-skeleton.

5.5.10 Efficiency

Thinning is practically a linear process in the number of object voxels. Most of the voxels of the input object are removed when they are first processed (they are simple points). The non-simple points are processed again at every subsequent thinning step until they are finally removed or the algorithm terminates. An exact complexity analysis of such algorithms is difficult since they are data dependent.

The Euclidean distance field of a 3D object can be computed in linear time using the algorithm of Saito and Toriwaki [199]. The subsequent steps of filtering and reconnecting the curve-skeleton may, however, have a higher complexity (but they usually operate on a greatly reduced set of voxels).

Computation of the Voronoi diagram of a set of \( n \) points in 3D is \( O(n^2) \) in the worst case, although in practice it is almost linear [5]. As in the distance field case, additional processing is necessary to get a curve-skeleton, but the number of input elements is reduced. The computational complexity of the level-set based geometric methods depends on the kind of function computed over the object. The height function can be computed in linear time, while the exact integral of the geodesic function is \( O(n^2) \) [96] (\( n \) is the number of mesh vertices).
following steps, detecting connected components in each level set and computing their centroids, are linear in $n$.

The complexity of potential field computation is $O(n^2)$ [48], where $n$ is the number of object voxels. These methods are computationally more intensive than distance field methods because they take into consideration larger boundary areas, not just the closest point.

5.5.11 Hierarchy

Hierarchy (the ability to create a family of curve-skeletons of increased complexity) is not achievable using thinning algorithms because when processing a voxel there are only two choices: keep it, or remove it. A curve-skeleton is obtained only after the last iteration of the algorithm.

Distance field methods can produce a hierarchy of curve-skeletons by varying the number of candidate voxels selected in the pruning step, although, since many of the algorithms use some sort of averaging (e.g., clustering) and thresholding, a strict hierarchy is hard to maintain. In order to obtain a strict hierarchy (i.e., the curve-skeleton at one level is included in the next level curve-skeleton), the reconnection step must take into account the previous level curve-skeleton. Similar to distance field, Voronoi-based geometric methods can produce hierarchic curve-skeletons by pruning the surface skeleton at different thresholds and reconnecting the candidate voxels into a hierarchy of curve-skeletons. In the case of mesh decomposition based methods, if the decomposition process is a hierarchical one [107], the produced curve-skeletons will also form a hierarchy. Reeb graph based geometric methods are not hierarchic.

General-field methods produce hierarchic curve-skeletons by varying the number of seed points used to construct individual curve-skeleton segments. Additional segments added to a curve-skeleton do not affect the existing segments, creating a strict hierarchy [48]. Hierarchy is described as a property in [48][50][51][107].
5.5.12  Junction detection and component-wise differentiation

The ability to distinguish the different components of the curve-skeleton depends on the ability to
detect the junction points, i.e., the points where two or more curves meet. From this
decomposition, one can infer the corresponding part structure of the original object.

Some thinning algorithms directly classify the skeleton points as junctions, either during
thinning [226] or as a post-processing step [37]. Distance field methods must test for joints after
significant pruning and clustering [239]. However, junction placement for these classes of
methods is sensitive to noise.

From the geometric algorithm class, level-set methods directly identify the joints as the
centroids of level-sets. Joint locations depend on the function used to define the level sets.
Similarly, the joints of the mesh decomposition based curve-skeleton, identified as the mesh
component junctions, depend on the coarseness of the decomposition. The edge-contraction
method can easily identify junctions by inspecting the vertices of the skeleton, as a post-
processing step.

General-field methods can identify joints directly before extracting the curve-skeleton by
locating the critical points of the underlying vector field [48], during extraction as the local
extrema where the force-following algorithm stops [43], or the points where a previously visited
location is encountered. Joint placement depends on the function used to define the underlying
field.

5.5.13  Ability to handle different object representations

Most algorithms described here operate on voxelized or polygonal representations of 3D objects.
Thinning, and distance field methods operate only on voxelized representations. Another kind of
representation is given as an unorganized point set. Objects represented by a set of points on the
boundary can be converted to a voxelized point representation by mapping each sample point to
the closest voxel. Note that this transformation is different from voxelizing a polygon mesh. In
the later case, the interior of the mesh is completely filled with object voxels, while in the former case, the interior and the full boundary of the object are still unknown. Thinning algorithms cannot operate on such representation since they need to know the complete interior and boundary of the object.

Distance fields can be computed using the known boundary points as sources, but the field will extend outside the object since the distinction inside/outside is not known. The resulting curve-skeleton will also have branches outside, as well as inside the object, and it may be difficult to distinguish between them.

Voronoi-based geometric methods directly work with objects represented by a set of samples on the boundary [5]. Level-set based geometric methods can also handle such representations as demonstrated in [235]. Mesh decomposition based geometric methods however, cannot handle this case.

Point samples on the objects boundary can be used as sources for the general field methods. As for the distance field, the field will also extend outside the object. However, the curve-skeleton segments outside the objects can be easily identified and removed if using a force-following algorithm since they will be touching the bounding box of the volume (see [48]).

5.5.14 Discussion summary

In Table 2 we list, for each algorithm class, the properties it can achieve, as discussed in this section. A “Y” signifies that the algorithm class guarantees that particular property. An “N” is used if the algorithm class cannot achieve a given property. If the property can be easily achieved during a post-processing step, we mark it with “P”. Finally, an “S” is used if a property can be achieved by some but not all the algorithms in a class, or if a property is difficult but not impossible to achieve using this class of algorithms (as per the detailed discussion above).
Table 2. Summary of properties achievable by the various algorithm classes. Y = guaranteed property, N = property not achievable, P = property achievable as post-processing, S = property can be achieved by some but not all the algorithms in a class, or the property is difficult but not impossible to achieve using this class of algorithms.

<table>
<thead>
<tr>
<th>Property</th>
<th>Thinning</th>
<th>Distance Field</th>
<th>Geometric</th>
<th>General-Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centeredness</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Topology preservation</td>
<td>Y</td>
<td>S</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Y</td>
<td>P</td>
<td>P</td>
<td>N</td>
</tr>
<tr>
<td>Transformation Invariance</td>
<td>S</td>
<td>Y</td>
<td>S</td>
<td>Y</td>
</tr>
<tr>
<td>Robustness</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Thinness</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>Y</td>
</tr>
<tr>
<td>Reconstructability</td>
<td>N</td>
<td>S</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Visibility</td>
<td>P</td>
<td>S</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>Smoothness</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>Y</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Hierarchy</td>
<td>N</td>
<td>S</td>
<td>S</td>
<td>Y</td>
</tr>
<tr>
<td>Junction Detection</td>
<td>S</td>
<td>P</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Handle Point Sets</td>
<td>N</td>
<td>S</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

5.6 Concluding remarks

In this chapter, we reviewed previous work on curve-skeleton extraction algorithms, and we described a taxonomy of the different algorithms. As opposed to previously proposed classifications, we classified the algorithms based on the “core” step of the underlying implementation into four classes: thinning, distance field-based, geometric, and general field-based methods. In addition to a comprehensive survey of previously proposed algorithms, we also provided a detailed discussion of the advantages and drawbacks of each methodology with respect to the curve-skeleton properties identified in Chapter 3.

Note that the discussion above is based on the methods and results presented in the various publications describing the various algorithms. In order to further evaluate the advantages and disadvantages of the various methodologies, the next chapter provides a comparison of actual implementations on the same set of objects.
Chapter 6
Implementation

In the previous chapter, we described four classes of curve-skeletonization algorithms. Many of the algorithms presented in the literature are difficult to implement based on the given descriptions because typically, not all the details are given. Many have hidden thresholds, or other kinds of parameters that are not discussed in detail. Sometimes, algorithms are demonstrated on synthetic objects, or on objects with similar characteristics (e.g., tubular objects), making it unclear how they can handle general 3D shapes. Furthermore, algorithms are usually tested on a small number of objects, usually a different set in each case, making it hard to compare the different approaches. Therefore, very few implementations are available to the general community and, given the lack of sufficient details in the accompanying publications, it is almost impossible to implement some of these algorithms effectively.

In this chapter, we provide a comparison between four actual implementations, one from each class of algorithms described in the previous chapter. We apply each implementation to a common set of general 3D shapes, including one “real” object (a colon dataset) and two objects with noise (the knight and ET), allowing a direct comparison of the resulting curve-skeletons. We provide an extensive qualitative discussion of the results, as well as numerical evaluations of the resulting curve-skeletons with respect to the properties described in Chapter 3, using our curve-skeleton evaluation framework, detailed in Chapter 4. This comparison reflects the general characteristics of each class of algorithms and provides the user with a starting point in selecting one methodology or another based on the set of requirements for a specific application.
The algorithm classes presented in the previous chapter were divided into a “core” part and a “post-core” step, which is necessary to prune, cluster, connect or smooth the curve-skeleton. As many of these algorithms are difficult to implement because of the lack of sufficient details, we have only implemented the “core” of each methodology (i.e., the first step(s)), as described in the following. For the distance field and the Voronoi diagram based geometric methods, a curve-skeleton is more difficult to obtain directly. The first step of these methods generates a structure closer to a medial surface, so additional processing is required to obtain a 1D curve-skeleton. To illustrate, we used the parameter controlled filtering of the distance function described by Gagvani and Silver in [74] for distance field based methods and Amenta’s implementation of the power shape (Voronoi diagram based approximation of the skeleton) [5]. Note that for the power shape algorithm, only the surface voxels were given as input to the program and the results shown in the figure are the inside poles determined by the algorithm [5]. A comparison of the resulting structures for several objects is shown in Figure 36.

The purpose of this comparison is to give the reader a sense of how much additional processing would be required in order to extract a curve-skeleton from these structures. Note that the additional processing has to rely on some other information, not given by the underlying method used in the first step. For example, pruning the medial surface patch of the thin block below cannot take advantage of the distance field values since these values are the same for all voxels on the medial patch.

As in [50], we could compare the different algorithms using the implementations described above for the distance field and the geometric classes. However, we felt that the comparison was not fair because these implementations did not extract a 1D curve-skeleton directly, while the thinning and potential field methods did. In order to make the comparison fairer, we attempt to modify the distance field and geometric implementations to extract a 1D curve-skeleton.

In the distance field case, we extended the implementation with two additional steps performed by most algorithms in this class (see Section 5.2): pruning the candidate voxels and
**Figure 36** The candidate voxels selected from a distance field using [74] and the Voronoi diagram based power shape [5] give a sense of how much additional post-processing is required to get a 1D curve-skeleton.

connecting them. Specifically, we used clustering to reduce the number of candidate voxels and a minimum spanning tree algorithm to connect the centroids of the clusters.

To represent the geometric methods, we changed our implementation to a Reeb graph based one, which does produce a curve-skeleton in the first step of processing. We have chosen the height function because of its straight-forward implementation. We detect connected components in each level-set of the height function, select the centroid of each connected component as its representative, and add it to the curve-skeleton. When the topology of the level sets changes, we connect the centroids of neighboring components in two adjacent level sets with a straight line.

From the thinning class, we implemented the 12-subiteration curve thinning algorithm described by Palágyi and Kuba in [176]. And, we implemented the potential field method described in [48] to illustrate the general field methods. Since this is a hierarchical method, which produces curve-skeletons of various complexities, for this comparison we only used the first level
of the hierarchy (level 0), i.e., the curve-skeleton generated in the first step by connecting the
critical points of the vector field. Both the thinning method and the potential field method directly
produce a 1D curve-skeleton.

The results of our implementations on a common set of test objects are shown in Figure 37.
Note that for these results, no attempt was made to tweak the parameters of the programs in order
to obtain a better curve-skeleton for one object or another. The distance-field based curve-
skeleton was obtained using a thinness threshold of 0.6 for all objects (see [74]). This threshold
was selected for one object (the horse) to remove most of the extra branches but still maintain
most of the characteristics of the horse shape. Then this value was used for all the other objects.
For the potential field implementation, we used a field strength parameter value of 6 since in our
experience this value produced good curve-skeletons, and no additional seed points besides the
critical points were used. For the Reeb graph based geometric implementation, the height
function sampled the objects in the “up” direction (increasing Y values) at every discrete value of
Y. There were no parameters for the thinning implementation.

Most of the test objects, except the colon and the knight chess piece, were voxelized from
models downloaded from the Princeton Shape Benchmark Database [213]. The grid resolution
was chosen such that its largest dimension is 300 voxels and the aspect ratio of the object is
maintained. The colon dataset has a resolution of 204x132x260 and the knight chess piece is
40x39x87. The source code for our implementations and all models used in the experimental
section can be downloaded from [242].

From these results, it is clear that the potential field method yields the cleanest and
smoothesest curve-skeleton at the initial stage. This is due to the “global averaging” effect of the
potential field. For objects with thin, flat regions, like the ears of the bunny, the algorithm
identifies too many critical points, creating a large number of curve-skeleton branches in that
area, very similar to a medial surface. This is because the algorithm is resolution dependent, a
disadvantage identified in [48]. Also note that we are using only the first level (level 0) of the
Figure 37  Skeletons of various objects obtained using different curve-skeletonization algorithms. The comparisons are on the first steps of all of the algorithms. Note: our implementations are not fully featured and the resultant curve-skeletons may be improved. See text for details about each implementation. All of the objects and algorithm implementations are also available for download at [242].

curve-skeleton hierarchy presented in [48]. While this simple curve-skeleton is good enough for some objects, for others certain parts are not represented in the curve-skeleton at all: the tail of the
The thining algorithm performs well in all cases, but the resulting curve-skeletons are not smooth, since all irregularities of the surface propagate to the curve-skeleton through the erosion process. This also makes it more sensitive to noise. Furthermore, this noise propagation problem affects the location of the joints, which can be identified in a post-processing step because the resulting curve-skeletons are one voxel thick in all directions. As there are no parameters to set for this algorithm, one can only improve the curve-skeleton in a post-processing step, using smoothing and possibly branch pruning to remove extraneous branches.

The distance field-based algorithm produces curve-skeletons with a large number of extra branches. They are generated by the relatively large number of candidate voxels selected using the thinness heuristic. The centroids of the clusters are connected using straight lines, so the centeredness property is not always satisfied. In fact some curve-skeleton segments can go outside the object entirely as for the spider and the dinosaur. Clearly, the resulting curve-skeleton can be improved by using more sophisticated algorithms for selecting candidate voxels or for connecting them, but they come at a cost: more effort from the implementer and possibly longer running times.

The Reeb graph based geometric algorithm produces strange results for some objects because the objects were not always oriented in the natural "up-down" direction as they are presented in the figure. For example, for the dinosaur and commercial airplane shapes, the slicing direction corresponds to the front-to-back direction in the figure. This aspect can be improved upon by using different slicing directions for each object. A better alternative is to use an orientation independent function to define the level sets, such as the geodesic distance. Since the centroids of adjacent connected components are connected using straight lines, the centeredness
property is compromised, especially when the topology of the level sets changes. As in the distance field case, some segments are outside the object, as we can see for the dinosaur or the commercial airplane objects. This problem is difficult to fix regardless of the function used to define the level sets.

Although the implementations directly produce a curve-skeleton for each class of methods, the comparison is still not completely fair. We cannot claim that each of our implementations is fully representative for an entire class of algorithms, as many improvements to the resulting curve-skeletons can surely be made. However, we believe that general characteristics of each class of methods transpire from these results as discussed in this section.

As mentioned in the beginning of this chapter, the various algorithms presented here operate either on discrete voxelized datasets or on continuous polygonal representations of 3D object. Since conversion between these representations can be done using well-known algorithms (voxelization or surface extraction), the availability of one representation or the other should not be a deciding factor in choosing the curve-skeletonization algorithm.

6.1 Evaluation of implemented algorithms

In this section we evaluate each of the above implementations with respect to each of the curve-skeleton properties. We implemented the algorithms described in Chapter 4 for each property and we present the results on the set of test objects shown in Figure 37, unless otherwise specified. For those properties that can be evaluated numerically, we present the numerical values as percentages, with 100% indicating ideal performance. The details about the computation of these percentages will be presented for each property separately. For those properties which are quantified by a simple “Yes” or “No”, we define “Yes” as being 100% and “No” as 0%. For the properties of the curve-skeletonization process itself, we will discuss the implementations without
giving any quantitative measures, except in the case of efficiency, where we present a comparison of the actual running times of the different implementations on the test set.

Note that, for the results shown in Figure 37 and further used in the following tests, the parameters of the four implementations were kept constant for all objects in order to have a fair comparison of the different algorithms on the different shapes. However, this also means that the quality of the resulting curve-skeletons can be improved by appropriately “tweaking” the algorithm parameters.

### 6.1.1 Centeredness

Centeredness was measured using the algorithms described in Section 4.1. Since centeredness is defined as a normalized index, with values between zero and one, with zero representing perfect centeredness, we transform this value into a percentage by subtracting it from one. The table below shows centeredness of the resulting curve-skeletons computed with each of the four

<table>
<thead>
<tr>
<th>Object</th>
<th>Thinning</th>
<th>Distance Field</th>
<th>Geometric</th>
<th>Potential Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colon</td>
<td>72.86%</td>
<td>77.94%</td>
<td>27.51%</td>
<td>86.88%</td>
</tr>
<tr>
<td>Knight-noise</td>
<td>78.80%</td>
<td>78.79%</td>
<td>9.13%</td>
<td>94.96%</td>
</tr>
<tr>
<td>Knight</td>
<td>90.63%</td>
<td>73.06%</td>
<td>89.18%</td>
<td>95.26%</td>
</tr>
<tr>
<td>Horse</td>
<td>72.79%</td>
<td>77.22%</td>
<td>65.14%</td>
<td>94.12%</td>
</tr>
<tr>
<td>Tree</td>
<td>73.89%</td>
<td>75.53%</td>
<td>65.25%</td>
<td>93.55%</td>
</tr>
<tr>
<td>Hammer</td>
<td>72.78%</td>
<td>87.66%</td>
<td>65.06%</td>
<td>92.60%</td>
</tr>
<tr>
<td>Bunny</td>
<td>77.08%</td>
<td>78.24%</td>
<td>63.63%</td>
<td>84.73%</td>
</tr>
<tr>
<td>Biplane</td>
<td>84.70%</td>
<td>86.17%</td>
<td>57.15%</td>
<td>89.94%</td>
</tr>
<tr>
<td>Commercial Airplane</td>
<td>84.07%</td>
<td>86.76%</td>
<td>74.31%</td>
<td>91.71%</td>
</tr>
<tr>
<td>Girl with ponytail</td>
<td>72.20%</td>
<td>81.68%</td>
<td>63.41%</td>
<td>92.08%</td>
</tr>
<tr>
<td>Helicopter</td>
<td>81.21%</td>
<td>87.86%</td>
<td>58.66%</td>
<td>94.51%</td>
</tr>
<tr>
<td>ET-noise</td>
<td>65.06%</td>
<td>77.25%</td>
<td>45.58%</td>
<td>91.14%</td>
</tr>
<tr>
<td>ET</td>
<td>79.93%</td>
<td>77.60%</td>
<td>81.29%</td>
<td>93.10%</td>
</tr>
<tr>
<td>Bishop</td>
<td>73.81%</td>
<td>72.41%</td>
<td>84.77%</td>
<td>97.79%</td>
</tr>
<tr>
<td>Spider</td>
<td>67.54%</td>
<td>68.89%</td>
<td>75.63%</td>
<td>91.96%</td>
</tr>
<tr>
<td>Dinosaur</td>
<td>70.74%</td>
<td>79.84%</td>
<td>45.95%</td>
<td>94.79%</td>
</tr>
<tr>
<td>Hand</td>
<td>74.75%</td>
<td>83.31%</td>
<td>61.31%</td>
<td>94.05%</td>
</tr>
<tr>
<td>Castle</td>
<td>71.63%</td>
<td>80.25%</td>
<td>79.79%</td>
<td>85.40%</td>
</tr>
</tbody>
</table>
implementations and on each of the test objects from Figure 37. We also show the overall performance of each implementation as the average centeredness over the entire test set.

As we can see from the above table, the potential field based implementation achieves the best centeredness on this set of test objects. However, the poor performance of the distance field implementation can be explained by the fact that we are connecting candidate voxels with straight lines. This particular implementation could be improved by performing an energy minimization for each of these connections (similar to [150]), allowing them to align with the ridges of the distance field, which are more centered within the object.

### 6.1.2 Topology preservation

As topology preservation is a Yes/No property, we will interpret a Yes as 100% and a No as 0%. The table below gives the topology preservation results for the individual objects for each implementation and an average of the performance with respect to this property for each implementation.

<table>
<thead>
<tr>
<th>Object</th>
<th>Thinning</th>
<th>Distance Field</th>
<th>Geometric</th>
<th>Potential Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Performance</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100%</td>
<td>77.78%</td>
<td>88.89%</td>
<td>38.89%</td>
<td></td>
</tr>
<tr>
<td>Performance by Case</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Colon</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Knight-noise</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Knight</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Horse</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Tree</td>
<td>100.00%</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Hammer</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Bunny</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Biplane</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Commercial Airplane</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Girl with ponytail</td>
<td>100.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Helicopter</td>
<td>100.00%</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>ET-noise</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>ET</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Bishop</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Spider</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Dinosaur</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Hand</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Castle</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
</tbody>
</table>
Note that thinning is always at 100% since thinning algorithms are designed with preservation of topology in mind. Since our distance field implementation cannot produce loops in the curve-skeleton because we are using a minimum spanning tree algorithm to connect candidate voxels, topology preservation will not be achieved for objects with tunnels such as the colon, tree, girl with ponytail and the helicopter. The Reeb graph implementation also fails to identify certain loops that are not apparent for the particular height function used here. The girl with ponytail is an example where the tunnel created between the ponytail and the body is not represented by a loop in the resulting curve-skeleton because the direction of the used high-function was back-to-front. Looking at slices of the object in this direction, the ponytail does not become a separate component from the main body at any point, and thus no loop is created. In contrast, the potential field implementation detects too many loops in some flat areas, as the tail section of the commercial airplane, or the ears of the bunny. Also, due to numerical errors which hamper the critical point detection and the force-following algorithms, the resulting skeleton is sometime disconnected (see for example the biplane). Note that by varying the parameters given to the potential field, the results could improve. Also, a post-processing step could be used to reconnect disconnected regions of the potential-field based curve-skeleton.

6.1.3 Connectivity

Connectivity is also a Yes/No property. As in the case of topology preservation, we considered a Yes to be 100% and a No to be 0%. The table below shows the scores for individual objects as well as the average performance.

Again, the thinning implementation achieves connectivity every time. The radical improvement in performance of the distance field and the Reeb graph implementations, when compared to topology preservation, is explained by the fact that connectivity is built into these implementations. The distance field implementation uses a distinct connection step, while the Reeb graph implementation explicitly maintains connectivity between successive level set
Table 5. Connectivity

<table>
<thead>
<tr>
<th>Object</th>
<th>Thinning</th>
<th>Distance Field</th>
<th>Geometric</th>
<th>Potential Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Performance</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>55.56%</td>
</tr>
<tr>
<td>Performance by Case</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Colon</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Knight-noise</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Knight</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Horse</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Tree</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Hammer</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Bunny</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Biplane</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Commercial Airplane</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Girl with ponytail</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Helicopter</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>ET-noise</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>ET</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Bishop</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Spider</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Dinosaur</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Hand</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Castle</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

components as it traces their evolution. We also note an improvement of the potential field algorithm when compared to topology preservation. This can be attributed to the fact that the loops in the curve-skeleton are now ignored. While its connectivity score is relatively low compared to the other methods, it could be further improved by adding a re-connection step as post-processing.

### 6.1.4 Invariance under isometric transformations

To quantify invariance under transformation, we have to compute the curve-skeleton for a set of transformed and un-transformed objects, and compare the resulting curve-skeletons. We use the procedure described in Section 4.4: for each one of the objects in Figure 37, we compute the curve-skeleton of the object in the original orientation. Then, we transform the object using a random transformation and we re-compute the curve-skeleton of the transformed object. Finally, we compute the distance between the transformed curve-skeleton of the original object and the curve-skeleton of the transformed object, using the “integral” distance ($IDist$) and the set $W_2$ of
parameterizations defined in Section 4.4.1. Since none of the implementations should be sensitive
to the actual position of the object in space, but it may be sensitive to the object’s orientation, we
only used rotations to build a random transformation matrix. The random transformation was
constructed by concatenating three rotations with random angles about the three main axes.

The distance between two skeletons, as defined in Section 4.4 is an un-bounded quantity
(i.e., it is not normalized). A value of zero indicates that the two skeletons are identical and as the
value increases, the two skeletons are more and more different. In order to maintain consistency
of presentation, we will normalize these values with respect to the largest value recorded for our
set of test objects as follows: let \( \{v_1, v_2, v_3, \ldots, v_n\} \) be the set of distances computed using the
procedure outlines above and let \( v_k \) be the maximum value among them. The normalized version
\( v_{i}^{(n)} \) of \( v_i \) is computed as \( v_{i}^{(n)} = v_i / v_k \) and then transformed into a percentage value by subtracting
it from one. The resulting percentage value is zero if there is no different between the transformed
skeleton and the skeleton of the transformed object (i.e., the implementation is invariant under
transformation) and non-zero otherwise.

The results of the evaluation with respect to the invariance under transformation criterion are
shown in the table below, for each object and as average performance of each implementation
over the entire test set.

As expected, the Reeb graph implementation behaves the worst, because the height function
used to define the level sets is actually dependent on the orientation of the object. We also expect
thinning to be affected by object orientation since this is a directional thinning algorithm. However, the effect of object orientation is not as large as for the Reeb graph algorithm. Note
also that the distance field and the potential field implementations are also somehow affected by
changes in object orientation. This can be explained by the fact that object transformations in
discrete space are subject to round-off errors, and so the resulting skeletons may be slightly offset
from the “true” position.
Table 6. Invariance under isometric transformations

<table>
<thead>
<tr>
<th>Object</th>
<th>Thinning</th>
<th>Distance Field</th>
<th>Geometric</th>
<th>Potential Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colon</td>
<td>69.40%</td>
<td>86.90%</td>
<td>0.00%</td>
<td>97.24%</td>
</tr>
<tr>
<td>Knight</td>
<td>97.30%</td>
<td>97.52%</td>
<td>97.17%</td>
<td>99.88%</td>
</tr>
<tr>
<td>Horse</td>
<td>88.33%</td>
<td>89.55%</td>
<td>78.81%</td>
<td>98.16%</td>
</tr>
<tr>
<td>Tree</td>
<td>77.40%</td>
<td>93.00%</td>
<td>69.34%</td>
<td>85.46%</td>
</tr>
<tr>
<td>Hammer</td>
<td>91.68%</td>
<td>96.04%</td>
<td>93.44%</td>
<td>98.11%</td>
</tr>
<tr>
<td>Bunny</td>
<td>84.75%</td>
<td>92.41%</td>
<td>75.69%</td>
<td>98.39%</td>
</tr>
<tr>
<td>Biplane</td>
<td>84.35%</td>
<td>80.60%</td>
<td>74.44%</td>
<td>87.83%</td>
</tr>
<tr>
<td>Commercial Airplane</td>
<td>96.82%</td>
<td>95.08%</td>
<td>81.51%</td>
<td>93.98%</td>
</tr>
<tr>
<td>Girl with ponytail</td>
<td>94.07%</td>
<td>94.99%</td>
<td>78.97%</td>
<td>98.95%</td>
</tr>
<tr>
<td>Helicopter</td>
<td>96.08%</td>
<td>96.33%</td>
<td>85.66%</td>
<td>94.00%</td>
</tr>
<tr>
<td>ET</td>
<td>98.02%</td>
<td>97.21%</td>
<td>95.03%</td>
<td>99.06%</td>
</tr>
<tr>
<td>Bishop</td>
<td>90.79%</td>
<td>92.73%</td>
<td>85.54%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Spider</td>
<td>85.82%</td>
<td>93.82%</td>
<td>39.25%</td>
<td>98.90%</td>
</tr>
<tr>
<td>Dinosaur</td>
<td>92.91%</td>
<td>94.28%</td>
<td>75.83%</td>
<td>99.62%</td>
</tr>
<tr>
<td>Hand</td>
<td>88.56%</td>
<td>91.01%</td>
<td>62.08%</td>
<td>98.55%</td>
</tr>
<tr>
<td>Castle</td>
<td>69.37%</td>
<td>65.34%</td>
<td>62.84%</td>
<td>22.36%</td>
</tr>
</tbody>
</table>

Overall Performance:
- 87.85% for Thinning
- 91.05% for Distance Field
- 72.22% for Geometric
- 91.91% for Potential Field

Performance by Case

6.1.5 Robustness to noise

Robustness to noise is evaluated computed similarly to invariance under transformation. For each test object shown in Figure 37, we artificially added uniform noise and re-computed the curve-skeleton using each of our four implementations. The amount of noise added to the objects, expressed as a percentage (20% in this case) represents the percentage of original surface voxels from the object that will be altered (either deleted or added). Next, we computed the distance between each pair of original curve-skeleton and curve-skeleton of the object with noise. Finally, for presentation, we normalized the distances between the corresponding skeletons using the same method as for transformation invariance: the range between the minimum and the maximum values in this test set was mapped to percentages from 100% to 0%.

The table below shows the robustness to noise score for each implementation and for each test object, and also the mean performance of each algorithm.
Table 7. Robustness to noise

<table>
<thead>
<tr>
<th>Object</th>
<th>Thinning</th>
<th>Distance Field</th>
<th>Geometric</th>
<th>Potential Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Performance</td>
<td>88.22%</td>
<td>90.30%</td>
<td>50.84%</td>
<td>92.54%</td>
</tr>
<tr>
<td>Performance by Case</td>
<td>81.19%</td>
<td>82.58%</td>
<td>23.33%</td>
<td>96.18%</td>
</tr>
<tr>
<td>Colon</td>
<td>95.90%</td>
<td>95.81%</td>
<td>87.18%</td>
<td>99.86%</td>
</tr>
<tr>
<td>Horse</td>
<td>95.11%</td>
<td>95.81%</td>
<td>67.31%</td>
<td>93.83%</td>
</tr>
<tr>
<td>Tree</td>
<td>90.80%</td>
<td>90.83%</td>
<td>77.22%</td>
<td>80.09%</td>
</tr>
<tr>
<td>Hammer</td>
<td>86.92%</td>
<td>90.42%</td>
<td>36.55%</td>
<td>97.06%</td>
</tr>
<tr>
<td>Biplane</td>
<td>87.34%</td>
<td>71.84%</td>
<td>58.25%</td>
<td>49.55%</td>
</tr>
<tr>
<td>Commercial Airplane</td>
<td>94.56%</td>
<td>94.41%</td>
<td>45.97%</td>
<td>91.56%</td>
</tr>
<tr>
<td>Girl with ponytail</td>
<td>91.78%</td>
<td>96.83%</td>
<td>67.74%</td>
<td>91.56%</td>
</tr>
<tr>
<td>Helicopter</td>
<td>90.30%</td>
<td>90.01%</td>
<td>45.97%</td>
<td>94.08%</td>
</tr>
<tr>
<td>ET</td>
<td>88.65%</td>
<td>90.25%</td>
<td>0.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Spider</td>
<td>76.91%</td>
<td>90.35%</td>
<td>64.04%</td>
<td>98.54%</td>
</tr>
<tr>
<td>Dinosaur</td>
<td>87.51%</td>
<td>91.34%</td>
<td>32.28%</td>
<td>98.87%</td>
</tr>
<tr>
<td>Hand</td>
<td>89.89%</td>
<td>86.08%</td>
<td>14.99%</td>
<td>97.03%</td>
</tr>
</tbody>
</table>

As in the case of transformation invariance, the potential field implementation is the least sensitive to noise, closely followed by the distance field, where clustering of the candidate voxels removes some of the effects of the noise. For the Reeb graph implementation, although a certain degree of averaging is present in the computation of the centers of the connected the level sets, the number of connected components in each level set is affected by the presence of noise, and numerous extra skeleton segments are generated, which explains the relatively score obtained by this implementation.

6.1.6 Thinness

Since our distance field, Reeb graph and potential field implementations directly compute the curve-skeletons as curve (or straight line) segments, they are considered thin (one dimensional) by definition. The thinness question arises only when evaluating curve-skeletons defined as voxels on a discrete grid (see Section 4.6). The thinning algorithm implemented here is the only one determining the curve-skeleton as a set of voxels and we will only compute the thinness index for these curve-skeletons. For the other implementations we will assign a score of 100% for
the thinness property. The thinness index defined in Section 4.6 (a normalized value) is transformed into a percentage by subtracting it from one.

The table below shows the thinness measures, as defined under Section 4.6, for the curve-skeletons obtained using our four curve-skeletonization implementations for every object in Figure 37. The scores for the distance field, geometric and potential field are always 100% since they represent the curve-skeletons are curve segments.

<table>
<thead>
<tr>
<th>Table 8. Thinnness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
</tr>
<tr>
<td>Overall Performance</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance by Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colon</td>
</tr>
<tr>
<td>Knight-noise</td>
</tr>
<tr>
<td>Knight</td>
</tr>
<tr>
<td>Horse</td>
</tr>
<tr>
<td>Tree</td>
</tr>
<tr>
<td>Hammer</td>
</tr>
<tr>
<td>Bunny</td>
</tr>
<tr>
<td>Biplane</td>
</tr>
<tr>
<td>Commercial Airplane</td>
</tr>
<tr>
<td>Girl with ponytail</td>
</tr>
<tr>
<td>Helicopter</td>
</tr>
<tr>
<td>ET-noise</td>
</tr>
<tr>
<td>ET</td>
</tr>
<tr>
<td>Bishop</td>
</tr>
<tr>
<td>Spider</td>
</tr>
<tr>
<td>Dinosaur</td>
</tr>
<tr>
<td>Hand</td>
</tr>
<tr>
<td>Castle</td>
</tr>
</tbody>
</table>

Note that even though the thinning algorithm was implemented using the curve-thinning templates, the resulting curve-skeleton is not 100% thin because in order to preserve topology of the original object, extra voxels need to be preserved around junctions.

6.1.7 Reconstructability

To evaluate the reconstruction ability of each of the curve-skeletons, we re-constructed an object from a curve-skeleton by growing a sphere at each curve-skeleton point, with a radius equal to the
distance transform value at that point. Next, we compared the original object and the reconstructed object and we computed the reconstructed index defined in Section 4.7. The reconstruction index is normalized, with a value of zero indicating that the original object can be fully reconstructed, and a value of one indicating that nothing from the original object can be recovered. For presentation consistency, we transform this value by subtracting it from one and showing it as a percentage, with 100% indicating perfect reconstruction. The table below lists the reconstruction score for each test object shown in Figure 37, as well as the average performance of each of the implementations on our test set.

<table>
<thead>
<tr>
<th>Object</th>
<th>Thinning</th>
<th>Distance Field</th>
<th>Geometric</th>
<th>Potential Field</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Overall</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Performance</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Colon</td>
<td>33.45%</td>
<td>79.69%</td>
<td>23.48%</td>
<td>56.68%</td>
</tr>
<tr>
<td>Knight-noise</td>
<td>54.92%</td>
<td>71.26%</td>
<td>53.93%</td>
<td>49.81%</td>
</tr>
<tr>
<td>Knight</td>
<td>52.94%</td>
<td>69.29%</td>
<td>62.20%</td>
<td>57.85%</td>
</tr>
<tr>
<td>Horse</td>
<td>54.84%</td>
<td>78.96%</td>
<td>47.59%</td>
<td>66.69%</td>
</tr>
<tr>
<td>Tree</td>
<td>58.88%</td>
<td>35.98%</td>
<td>53.21%</td>
<td>65.81%</td>
</tr>
<tr>
<td>Hammer</td>
<td>54.35%</td>
<td>73.04%</td>
<td>48.63%</td>
<td>59.01%</td>
</tr>
<tr>
<td>Bunny</td>
<td>49.00%</td>
<td>82.51%</td>
<td>37.08%</td>
<td>69.23%</td>
</tr>
<tr>
<td>Biplane</td>
<td>32.71%</td>
<td>68.32%</td>
<td>21.51%</td>
<td>28.46%</td>
</tr>
<tr>
<td>Commercial Airplane</td>
<td>65.25%</td>
<td>77.70%</td>
<td>23.40%</td>
<td>66.23%</td>
</tr>
<tr>
<td>Girl with ponytail</td>
<td>57.90%</td>
<td>65.27%</td>
<td>44.68%</td>
<td>72.42%</td>
</tr>
<tr>
<td>Helicopter</td>
<td>67.68%</td>
<td>77.31%</td>
<td>43.93%</td>
<td>72.96%</td>
</tr>
<tr>
<td>ET-noise</td>
<td>47.71%</td>
<td>52.07%</td>
<td>52.65%</td>
<td>58.43%</td>
</tr>
<tr>
<td>ET</td>
<td>51.67%</td>
<td>50.57%</td>
<td>60.08%</td>
<td>66.76%</td>
</tr>
<tr>
<td>Bishop</td>
<td>65.42%</td>
<td>78.17%</td>
<td>79.53%</td>
<td>58.54%</td>
</tr>
<tr>
<td>Spider</td>
<td>60.95%</td>
<td>71.58%</td>
<td>51.23%</td>
<td>64.49%</td>
</tr>
<tr>
<td>Dinosaur</td>
<td>53.33%</td>
<td>81.34%</td>
<td>26.37%</td>
<td>71.08%</td>
</tr>
<tr>
<td>Hand</td>
<td>43.98%</td>
<td>87.19%</td>
<td>18.25%</td>
<td>54.37%</td>
</tr>
<tr>
<td>Castle</td>
<td>37.44%</td>
<td>78.28%</td>
<td>47.22%</td>
<td>59.48%</td>
</tr>
</tbody>
</table>

Since, in the general case, an object cannot be fully reconstructed from a curve-skeleton, we do not expect to see particularly large scores in the table above for any of the algorithms. Also, none of our implementations includes a step to maximize reconstruction. Such a step could be easily added as post-processing to the potential field or distance field implementations, by adding
more segments to the curve-skeleton, using new candidate points. It is unclear how to maximize reconstruction for the Reeb graph and thinning implementations.

6.1.8 Visibility

Visibility is achieved by a curve-skeleton if the entire surface of the object can be seen from the curve-skeleton. Using the algorithm from Section 4.8, we compute the visibility index for each of the curve-skeletons computed using our four implementations for all objects shown in Figure 37. Since the visibility index defined in Section 4.8 is a normalized value, we subtract it from one and transform it into a percentage. The resulting value represents the percentage of the object’s surface that is visible from the curve-skeleton. The table below shows the individual visibility percentages for each of the objects in Figure 37 as well as the overall performance of each of the four implementations.

<table>
<thead>
<tr>
<th>Object</th>
<th>Thinning</th>
<th>Distance Field</th>
<th>Geometric</th>
<th>Potential Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Performance</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Colon</td>
<td>99.94%</td>
<td>99.99%</td>
<td>99.85%</td>
<td>99.90%</td>
</tr>
<tr>
<td>Knight-noise</td>
<td>99.88%</td>
<td>100.00%</td>
<td>99.94%</td>
<td>99.56%</td>
</tr>
<tr>
<td>Knight</td>
<td>99.98%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>99.95%</td>
</tr>
<tr>
<td>Horse</td>
<td>99.94%</td>
<td>99.95%</td>
<td>99.95%</td>
<td>99.69%</td>
</tr>
<tr>
<td>Tree</td>
<td>99.99%</td>
<td>90.89%</td>
<td>99.99%</td>
<td>99.88%</td>
</tr>
<tr>
<td>Hammer</td>
<td>100.00%</td>
<td>99.92%</td>
<td>99.91%</td>
<td>99.98%</td>
</tr>
<tr>
<td>Bunny</td>
<td>99.92%</td>
<td>100.00%</td>
<td>99.96%</td>
<td>99.92%</td>
</tr>
<tr>
<td>Biplane</td>
<td>100.00%</td>
<td>100.00%</td>
<td>99.61%</td>
<td>99.61%</td>
</tr>
<tr>
<td>Commercial Airplane</td>
<td>99.62%</td>
<td>100.00%</td>
<td>99.94%</td>
<td>99.97%</td>
</tr>
<tr>
<td>Girl with ponytail</td>
<td>99.98%</td>
<td>99.13%</td>
<td>99.69%</td>
<td>99.37%</td>
</tr>
<tr>
<td>Helicopter</td>
<td>100.00%</td>
<td>99.89%</td>
<td>100.00%</td>
<td>99.96%</td>
</tr>
<tr>
<td>ET-noise</td>
<td>99.93%</td>
<td>98.28%</td>
<td>99.84%</td>
<td>99.22%</td>
</tr>
<tr>
<td>ET</td>
<td>99.96%</td>
<td>96.32%</td>
<td>99.95%</td>
<td>99.49%</td>
</tr>
<tr>
<td>Bishop</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>98.53%</td>
</tr>
<tr>
<td>Spider</td>
<td>100.00%</td>
<td>75.26%</td>
<td>99.97%</td>
<td>99.12%</td>
</tr>
<tr>
<td>Dinosaur</td>
<td>100.00%</td>
<td>99.87%</td>
<td>95.83%</td>
<td>97.47%</td>
</tr>
<tr>
<td>Hand</td>
<td>99.98%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>95.27%</td>
</tr>
<tr>
<td>Castle</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>99.35%</td>
</tr>
</tbody>
</table>
Note that even though none of the actual implementation guarantees total visibility, the results are very close to 100% for all implementations. For the colon dataset, the distance field algorithm seems to perform best, but the other implementations are very close. Of course, for a virtual colonoscopy application, perfect visibility is required. As in the case of reconstruction, such a post-processing step can be easily included in the potential field and distance field implementations by using more candidate voxels for new skeleton segments, it is however unclear how to proceed for the Reeb graph and thinning implementations.

### 6.1.9 Smoothness

To measure smoothness, we used the minimum smoothness formulation from Section 4.9, since many of our curve-skeletons (the results of distance field and Reeb graph implementations) are built using straight-line segments. We also used a value of zero for the smoothness threshold. As the result is an un-bounded value, for presentation purposes, we will normalize it across our test set and present it as a percentage, similar to invariance under transformation or robustness to noise. Let \( \{v_1, v_2, v_3, \ldots v_n\} \) be the set of smoothness values computed using the procedure described in Section 4.9, and let \( v_k \) be the maximum value among them. The normalized version \( v_i^{(n)} \) of \( v_i \) is computed as \( v_i^{(n)} = v_i / v_k \) and then transformed into a percentage value by subtracting it from one. The resulting percentage value is zero if the curve-skeleton is perfectly smooth (i.e. it is a straight line since we are using a threshold of zero), and non-zero otherwise, with 100% indicating the least smooth result in our test set. The table below shows the smoothness scores for each of the objects in Figure 37, for each of the four implemented algorithms and the average performance of each algorithm on our test set.

Note that, since we used a value of zero for the smoothness threshold, any small variation of the normal direction to the curve penalizes the final smoothness value and, as a result, the average performance over the test set is not very good for either of the implementations. According to our
qualitative evaluation of the resulting curve-skeletons, the smoothness measure is largest for the potential field implementation.

<table>
<thead>
<tr>
<th>Object</th>
<th>Thinning</th>
<th>Distance Field</th>
<th>Geometric</th>
<th>Potential Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Performance</td>
<td>14.62%</td>
<td>34.01%</td>
<td>14.44%</td>
<td>38.35%</td>
</tr>
<tr>
<td>Performance by Case</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Colon</td>
<td>10.41%</td>
<td>28.33%</td>
<td>21.60%</td>
<td>64.34%</td>
</tr>
<tr>
<td>Knight-noise</td>
<td>20.60%</td>
<td>16.40%</td>
<td>33.05%</td>
<td>68.13%</td>
</tr>
<tr>
<td>Knight</td>
<td>11.42%</td>
<td>0.00%</td>
<td>10.98%</td>
<td>83.59%</td>
</tr>
<tr>
<td>Horse</td>
<td>13.97%</td>
<td>22.81%</td>
<td>4.83%</td>
<td>42.36%</td>
</tr>
<tr>
<td>Tree</td>
<td>21.49%</td>
<td>30.34%</td>
<td>15.29%</td>
<td>1.77%</td>
</tr>
<tr>
<td>Hammer</td>
<td>17.30%</td>
<td>46.49%</td>
<td>16.84%</td>
<td>17.67%</td>
</tr>
<tr>
<td>Bunny</td>
<td>19.63%</td>
<td>32.66%</td>
<td>7.65%</td>
<td>39.69%</td>
</tr>
<tr>
<td>Biplane</td>
<td>17.58%</td>
<td>52.92%</td>
<td>4.27%</td>
<td>36.91%</td>
</tr>
<tr>
<td>Commercial Airplane</td>
<td>5.85%</td>
<td>47.08%</td>
<td>27.61%</td>
<td>43.31%</td>
</tr>
<tr>
<td>Girl with ponytail</td>
<td>14.38%</td>
<td>39.74%</td>
<td>6.93%</td>
<td>58.70%</td>
</tr>
<tr>
<td>Helicopter</td>
<td>10.77%</td>
<td>100.00%</td>
<td>27.71%</td>
<td>9.02%</td>
</tr>
<tr>
<td>ET-noise</td>
<td>21.38%</td>
<td>17.19%</td>
<td>21.82%</td>
<td>38.25%</td>
</tr>
<tr>
<td>ET</td>
<td>15.44%</td>
<td>27.00%</td>
<td>7.05%</td>
<td>34.83%</td>
</tr>
<tr>
<td>Bishop</td>
<td>14.59%</td>
<td>13.12%</td>
<td>7.80%</td>
<td>34.62%</td>
</tr>
<tr>
<td>Spider</td>
<td>18.02%</td>
<td>26.88%</td>
<td>12.07%</td>
<td>12.02%</td>
</tr>
<tr>
<td>Dinosaur</td>
<td>8.72%</td>
<td>27.14%</td>
<td>11.51%</td>
<td>41.72%</td>
</tr>
<tr>
<td>Hand</td>
<td>8.85%</td>
<td>18.34%</td>
<td>12.3</td>
<td>46.56%</td>
</tr>
<tr>
<td>Castle</td>
<td>12.80%</td>
<td>65.68%</td>
<td>10.67%</td>
<td>16.88%</td>
</tr>
</tbody>
</table>

### 6.1.10 Efficiency

We measured efficiency in terms of the running time of our implementations, on the same machine, on the set of test objects shown in Figure 37. In Figure 38 we show a comparison of the running times (in milliseconds, on a logarithmic scale) as a function of the total number of object voxels. Note that no optimizations were done for the implementations.

Obviously, the potential field implementation whose algorithmic complexity is $O(n^2)$, where $n$ is the number of object voxels, is the slowest. The initial sharp increase in running time visible in every curve in the graph is due to input/output operations which account for most of the total processing time for very small objects. As the objects become larger, the input/output operations
become insignificant compared to the actual curve-skeleton computation, revealing the true dependency of the running time on the total number of object voxels.

6.1.11 Hierarchy

Only the distance field implementation included a parameter will generate skeletons of different complexities. However, the resulting hierarchy is unlikely to be a strict hierarchy because of the connectivity step. In this particular implementation, we used a minimum spanning tree to connect the candidate voxels. As a result, as more candidate voxels survive the pruning step, it is unlikely that the previous spanning tree will be a subset of the new one. A hierarchical component can be easily added to the potential field implementation by starting the force-following procedure at some other voxels, for example surface voxels, selected by curvature. In addition, the resulting hierarchy will be a strict one as previously detected curve-skeleton segments will not be changed as more surface voxels are processed. For the thinning and Reeb graph implementations however, no hierarchy can be defined.
6.1.12 Junction detection

Except thinning, all other three implementations detect some kind of junctions during processing. The distance field algorithm identifies the candidate voxels surviving the pruning step as junctions, and connects those using straight lines. The Reeb graph implementation detects the centroids of each level set as a junction and connects them with straight lines as well. The potential field algorithm, identifies the critical points of the potential field as the skeleton junctions and connects them using a force following algorithm. Thinning can only detect junctions as a post-processing step.

6.1.13 Ability to handle different object representations

Each of our four implementations handles only objects represented by voxels on a discrete grid. As polygonal models can be easily voxelized, we can accommodate such representations as well. Unorganized point sets can also be easily mapped to a discrete grid, but now the distinction between the inside and the outside of the object is not known. This distinction however is essential for the thinning algorithm, since we iteratively remove object voxels, and for the Reeb graph algorithm which needs to compute connected components of object voxels. The distance field and the potential field can both be computed on the entire voxel grid, both inside and outside the object, using the point samples as generators. The resulting curve-skeleton will represent both the inside and the outside of the object. While the potential field skeleton can then be easily separated into an inside and several outside components (see Section 7.5.1.1), the task of separating the inside and the outside skeletons is more difficult for the distance field algorithm which uses a minimum spanning tree on the disconnected set of candidate voxel, without considering the values in the underlying distance field.

A summary of the scores for the measurable properties of the curve-skeleton obtained by the different implementations is shown in Table 12.
Table 12. Summary of overall performance of each implementation with respect to the curve-skeleton properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Thinning</th>
<th>Distance Field</th>
<th>Geometric</th>
<th>Potential Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Performance</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Centeredness</td>
<td>75.80%</td>
<td>79.47%</td>
<td>61.82%</td>
<td>92.14%</td>
</tr>
<tr>
<td>Topology preservation</td>
<td>100%</td>
<td>77.78%</td>
<td>88.89%</td>
<td>38.89%</td>
</tr>
<tr>
<td>Connectivity</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>55.56%</td>
</tr>
<tr>
<td>Invariance under transformation</td>
<td>87.85%</td>
<td>91.05%</td>
<td>72.22%</td>
<td>91.91%</td>
</tr>
<tr>
<td>Robustness to noise</td>
<td>88.22%</td>
<td>90.30%</td>
<td>50.84%</td>
<td>92.54%</td>
</tr>
<tr>
<td>Thinness</td>
<td>77.47%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Reconstruction</td>
<td>52.36%</td>
<td>71.03%</td>
<td>44.16%</td>
<td>61.02%</td>
</tr>
<tr>
<td>Visibility</td>
<td>99.95%</td>
<td>97.75%</td>
<td>99.69%</td>
<td>99.24%</td>
</tr>
<tr>
<td>Smoothness</td>
<td>14.62%</td>
<td>54.01%</td>
<td>14.44%</td>
<td>38.35%</td>
</tr>
</tbody>
</table>

6.2 Concluding remarks

Many of the skeletonization algorithms presented in the literature are demonstrated on a small set of test objects, usually a different one in each case, making it difficult to directly compare their performance. In many cases, the objects used to demonstrate the algorithm have similar characteristics, such as tubular objects, and it is unclear whether these algorithms can handle general 3D shapes. Furthermore, many of these algorithms are difficult to implement because, typically, not all the details are given: missing thresholds, hidden parameter settings.

In this chapter, we extended the discussion of the various curve-skeletonization methodologies presented in Chapter 5, by comparing actual implementations of each algorithm class on the same set of test objects, representing general 3D shapes. As many of the algorithms are difficult to implement entirely because of lack of sufficient detail, we implemented a “core” part from each class. The purpose of this comparison was to give a sense of how close each of these methodologies comes to a curve-skeleton after the first step of processing. Certainly the comparison of specific implementations does not make justice to an entire class of algorithms, but
it does give a sense of how much post-processing is required to go from the result of the “core” step to a usable curve-skeleton. We have also evaluated each of the resulting curve-skeletons with respect to the curve-skeleton properties introduced in Chapter 3. For those properties that can be evaluated numerically, we provided tables indicating numerical performance on the individual objects, as well as average performance of each of our implementations on the set of test objects. We focused our attention on discussing how each of the presented methodologies can achieve each of the curve-skeleton properties.

Evaluation of new curve-skeletonization algorithms with respect to the criteria (properties) presented here (and more) is an important issue for real users of curve-skeletons. We hope this chapter in particular, and this whole thesis in general, provides a reference model for evaluating existing and future curve-skeletonization algorithms. The side-by-side comparison on the same set of test objects described here could be used to develop a curve-skeleton benchmark. In this scenario, developers of skeletonization algorithms would submit their algorithms to the system and automated tests would be run on the common set of test objects to determine the degree to which each property is achieved. Then, curve-skeleton users would have a quick way to select the best curve-skeletonization implementation based on the properties relevant to their application.
Chapter 7

Vector Field-Based Hierarchical Skeletonization

Framework

The curve-skeleton definition adopted in this work does not completely specify the curve-skeleton for a given object. An important property of a good skeletonization algorithm is the ability to generate curve-skeletons of different complexities (multi-scale), given that the curve-skeleton is not unique for a given object. To account for the multi-scale aspect, we defined the curve-skeleton set as a strict hierarchy of curve-skeletons (Section 3.6).

Figure 39  A cow model (a) and several levels of a curve-skeleton hierarchy (b), (c) and (d).

In this chapter we present our work on curve-skeleton extraction algorithms. We introduce the concept of hierarchical curve-skeleton and describe a general and robust methodology which computes a multi-scale family of increasingly detailed curve-skeletons: a curve-skeleton set. Figure 39 shows three examples of hierarchical curve-skeletons of a cow object. The algorithm is based upon computing a vector (force) field over a discretization of the 3D object and using topological characteristics of the resulting vector field, such as critical points and critical curves, to extract the curve-skeleton. The framework is general enough to accommodate any type of
vector field computed based on the original 3D object. We demonstrate this framework using two
types of vector fields on many different types of 3D objects (volumetric, polygonal and scattered
point sets). We evaluate the performance of our framework with respect to the two types of vector
fields and the curve-skeleton properties.

Without loss of generality we assume that a 3D object refers to a 3D voxelized discretization
of that object. Polygonal models can be converted into volumetric objects by voxelization [221].
In Section 7.5.1.1, we also show how the method is applicable to objects represented as scattered
point sets.

The methodology presented here is an enhancement and extension of the methods presented
in [3][43][138][245]. We explicitly compute a vector field over the entire object, similar to [138]
(not just along the path as in [3][43][245]): a 3D array where each voxel contains a vector value
(magnitude and direction). The difference from all the previously discussed approaches is that
once we compute this vector field, we use its topological characteristics [57][82][95] such as
critical points and low divergence points together with other user-selected seed points, to extract
our skeleton hierarchy. The curve-skeleton extraction framework described here is general, and
can be used to extract a curve-skeleton from various types of vector field functions such as
generalized potential [43], electrostatic [90], visible repulsive force [138][245].

The presented methodology has a number of significant advantages over the previous
methods, namely:

- it directly produces connected skeletons without the use of a re-connection step, if
  the resolution is sufficient,
- it works both on un-segmented objects (where only the boundary of the object is
  known) and segmented objects (where the interior is known),
- it has the ability to produce a strict hierarchy of skeletons (a curve-skeleton set) and
• it has the ability to automatically extract an “IK skeleton” that can be used for animation i.e., the joints are automatically identified.

7.1 The framework algorithm

The framework algorithm can be summarized in three steps as follows:

**Step 1** Compute a vector field over a discretization of the entire 3D object.

**Step 2** Detect the critical points of the vector field.

**Step 3** Extract the hierarchical curve-skeleton using a force following algorithm, with the following seed points:

**Step 3.1** (Level 0): Using the critical points as seeds (the “core” skeleton).

**Step 3.2** (Level 1): Using the lowest divergence voxels as seeds.

**Step 3.3** (Level 2): Using other user-selected seeds.

In Step 1, we compute a vector field over a discretization of the 3D object. If the object is given in polygonal representation, it can be voxelized. If a point set representation is available, the points can be mapped to a discrete grid. In this case however, the distinction between the interior and exterior of the object is not known. Let $O \subset Z^3$ be a discretization of a 3D object and let $vf : Z^3 \rightarrow R^3$ be a vector function, which computes a vector $\vec{v}$ for each voxel $p = (x, y, z)$ of the grid: $vf(p) = \vec{v}$. The vector field consists of the vectors computed at each voxel starting from the object voxels (or a subset of them) and it may be calculated only inside the object, or over the entire grid if the distinction between inside and outside cannot be made. If the distinction inside/outside is known, it is sufficient to compute the vector field only inside the object, since that is where we expect the curve-skeleton to lie. Additionally, all the subsequent steps should only consider the inside of the object. That is, we can let $vf$ return the zero vector for voxels outside the object: $vf(p) = \vec{0}$, if $p \notin O$. On the other hand, if the inside cannot be distinguished
from the outside, we must compute the vector field over the entire grid. In this case however, we must find a way to separate the curve-skeleton corresponding to the object, from the curve-skeleton corresponding to the background. We show in Section 7.5.1.1 how this can be achieved for a particular type of vector function, but note that this may not be possible for any type of vector function.

Once a vector field is computed, in Step 2, we locate the critical points of this vector field. A critical point is a point where the magnitude of the vector function vanishes, i.e., \( v_f(p) = 0 \). We will detail the procedure of locating critical points in a discrete vector field in Section 7.2, but several aspects should be noted here: 1) the position \( p \) is usually not at the integer locations of the voxel grid; 2) if the vector field is computed only inside the object, we should restrict our search for critical points inside the object as well.

In Steps 3 we compute the divergence of the vector field, at each voxel. The divergence measures the “sinkness” of a position in the vector field. More details can be found in Section 7.4.2.

The actual curve-skeleton is extracted in Step 3 in several phases, producing a hierarchy of increasingly complex curve-skeletons. The basic operation at this step is a force following procedure (Section 7.3) that starts at a seed point and follows the direction given by the underlying vector field until it reaches a position that was already visited, or when the force becomes zero, i.e. a critical point is reached. The force following procedure builds the curve-skeleton segments. In Step 3.1, the critical points of the vector fields are used as seeds. This generates the most basic curve-skeleton that can be extracted from the underlying vector field, the “core skeleton” (Section 7.4.1). In Step 3.2, the next level of the hierarchy (level 1) is obtained by using the lowest divergence points in the vector field as seeds (Section 7.4.2). Finally, the third level of the hierarchy (level 2) is achieved by using other user-supplied seed points as starting points for the force following procedure (Section 7.4.3). Below, we detail each of these steps.
7.2 Locating critical points

Critical points are important vector field topology components and are often used in vector field visualization. These are the points where the magnitude of the force vector vanishes ($vf(p) = 0$).

In 3D, we have three main types of critical points: attracting nodes and foci, repelling nodes and foci, and saddles. Figure 40 shows a schematic representation for these main types of critical points. The vector field flows toward an attracting node from all directions (Figure 40(a)) and away from a repelling node in all directions (Figure 40(b)). Around a saddle, the flow moves both toward and away from the critical point. Figure 40(c) shows the case where the flow is directed toward the saddle point in a horizontal plane, and away from the saddle in the vertical direction. Alternatively, the flow can move toward the critical point in a single direction (a line) and away from it in a plane, as shown schematically in Figure 40(d). In Figure 45, a critical point (attracting node) is visible in the middle of the head of the cow object.

Figure 40 Critical points in 3D. Attracting node (a); repelling node (b); saddle with two attracting directions and one repelling direction (c); saddle with one attracting direction and two repelling directions (d).
Note that the schematic representations in Figure 40 do not accurately represent the actual shape of the flow field around critical points. For example, in the case of attracting or repelling foci, the flow field exhibits a spiraling motion around the critical point. In [57][82][95][181], a full discussion of the visualization of vector-field topology and the different types of critical points can be found. In what follows, we present a brief description relevant to extracting the curve-skeleton.

Critical points are difficult to locate in a discrete vector field, particularly because they do not necessarily occur at the given sample locations (integer position of the voxel grid), but often occur in between sampling points. A good heuristic for detecting critical points is described in [82]: a zero in the vector field occurs when all 3 components (x, y and z) of the force vector vanish. Thus, if we can identify a region where each vector component changes sign, the region is a candidate for containing a critical point. In our case, the smallest region we can consider is a grid cell: a cube whose corners are formed by eight neighboring voxels in the grid. The force field value is inspected at each of the 8 corners of a grid cell. Cells containing both positive and negative values for every vector component are potential candidates for containing critical points. Candidate cells are recursively divided into 8 sub-cells and the candidacy test is repeated for each sub-cell. When inspecting the corners of a sub-cell, tri-linear interpolation can be used to estimate the values at the non-integer positions in the grid. The process ends either when a cell fails the candidacy test or when the cell is too small and is still a candidate, in which case a critical point is assumed to exist at the center of the cell, or a more elaborate scheme like Newton’s method starting somewhere inside the cell can be employed to find the exact location where the vector field becomes zero.

In general, it is desirable to identify the position of the critical point as accurately as possible for the next phase (classification). Furthermore, the fact that a cell is a candidate for containing a critical point does not mean that a critical point actually exists there. It may happen that each of the components of the vector field becomes zero inside the cell, but not all at the same time. In
case a critical point exists, Newton’s method guarantees fast convergence provided we are close enough to the real critical point. If a critical point does not exist in the cell, Newton’s method will quickly diverge from the initial guess.

Newton’s method in 3D can be described as follows: Let position $P$ be the initial guess:

$$
P = \begin{pmatrix}
  x \\
  y \\
  z
\end{pmatrix}
$$

In our case, this can be the center of the smallest candidate cell. Let $F$ be the value of the vector field at $P$, $F = \text{fv}(P)$. We can determine this value by tri-linear interpolation if $P$ is not at an integer grid location. Let $J(P)$ be the Jacobian matrix at position $P$:

$$
J(P) = \begin{pmatrix}
  \frac{\partial \text{fv}_x}{\partial x} & \frac{\partial \text{fv}_x}{\partial y} & \frac{\partial \text{fv}_x}{\partial z} \\
  \frac{\partial \text{fv}_y}{\partial x} & \frac{\partial \text{fv}_y}{\partial y} & \frac{\partial \text{fv}_y}{\partial z} \\
  \frac{\partial \text{fv}_z}{\partial x} & \frac{\partial \text{fv}_z}{\partial y} & \frac{\partial \text{fv}_z}{\partial z}
\end{pmatrix}
$$

where $\text{fv}_x$, $\text{fv}_y$, and $\text{fv}_z$ are the $x$, $y$ and $z$ components of the vector field function, $\text{fv}$.

To determine the next guess, we solve the following equation: $J(P) \cdot dP = -F$ for $dP$. Then the next guess is $P = P + dP$.

These steps are repeated until the force $F = \text{fv}(P)$ becomes zero, or, when the next guess takes us outside of the initial voxel cell. In the later case, we can conclude that a critical point does not exist in the given voxel cell.

Once extracted, the critical points are classified as: attracting nodes (where all vectors are pointing towards the critical point), repelling nodes (where all the vectors are pointing away from the critical point) and saddle points (where some vectors are pointing towards the critical point and others away from it). Critical points can be classified by evaluating the eigenvalues of the Jacobian matrix of the vector-field at the critical point. The eigenvalues and eigenvectors of the
Jacobian matrix at a position $P$ in the vector field can be determined from the following equation [115]:

$$J(P) \cdot v = \lambda \cdot v, \text{ or}$$

$$(J(P) - \lambda \cdot I) \cdot v = 0$$

where $\lambda$ is the eigenvalue, $v$ is the corresponding non-zero eigenvector, and $I$ is the identity matrix. For the second equation above to have non-trivial solutions ($v \neq 0$), the following condition needs to be satisfied:

$$\det(J(P) - \lambda \cdot I) = 0$$

The above equation can be used to determine all the eigenvalues as the roots of the polynomial in $\lambda$ and then the corresponding eigenvectors can be found from the previous equations.

According to the real and imaginary components of the eigenvalues of the Jacobian, critical points can be classified as follows:

- a positive real part of an eigenvalue denotes the existence of a repelling direction (given by the corresponding eigenvector)
- a negative real part denotes an attracting direction, and
- an imaginary part describes a spiraling motion around the point.

If all the real parts of the eigenvalues are of the same sign, the critical point is classified as an attracting (if negative) or repelling (if positive) node. The critical point is said to be a saddle if two real parts of the eigenvalues have the same sign and the third one has an opposite sign. The flow is directed toward the saddle point along the negative eigendirections (directions given by the eigenvectors of the Jacobian), and away from the critical point along the positive eigendirection (see Figure 40(c and d)).
7.3 Force-following

To extract the curve segments forming the curve-skeleton, we employ a force-following procedure, starting at various locations in the vector field. The force following algorithm, using a simple Euler integration scheme, can be summarized as follows:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( P ) = the initial position (the seed point)</td>
</tr>
<tr>
<td>2</td>
<td>( C = {P} ) = curve-skeleton segment identified so far</td>
</tr>
<tr>
<td>3</td>
<td>Evaluate the vector field value at ( P ): ( F = \text{fv}(P) )</td>
</tr>
<tr>
<td>4</td>
<td>While ( F \neq 0 ) and ( P ) was not visited before</td>
</tr>
<tr>
<td>5</td>
<td>( P = ) the next position in the direction of ( F )</td>
</tr>
<tr>
<td>6</td>
<td>( C = C \cup P )</td>
</tr>
<tr>
<td>7</td>
<td>Evaluate the vector field value at the new position: ( F = \text{fv}(P) )</td>
</tr>
</tbody>
</table>

We start with the seed point as the initial position. At each position \( P \), we evaluate the value of the vector field \( F \) at \( P \): \( F = \text{fv}(P) \), using tri-linear interpolation. If the vector field is zero at the current position, we have reached a critical point, and the algorithm terminates. Otherwise, we move in the direction pointed to by \( F \) with a small step. The next position is:

\[
P = P + \text{stepSize} \cdot \frac{\vec{F}}{\|\vec{F}\|}.
\]

Another possibility is that the force-following process leads outside the domain of the vector field (outside the discrete grid where the vector field was defined). If we consider that \( \text{fv}(P) = 0 \) if \( P \) is outside the domain, the algorithm terminates as soon as we reach the boundary of the domain.

This is basically a numerical integration scheme, where the vector field gives the instantaneous velocity and we try to find the trajectory of a mass-less particle that is freed in this field at the seed position. Alternatively, instead of a first-order evaluation of the next direction at position \( P \) (Euler scheme) [115], we can use a higher degree approximation, such as a Runge-Kutta second or fourth order integration scheme [115] to evaluate the direction in which the next
step should be taken. This would help reduce the integration error, allowing larger steps to be taken, but the computational cost increases. That is, instead of taking $F$ to be $F = f v(P)$, we can use a higher order estimation. For example, for a second order Runge-Kutta integration scheme (RK2), we would use the value of the vector field at $P$, $F_i = f v(P)$, to move only half-way towards the next position (half the step size) to point $P_h$: $P_h = P + \frac{1}{2} \cdot \text{stepSize} \cdot \frac{\vec{F}_1}{\|\vec{F}_1\|}$, evaluate the vector field at the mid-point $F_2 = f v(P_h)$, then use the force determined at the mid-point, $F_2$, as the direction in which the next step should be taken (the full step size this time): $P = P + \text{stepSize} \cdot \frac{\vec{F}_2}{\|\vec{F}_2\|}$. A fourth order Runge-Kutta scheme uses four function evaluations at each step, one at the beginning of the interval, two at trial mid-points, and one at the end of the interval; then computes the final direction as a weighted average of these vectors.

The efficiency of the force-following algorithm can be improved by using a variable step size, correlated with the change in direction observed at successive integration steps. If the direction remains approximately the same, we can use larger step sizes, if the direction changes a lot from one step to the next, smaller step sizes will reduce the error.

During the force-following algorithm, the initial seed point, the final point and all the intermediate positions reached during integration form a curve-skeleton segment $C$.

### 7.4 Extracting the curve-skeleton

The curve-skeleton is extracted from the underlying vector field, using the force-following procedure described above, starting at three different kinds of seed points: the critical points, low divergence points and user-defined seeds.

A hierarchy of increasingly complex skeletons can be obtained by using these different kinds of seed points in the following order: the most basic skeleton (the “core skeleton”) can be
obtained by using the critical points of the vector field as seeds for the force-following algorithm. The next level uses the low divergence locations in the vector field as seeds and adds more curve-skeleton segments to the core skeleton. The number of divergence seeds can be varied using a user-defined threshold, thus generating an entire family of skeletons under this hierarchy level. The final level uses user-defined seed points to be added to the previous levels. Details about this procedure can be found below.

7.4.1 Level 0 – the “core” skeleton

Critical points represent the singularities of the vector field. A special set of field lines, called integral curves [95], can be extracted from a vector field by integrating (forward and backward) along the eigendirections at each critical point. The “core skeleton” is the subset of integral curves that connect all the critical points in the vector field. Each core skeleton segment must start and end at a critical point.

The “core skeleton” can be obtained by using only forward integration, along the outgoing eigendirections starting at each critical point and keeping only those skeleton segments that end at another critical point. In a backward integration scheme, starting at a critical point, we would follow the incoming eigendirections backwards, in other words, we would determine the direction in which we should take the next step as before, and then take the actual step in the exact opposite direction: \( P = P + \text{stepSize} \cdot \frac{-\vec{F}}{||\vec{F}||} \). When reaching another critical point, since we are moving against the direction of the vector field, we would be approaching it from a direction where the flow moves away from the point, a direction we would normally follow in a forward integration scheme starting at the critical point we are approaching. Since we are only considering segments that begin and end at a critical point as core skeleton segments, there is no need for both forward and backward integration.
The force-following algorithm as described above cannot be used with a critical point as the initial position, because the vector field is zero there and we will not be able to take the first step to the next position. Therefore, when starting at a critical point, the first step will be taken differently: instead of evaluating the value of the vector field at the current position, we directly use the eigendirections determined at the critical point (the eigendirection corresponding to a positive eigenvalue indicates the outgoing direction). After the first step, as we moved away from the critical point and the vector field value is no longer zero, we can follow the usual force-following procedure.

Because of resolution issues, it is very unlikely that the last step of the force following procedure will take us at the exact position of the other critical point, where the vector field value is zero and the algorithm can terminate normally. For example, if we are approaching an attracting node, we may overshoot it in one step, then we turn back when computing the next position and we may overshoot it again, and so on. The result is that we will not advance anymore, but we may never hit the exact position of the critical point where the force is zero because of the non-zero step size. To avoid such problems, at each step we should check whether we are sufficiently close to a critical point, and in that case, stop the force-following algorithm.

Figure 41 shows the “core” skeleton of a cow object. In the figure, critical points are shown as red spheres: attracting nodes are depicted as larger red spheres, while saddle points are shown as smaller red spheres. The locations sampled by the force-following algorithm (representing the “core” skeleton) are shown as blue spheres. A slice of the underlying vector field is also shown as light blue arrows.

The core skeleton represents level 0 in the skeleton hierarchy. However, the core skeleton captures only part of the given object. This can be observed in Figure 41, where the ears, horns, tail and front of head of the cow are not represented by curve-skeleton segments. The next level-of-detail is added by considering another important feature of the force field: the divergence.
Critical points and “core” skeleton of the cow object. Attracting nodes are shown as larger red spheres, saddle points are shown as smaller red spheres. The skeleton obtained by force-following is shown in blue. A slice of the underlying vector field is represented as light blue arrows.

7.4.2 Level 1 – the divergence skeleton

For level 1 of the curve-skeleton hierarchy, we use the divergence of the vector field to select seed points for the force-following algorithm. Varying the divergence threshold creates an entire hierarchy of curve-skeletons.

The divergence of a vector field in a given region of space is a scalar quantity that characterizes the rate of flow leaving that region. Divergence is an interesting property because it measures the “sinkness” of a point [32]. A negative divergence value at a point indicates that the flow is moving mainly towards the given point. Divergence can be computed at a grid point using the formula:

\[
\text{div}(\mathbf{f}) = \nabla \cdot \mathbf{f} = \frac{\partial f_x}{\partial x} + \frac{\partial f_y}{\partial y} + \frac{\partial f_z}{\partial z}
\]

where \(f_x, f_y, f_z\) are the vector field components in the \(x, y\) and \(z\) direction.

Of interest are points with low divergence values, which indicate a “sink”. To select seed voxels based on divergence, one can use a “local minimum” criterion, selecting those voxels that have smaller divergence value than any of their neighbors, or a threshold. The threshold can be
given as a percentage of the maximum computed divergence value. Note that the user can determine using these criteria how many new seed points are identified.

Each of the selected seeds will be the starting point of a force-following procedure identifying a new curve segment that will be added to the existing skeleton. In this case, the force-following algorithm will usually terminate when a previously visited location is encountered, since part of the curve-skeleton was already determined in the form of the core skeleton and the segments discovered while using seeds with lower divergence value than the current one.

Note that, by varying the divergence threshold, a strict hierarchy is obtained only if the divergence seeds are always processed in the order of their divergence value (smaller values first). By a strict hierarchy, we mean that each curve-skeleton obtained using a divergence threshold value $t$, is a subset of any other curve-skeleton obtained using a divergence threshold larger than $t$. This is true because the force-following algorithm stops when encountering a previously visited location, effectively splicing the new curve segments onto the existing curve-skeleton.

The curve-skeletons determined using divergence seeds form level 1 of the hierarchy. A few examples of level-1 skeletons are shown in Figure 42. On the left, the low divergence seeds are shown as larger light-blue spheres. The positions sampled by the force-following algorithm are shown as smaller blue spheres. In the two images on the right of Figure 42, more divergence seeds were added (not shown) and only the resulting curve-skeleton is shown as blue spheres.

### 7.4.3 Level 2 – the user skeleton

For level 2 of the hierarchy, we can use a set of user-defined seed points as starting point for the force-following algorithm. As in the case of divergence seeds, the force-following procedure stops when encountering a previously visited location, effectively splicing the new curve segments onto the existing curve-skeleton.
A possible choice for user-defined seeds is to select high curvature [93] voxels on the boundary of the object. Using a user-defined curvature threshold, or a local minimum criterion, we can select new seed points and use them to develop another level of the curve-skeleton hierarchy. Note that, as in the case of high divergence seeds, in order to obtain a strict hierarchy, the user-defined seeds need to be processed in the same order every time the set of user-defined seeds changes. For the case of high curvature seeds, they would have to be processed in increasing order of the curvature estimation for example.

The user-defined seeds can be added either directly to the core skeleton, or to a level 1 divergence skeleton. However, in order to have a strict hierarchy, all the levels below the current one need to be fixed and completely processed before processing any seeds at the current level. For example, the set of divergence seeds needs to be fixed and completely processed before allowing the curvature threshold to vary and processing any curvature-based seeds. This ensures that the same level 1 curve-skeleton is always a subset of any level 2 curve-skeleton. Thus, every level 1 curve-skeleton can be the root of an entire level 2 hierarchy obtained by varying the
curvature threshold. Figure 43 shows one curve-skeleton from each level of the hierarchy obtained from a model of the letter K.

Figure 43 A 3D model of the letter K with varying curve-skeletons. (a) 3D model, (b) core skeleton using only critical points, (c) adding low divergence values (top 20%), (d) adding boundary seeds at high curvature boundary locations.

A disadvantage of boundary seeding based on the curvature is that it is affected even by small amounts of noise present on the object’s boundary. A possible solution to this problem is to consider an extended neighborhood when computing the curvature at boundary voxels, not just the face, edge and vertex neighbors [93]. For certain applications, such as virtual navigation, user-defined seeds could be used to specifically generate certain navigation paths.

7.5 Types of vector fields

In this section we present two types of vector fields we have experimented with. The potential field based repulsive function, first used in [3][43] is based on a generalization of the Newtonian potential. The normal diffusion field function was designed to mimic the potential field function, and remove its most serious limitations: computational complexity and the visibility problem.

7.5.1 The potential field based repulsive function

The potential field based repulsive force is defined as the repulsive force of the generalized potential field [3][43]. The key idea behind the potential field approach is to generate a force field inside the object by charging the object’s boundary. The basic procedure to compute the
generalized potential and force for polyhedral objects is summarized in [43]. Since our algorithm operates on 3D objects represented by voxels, our boundary elements are also voxels, and we consider them to be point charges (this also simplifies the calculations involved in computing the force field). A boundary point is defined as an object voxel that has an exterior (empty) neighbor. An interior voxel (interior point) is an object voxel whose neighbors are all object voxels.

The repulsive force at a point due to a nearby point charge is defined as a force pushing the point away from the charge with a strength that is inverse proportional to a power of the distance between the point and the charge namely:

$$F_{PC} = \frac{C \cdot P}{R^m}$$  \hspace{1cm} (1)

where \( F_{PC} \) is the repulsive force at point \( P \), due to point charge \( C \), \( C \cdot P \) is the normalized vector from \( C \) to \( P \) which gives the direction of the force, \( R \) is the distance between \( P \) and the charge \( C \) and the power \( m \) is called the order of the force function (\( m=2 \) for the Newtonian force).

The force at a point \( P \) due to the influence of multiple point charges \( C_i \) can be found discretely by simply summing all the forces at \( P \).

$$\overrightarrow{F}_p = \sum \overrightarrow{F}_{PC_i}$$  \hspace{1cm} (2)

where \( \overrightarrow{F}_p \) is the resulting force at point \( P \) and \( \overrightarrow{F}_{PC_i} \) are the forces due to the individual point charges \( C_i \).

We consider every boundary voxel to be a point charge and the repulsive force at each interior voxel is explicitly computed by summing the influences of all the point charges. The process is illustrated in Figure 44.
Figure 44  Construction of the repulsive force field. Charges $C_1$ and $C_2$ placed on the boundary exert a repelling force on interior point $P$, inverse proportional with the distance between the charge and point $P$. The resulting force at $P$ ($F_P$) is the sum of the forces exerted by individual point charges.

Figure 45 shows the repulsive force field along the center slice of a 3D cow model.

A high order of the force function ($m$) will cause the local boundary points to have a higher influence on a given interior point than the more distant boundary voxels, thus creating a vector field with sharper path-lines because it follows the local boundary topology more closely. A low value for the $m$ parameter will produce a smoother vector field, with more rounded corners, since the vector direction at a particular point is now influenced by more boundary charges [3].

Figure 45  A slice of the repulsive force field of a 3D cow model. The vectors at each point are shown with arrows.
Figure 46 illustrates the effect of the field strength parameter on the resulting curve-skeleton of an “L”-shaped object. Note how, in the “elbow” area, the curve-skeleton is smoother for smaller values of the $m$ parameter and, as $m$ increases, the curve-skeleton becomes less smooth (Figure 46(a)-(c)).

As the curve-skeleton segments approach the “elbow” region the boundary symmetry breaks and the curve-skeleton deviates from the middle of the shape. This behavior is also visible on the medial axis shown in Figure 46(d), although the effect is less pronounced. For $m=3$ (Figure 46(a)), the vertical boundary region on the inside of the elbow, causes the curve-skeleton to shift downwards as it approaches the elbow. This is because all the charges on that vertical boundary section will generate repulsive forces pointing downwards. For $m=6$ and $m=10$ however (Figure 46(b) and (c)), fewer points on that vertical boundary section will have a significant influence on the curve-skeleton. Therefore, the skeleton will shift upwards, under the effect of the repulsive forces pointing upwards, generated by the horizontal boundary below, which is closer.

Also note that the branches of the curve-skeleton become shorter as the strength of the field is increased. As $m$ increases, and the effect of more distant boundary charges becomes negligible, the curve-skeleton is increasingly influenced by local symmetries. This is why, for $m=10$, the skeleton branches seem to end exactly in the middle of the horizontal and vertical sections of the “L”-shaped object.
Each different value of the field strength parameter \((m)\) will produce a different curve-skeleton set. Once the value of \(m\) is fixed, an entire hierarchical family of increasingly complex curve-skeletons can be obtained as level 1 and level 2 curve-skeletons using this framework.

Because the algorithm uses all of the boundary points for force computation, visibility errors can result for objects with tapered limbs (like a comb) since individual points within each prong of the comb cannot “see” boundary points on the other prongs and those points should not be considered in the force field calculation [3][43]. This can be overcome by determining the visibility with a line-of-sight calculation, which checks to see that the surface is not pierced when connecting a point to a charge with a straight line. While this is more accurate, it unfortunately increases the running time substantially and is only applicable if the surface is known. Using a high order of \(m\) also minimizes this effect [3] by reducing the influence of more distant boundary charges.

The computational complexity of the force field calculation depends on the number of object and boundary voxels: \(O(No \times Nb)\) where \(No\) is the number of object voxels and \(Nb\) is the number of boundary voxels. Since \(Nb\) is a fraction of \(No\), the computational complexity is approximately \(O(No^2)\). This is the most time consuming step of the algorithm, accounting for about 98% of the total running time.

In Figure 47 we show several examples of core and level 1 curve-skeletons for several 3D objects. The parameters used to obtain these skeletons (potential field strength \((m)\) and percentage of divergence seeds \((div)\)) are given below each curve-skeleton.

Note that the core skeleton is not affected very much even by a substantial amount of noise added to the boundary of the object. As the core skeleton is the most centered part, the influence of boundary noise is usually averaged out. Once divergence seeds are added to the curve-skeleton, since they are closer to the surface, their location will be affected by noise, as seem for both the knight and the ET objects. The effect of boundary noise can be reduced by choosing a lower values for the field strength parameter \((m)\), thus allowing more distant boundary charges to
contribute to the final field value at an interior point (as shown for the two noisy examples above). However, in very thin regions of the object, even a low value of the $m$ parameter cannot counteract the effects of severe noise and the effect is visible as disconnected or jagged skeleton segments, as is the case of the ET with noise in Figure 47.

<table>
<thead>
<tr>
<th>Core</th>
<th>Level 1</th>
<th>Core</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horse</td>
<td></td>
<td>Spider</td>
<td></td>
</tr>
<tr>
<td>m=4</td>
<td>m=4</td>
<td>m=4</td>
<td>m=4</td>
</tr>
<tr>
<td>div=35%</td>
<td></td>
<td>div=50%</td>
<td></td>
</tr>
<tr>
<td>Bunny</td>
<td></td>
<td>Girl with ponytail</td>
<td></td>
</tr>
<tr>
<td>m=3</td>
<td>m=3</td>
<td>m=4</td>
<td>m=4</td>
</tr>
<tr>
<td>div=20%</td>
<td></td>
<td>div=40%</td>
<td></td>
</tr>
<tr>
<td>Dinosaur</td>
<td></td>
<td>Hand</td>
<td></td>
</tr>
<tr>
<td>m=4</td>
<td>m=4</td>
<td>m=4</td>
<td>m=4</td>
</tr>
<tr>
<td>div=35%</td>
<td></td>
<td>div=30%</td>
<td></td>
</tr>
<tr>
<td>Castle</td>
<td></td>
<td>Tree</td>
<td></td>
</tr>
<tr>
<td>m=6</td>
<td>m=6</td>
<td>m=4</td>
<td>m=4</td>
</tr>
<tr>
<td>div=30%</td>
<td></td>
<td>div=20%</td>
<td></td>
</tr>
<tr>
<td>Object</td>
<td>m</td>
<td>div</td>
<td>Object</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---</td>
<td>-----</td>
<td>----------------------------</td>
</tr>
<tr>
<td>Hammer</td>
<td>4</td>
<td>15%</td>
<td>Commercial airplane</td>
</tr>
<tr>
<td>Biplane</td>
<td>8</td>
<td>20%</td>
<td>Helicopter</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>40%</td>
<td></td>
</tr>
<tr>
<td>Bishop</td>
<td>4</td>
<td>30%</td>
<td>Colon</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>30%</td>
<td></td>
</tr>
<tr>
<td>Knight</td>
<td>4</td>
<td>20%</td>
<td>Knight with 20% added noise</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>30%</td>
<td></td>
</tr>
<tr>
<td>ET</td>
<td>6</td>
<td>20%</td>
<td>ET with 20% added noise</td>
</tr>
</tbody>
</table>

Figure 47 Curve-skeletons of several objects. (a) core skeleton, (b) level 1 curve-skeleton. The parameters are given next to each curve-skeleton.
The time to determine the skeleton is dominated by the repulsive force field computation. It ranged from about one minute for most of the datasets to about a half an hour for the colon dataset (on a standard PC). Speedup of the computation can be achieved by sampling the boundary (i.e., using only a fraction of the boundary points for force field generation) or by using a less accurate distance computation in place of Euclidean distance.

The order of the repulsive force function \( (m) \) has a dominant role in the skeleton outcome. It shapes the vector field, determining the location of critical points and divergence seeds. While the force function is object dependent, experimental results showed that a high order \( (m > 4) \) produced stable results.

7.5.1.1 Skeletons from scattered point sets using the repulsive force function

An advantage of the repulsive force function is that it can be used to extract curve-skeletons from objects represented as a scattered point set, where the surface is not known. Such data is usually obtained from 3D scanners. Although a discrete grid is needed for the computation of the repulsive force field, it is not necessary to know what is inside vs. outside the object or to have the exact boundary of the object specified. Only a set of surface points used as point charges is necessary to compute the vector field. The sample points are mapped to a grid and the force field is computed on the entire grid (inside and outside the object, since these distinctions are not known a-priori).

When computing the critical points in the vector field, between every boundary sample and each of its neighbors, we will obtain a critical point, either a saddle or an attracting node. However, these critical points are really on the surface of the object and should not be part of the core skeleton. They are artificially created by the sampled nature of the boundary. One way to filter out these unwanted critical points is to impose a lower bound on the minimum distance between a critical point and a boundary sample, based on the sampling resolution of the boundary. If the critical point is closer to a boundary sample than this limit, it should be ignored.
The curve-skeleton can then be determined using the procedure described above. The resulting curve-skeleton will represent both the object and the background. However, if the filtering step removed all critical points on the boundary, the skeleton of the object and the skeleton of the background should be completely separate from each other. Note that the curve-skeleton corresponding to the background may in fact be composed of several connected components. Furthermore, the background skeleton can be easily identified because its segments always reach the boundary of the voxel grid domain (the bounding box of the dataset). In Figure 48 we show the core and level-1 skeletons (c) for a horse dataset represented by scattered points on the surface. Curve-segments that touch the bounding box are classified as belonging to the curve-skeleton of the background and can be removed to reveal the curve-skeleton of the object (d).

![Figure 48](image)

**Figure 48** Curve-skeleton from scattered points: (a) scattered point set, (b) “core” skeleton. (c) skeleton inside (in black) and outside the object (in gray) using top 20% divergence points as seeds. (d) “inside-only” skeleton obtained from (c) by removing skeleton segments touching the bounding box.

Note that critical points can also occur outside the object for certain types of objects. One example is a torus, where an attracting node will be detected at the center of the shape, but outside the object. In that case, the core skeleton is not necessarily completely inside the object as shown in Figure 48(b). Nevertheless, if critical points on the surface are ignored, the inside and the outside core skeletons will be completely separated, and the outside part can be removed as described above.
7.5.2 Normal diffusion field (NDF)

One of the main drawbacks of the potential field based repulsive force approach is its computational complexity resulting from the fact that all boundary points need to be considered when computing the vector field value at an interior voxel. Furthermore, there are visibility problems, and if visibility is taken into account, the running time increases even more.

We propose a new type of vector field generating function, based on diffusing surface normals towards the interior of the object. The new function tries to mimic the positive aspects of the potential-based repulsive function, namely the averaging effect, but at the same time, improve the computational cost. Since an analytical expression of this function is difficult to extract, and certainly less intuitive, we present an algorithm that describes the steps toward computing a vector for each object voxel. This algorithm is applicable to objects represented as voxels on a discrete grid, where the distinction between the inside and the outside of the object is known. The algorithm assumes the entire object interior is composed of object voxels and will compute a vector value for each voxel belonging to the object.

Let $O$ be the 3D object, defined using a discrete binary function $F_O : \mathbb{Z}^3 \rightarrow \{0, 1\}$:

$$F_O(p) = \begin{cases} 1, & \text{if } p \in O \\ 0, & \text{otherwise} \end{cases}$$

A surface voxel $sp$, is an object voxel that has at least one background face neighbor, more formally, $sp$ is a surface voxel if $F_O(sp) = 1$ and $\exists p \in N_6(sp)$, such that $F_O(p) = 0$. An interior voxel, $ip$ is an object voxel whose entire set of face neighbors consists of object voxels: $ip$ is an interior voxel if $F_O(ip) = 1$ and $F_O(p) = 1$, $\forall p \in N_6(ip)$.

We start with the observation that near the surface of the object, the repulsive force field is normal to the surface. This is due to the influence of the neighboring charges on each boundary voxel which makes the vector component along the boundary vanish. So the first step of the algorithm is to compute the surface normal at each voxel on the surface of the object. The normals are computed such that they are oriented toward the interior of the object (Figure 49).
More formally, for each surface voxel \( sp \), we compute the inward surface normal \( N(sp) \) as the gradient of the function defining the object: \( N(sp) = \nabla FO(sp) \). Figure 49(a) shows the inward surface normals computed using this procedure, by evaluating the gradient of \( FO \) in the immediate 6-neighborhood of the surface voxel \( sp \). Note how, compared to the vectors computed near the surface by the repulsive force function (Figure 45), the vector field along the surface defined by these inward surface normals is not as smooth. In order to improve smoothness along the surface, we employ a Gaussian smoothing, using a smoothing kernel of user-defined size. The weights of a Gaussian smoothing kernel, centered at a voxel \( p \) and having a standard deviation \( \sigma \), are given by the following density function:

\[
wf(q) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(q-p)^2}{2\sigma^2}}
\]

Since the values beyond three standard deviations from the center (voxel \( p \)) are practically zero, the radius of the kernel can be taken to be \( 3\sigma \). Figure 49(b) shows the resulting inward surface normals after smoothing them with a Gaussian kernel of radius 5.

**Figure 49**  Computing the normal diffusion field – Step 1. (a) Inward surface normals, computed as the gradient of the function defining the voxelized object. (b) Smoothed surface normals.
Note that the same result obtained using a smoothing kernel of radius \( k \), can be obtained by applying a kernel of radius one, \( k \) times. This simplification allows us to compute a single kernel, of radius one, and successively apply as many times as needed in order to obtain the smoothing effect of any kernel radius. An alternative is to compute the inward normals using a larger neighborhood for each surface voxel, effectively doing a smoothing while computing the normals.

The first step computes a vector for each surface voxel. Next, we will compute a vector for each of the remaining voxels in the object, by averaging the vector field at the already processed voxels. For this, we need to order the remaining voxels by proximity to the object’s surface and process them in that order. To order the voxels, we use a front propagation technique (onion peeling approach), with the surface of the object being the initial front position. The front advances one position towards the interior of the object at each step, direction given by the normal to the front. At each new position, we compute the vector for all the voxels on the front as the weighted average of their immediate neighbors that were processed in one of the previous steps. This is similar to the fast marching methods for front propagation [208][231]. Voxels on the front can be processed in any order. Figure 50 illustrates the front-propagation process. The successive front positions are shown in different colors and each voxel shows the step in which it is being processed. Note that the front advances to the set of face neighbors of the voxels that were on the front in the previous step.

Note that the front propagation is used only as a means of ordering the object voxels for processing, in successive layers from the surface of the object toward the interior. As a result, we are not concerned with topology changes and quench points that occur during front propagation [208]. Also, voxels on the current front can be processed in any order since their resulting vector value is a function of the vectors computed in one or more of the previous steps.
The entire algorithm for computing the normal diffusion field is given below: let $OS$ be the set of surface object voxels and $OI$ be the set of interior object voxels. Then the entire object $O = OS \cup IO$. And let $c(v)$ be a counter associated with each object voxel $v$.

1. $Q = \text{empty set}$
2. For all $v$ in $SO$
3. $fv(v) = \text{gradient}(FO(v))$
4. $c(v) = 0$
5. $Q = Q \cup \{v\}$
6. For all $v$ in $IO$
7. $c(v) = \infty$
8. while $Q$ is not empty
9. $v = \text{extractMin}(Q)$
10. For each face neighbor $w$ of $v$ in $IO$
11. $c(w) = c(v) + 1$
12. $IO = IO \setminus w$
13. For each 26-neighbor $z$ of $w$ such that $c(z) < c(w)$
14. $fv(w) = fv(w) + g(z, w) \cdot fv(z)$
15. $Q = Q \cup \{w\}$

To implement the front propagation scheme, we associate a counter $c(v)$ to each object voxel $v$, representing the time when the advancing front reaches that voxel. The counter is initialized to
0 for all surface voxels (line 4) and to ∞ (or a very large value in practice) for all the rest of the object voxels (lines 6 and 7), meaning they were not yet reached by the front. The voxels on the current front position are inserted into a min priority queue Q. Then, as each voxel is removed from the priority queue, its un-processed face neighbors are given a counter value (line 11), a vector value (lines 13,14) and are added to the queue (line 15). The weights used to compute the vector at a given position \( g(z, w) \) in line 14) are again given by a Gaussian distribution.

The computational complexity of this algorithm is \( O(n) \), where \( n \) is the number of object voxels. This can be easily seen from the algorithm given above, where each voxel is only once inserted and removed from the priority queue, and when the priority queue is empty, the algorithm terminates.

The reduced computational complexity is only one of the goals we set to achieve with the normal diffusion field. We also wanted to obtain a similar averaging effect which gives such function their robustness to noise. For the normal diffusion field, an averaging effect is obtained because the vector value at each interior voxel is computed as a weighted average of the vectors computed at the previous front propagation steps. Figure 52 illustrates this dependency: a small arrow originating at one voxel and ending at another signifies that the vector value of the second voxel is influenced by the vector value of the first. Finally, through the front propagation scheme,
the force at an interior voxel depends on the forces computed at a number of surface voxels, shown with yellow outlines.

![Image](image_url)

**Figure 52** Illustration of the averaging effect of the normal diffusion field

Clearly, the closer an interior voxel is to the center of the object, the larger the surface area that influences it. But the surface area influencing any interior can be further enlarged by taking advantage of the second step of the normal diffusion algorithm: smoothing of the surface normals. After smoothing the surface normals with a kernel of radius $k$, every surface vector is now influenced by all other surface normals up to a distance of $k$ on the surface of the object. Ultimately, this amounts to effectively increasing the surface area which influences any interior voxel. This is shown in Figure 52 as the additional surface voxels shown with red outlines which influence the interior voxel in blue if a smoothing of radius 3 is applied to the surface normals prior to front propagation.

As the amount of implicit averaging, during the front propagation steps is limited by the thickness of the object, this may lead to problems for certain objects. Let us take the example of a thin plate (Figure 53). Objects with flat parts are particularly difficult to skeletonize for algorithms based on the distance transform, because as shown in Figure 2, all the points on an entire medial surface patch have the same distance to the surface. If we compute a vector field as the gradient of the distance function, this translates into all the points on that medial surface patch
being critical points since the gradient will be zero (Figure 53(b)). This obviously cannot produce a one dimensional curve-skeleton. The front propagation scheme in itself (without additional smoothing of surface normals) is another method of computing the distance field of an object. As shown in Figure 53(c), the critical points resulting from the normal diffusion field computed without surface normal smoothing are the same as the ones computed from the gradient of the distance field (1050 critical points). However, as surface normal smoothing is employed, this medial surface patch is effectively thinned down until finally it becomes one-dimensional Figure 53(d), (e) and (f). The number of critical points identified in the resulting normal diffusion field is reduced from 1050 (with no smoothing), to 682, 250 and finally 21 when applying surface normal smoothing with a smoothing kernel of radius 2 (Figure 53(d)), 5 (Figure 53(e)) and 7 (Figure 53(f)) respectively.

![Figure 53](image)

**Figure 53** Critical points computed from a thin plate object. (a) the thin plate object, (b) using the gradient of the distance function as the vector function, (c) using the normal diffusion field with no surface smoothing, (d) using NDF with surface smoothing with a kernel of radius 2, (e) using NDF with surface smoothing of radius 5, (f) using NDF with surface smoothing of radius 7.

Clearly the amount of smoothing of surface normals required to obtain a one-dimensional curve-skeleton is dependent of the original object. For example, for a thinner plate than the one in Figure 53, the necessary smoothing kernel will be larger. However, too much smoothing is not
desirable either, because with more and more smoothing, the resulting vector field represents the original object less and less.

In Figure 54, we show several examples of curve-skeletons obtained using the normal diffusion field for the same 3D objects used to exemplify the potential field based repulsive function. The parameters used for each case (number of steps of smoothing (sms) and percentage of divergence seeds (div)) are shown next to the respective curve-skeleton.

Note that, the resulting curve-skeletons are very similar to the ones obtained using the potential field based vector field. Notice however that the normal diffusion field is more sensitive to noise than the potential field, since the averaging effect is not as large. This can be seen for the ET shape with noise, where the core skeleton is disconnected. In the figure, we used the same amount of smoothing for the original objects as well as for the objects with noise (the knight and the ET objects). Obviously the effect of boundary noise can be reversed by increasing the amount of smoothing of surface normals before the propagation step.

<table>
<thead>
<tr>
<th>Core</th>
<th>Level 1</th>
<th>Core</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horse</td>
<td>sms=20&lt;br&gt;div=25%</td>
<td>Spider</td>
<td>sms=30&lt;br&gt;div=30%</td>
</tr>
<tr>
<td>Bunny</td>
<td>sms=5&lt;br&gt;div=10%</td>
<td>Girl with ponytail</td>
<td>sms=20&lt;br&gt;div=30%</td>
</tr>
<tr>
<td>Dinosaur</td>
<td></td>
<td>Hand</td>
<td></td>
</tr>
<tr>
<td>Castle</td>
<td>Tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image1" alt="Castle" /></td>
<td><img src="image2" alt="Tree" /></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image3" alt="Castle" /></td>
<td><img src="image4" alt="Tree" /></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image5" alt="Castle" /></td>
<td><img src="image6" alt="Tree" /></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hammer</th>
<th>Biplane</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image7" alt="Hammer" /></td>
<td><img src="image8" alt="Biplane" /></td>
</tr>
<tr>
<td><img src="image9" alt="Hammer" /></td>
<td><img src="image10" alt="Biplane" /></td>
</tr>
<tr>
<td><img src="image11" alt="Hammer" /></td>
<td><img src="image12" alt="Biplane" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Commercial airplane</th>
<th>Helicopter</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image13" alt="Commercial airplane" /></td>
<td><img src="image14" alt="Helicopter" /></td>
</tr>
<tr>
<td><img src="image15" alt="Commercial airplane" /></td>
<td><img src="image16" alt="Helicopter" /></td>
</tr>
<tr>
<td><img src="image17" alt="Commercial airplane" /></td>
<td><img src="image18" alt="Helicopter" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bishop</th>
<th>Colon</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image19" alt="Bishop" /></td>
<td><img src="image20" alt="Colon" /></td>
</tr>
<tr>
<td><img src="image21" alt="Bishop" /></td>
<td><img src="image22" alt="Colon" /></td>
</tr>
<tr>
<td><img src="image23" alt="Bishop" /></td>
<td><img src="image24" alt="Colon" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Knight</th>
<th>Knight with 20% added noise</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image25" alt="Knight" /></td>
<td><img src="image26" alt="Knight with 20% added noise" /></td>
</tr>
<tr>
<td><img src="image27" alt="Knight" /></td>
<td><img src="image28" alt="Knight with 20% added noise" /></td>
</tr>
<tr>
<td><img src="image29" alt="Knight" /></td>
<td><img src="image30" alt="Knight with 20% added noise" /></td>
</tr>
</tbody>
</table>
Since the right amount of smoothing of surface normals is different for each object and it can be a long trial-and-error process, we present two heuristics for automatically determining the required amount of smoothing. The first heuristic is based on a closer observation of the thin plate example above. As long as we have neighboring surface normals that have the same direction, we are probably dealing with a flat part of the object which will generate a medial surface patch in the form of a lot of neighboring critical points. The amount of smoothing required depends on the actual thickness of the plate-like region, but a simple approach is to smooth until no two neighboring surface normals have the same direction. The difference in direction of two vectors can be easily evaluated by computing the dot product of the normalized versions of the two vectors. A second heuristic approach is based on the observation that the resulting skeleton cannot be one-dimensional as long as we have neighboring critical points. So, we smooth the surface voxels until no two identified critical points are immediate 26-neighbors. While this approach is more expensive computationally because after each smoothing step we need to re-compute the critical points, it guarantees that the resulting skeleton will be one-dimensional.
Various ways to improve the performance of this heuristic approach can be applied, such as: abandoning the search for critical points as soon as two neighboring critical points are found, doubling the smoothing kernel size at each step and then as soon as the condition on critical points is met, reduce it until we find the smallest sufficient value. The results of the two heuristics are shown in Figure 55 on several objects.

**Figure 55** Two heuristics for automatically determining the sufficient amount of surface normal smoothing. (a) no two surface normals have the same direction; (b) no two critical points are 26-neighbors.
Note that, given the limited precision of computer arithmetic, none of the heuristics presented here have any guarantees that the terminating condition will be met, so, in practice, a maximum number of iterations should be set. In the first case, even using double precision may not be sufficient to differentiate between two neighboring normals, while in the second case, the existing precision may not be sufficient to accurately identify the critical points. Nevertheless, these heuristics are useful for reducing the number of trial-and-error iterations before a one-dimensional curve-skeleton can be extracted using the normal diffusion field.

Note also that each of these heuristics require a smooth-and-test approach, meaning that we smooth the surface normals using a smoothing kernel of a specific size, test for the terminating condition, and if the condition is not met, we increase the radius of the smoothing kernel and repeat. Thus, for efficient implementation, it is important to use the successive application of a fixed size smoothing kernel (radius one for example). After each application, we can test for the terminating condition; this way, we do not restart the smoothing process from the original vector field each time.

### 7.6 Skeleton representations

The skeleton produced by the algorithm consists of a set of seed points connected by points sampled by the force-following algorithm. Alternatively, each curve sample can be mapped to the nearest grid location, creating a less smooth voxel skeleton, consistent with the discretized nature of the original 3D object. Figure 56(a) shows the curve representation and Figure 56 (c) shows the corresponding voxel skeleton of a cow model.
Another approach, useful for animation [75][239] or matching [49][223], is to transform the skeleton into straight-line segments by treating the critical points and the points where different skeleton segments meet as “joints” and connecting these points with straight lines. The resulting skeleton will be suitable for importing into commercial animation packages (Maya [155], 3D Studio Max [1], etc) since the joints are automatically detected (joints are the end-points of skeleton segments). Figure 57(a) shows the curve-skeleton of the cow objects, highlighting the automatically identified joints with blue spheres. In Figure 57(b), we show the IK-skeleton, obtained by connecting the joints with straight-line segments. In Chapter 8 we show how the “skinning” process (attaching surface polygons to the skeleton) can also be done automatically using the repulsive force field.

Figure 56  Skeleton representations for the cow model (a): curve skeleton (b); voxel skeleton (c).

Figure 57  Construction of the IK-skeleton of the cow model. Curve-skeleton with automatically identified joints, shown as green spheres (a), IK-skeleton obtained by connecting the joints with straight lines (b).
7.7 Discussion

The curve-skeletonization framework was tested on many different volumetric datasets containing various 3D objects both originally volumetric and voxelized polygonal objects, some of which can be seen in Figure 47 and Figure 54. More objects can be seen at [242].

The divergence and the curvature thresholds should be chosen based on the application of the resulting skeleton. There are no optimal values for these parameters, as different applications may require skeletons of different complexity. The simplest skeleton our algorithm can extract is the core skeleton, using no divergence or curvature seeds. If a more complex skeleton is desired, the user can increase the number of divergence seeds, or the number of curvature seeds or both, keeping in mind that curvature seeds are very sensitive to noise on the boundary of the object.

Resolution/discretization is an important external factor affecting the performance of the algorithm and changing the accuracy of the solution, especially for scattered point sets. Clearly a $4^3$ grid will yield a less accurate result than a $60^3$ grid. A 1 or 2 voxels thick object region will not have enough resolution to properly compute the repulsive force field in that region (this is a common simulation issue and standard multiresolution solutions can be employed [16]). For such regions, the repulsive force field is extremely sensitive to even small boundary perturbations, resulting in discontinuities of the flow pattern inside the object’s boundaries. For the normal diffusion field, the smoothing of surface normals step will preserve the averaging effect even in thin regions, but this will be the only source of averaging since the front propagation will only take a few steps in thin regions. As a result, the interior skeleton can become disconnected just because the force-following algorithm becomes stuck in these perturbed regions of the force field. To overcome this problem, one can either increase the resolution of the voxelization or pad the object with a number of extra layers of voxels (dilation), making the object thicker. Padding produces an additional smoothing effect of the object’s boundary, which makes it suitable for noisy objects, but can also merge object features that are very close to each other.
Currently, the entire force field is computed and stored in memory. Although the memory requirement is linear in the size of the original volume, for massive datasets, where the size of the vector field can easily reach gigabytes, out-of-core solutions need to be investigated.

### 7.8 Comparison of the two types of vector fields

The extracted curve-skeletons have many of the desirable properties described in Chapter 3. Below we present a discussion of the advantages and drawback of the two types of vector fields with respect to the desirable properties of curve-skeletons. For the measurable properties, we also give the numerical scores for the set of test objects used to illustrate the different techniques.

Note that these are the same test objects used in Chapter 5, so a direct comparison with the other implemented algorithms is possible. For brevity, in this section we will only provide the average scores over the entire test set for each type of vector field, instead of individual scores for each object. In order to make the comparison meaningful, we normalize those properties that are not directly given as a normalized index using the same maximum and minimum values recorded for those properties in Chapter 5. Also note that all the following evaluations were performed on the “core” skeleton alone.

#### 7.8.1 Centeredness

Centeredness is not guaranteed by the potential field repulsive force, since each interior voxel is influenced by all boundary voxels even if they are not visible from that point. If visibility of a boundary point from the interior voxel is taken into account, then the resulting skeleton is more centered. Since a reduced number of surface voxels influence each interior voxel in the case of the normal diffusion field, we expect the result to be more centered than in the case of the potential field function. However, as the surface area influencing each interior voxel is increased by performing smoothing of surface normals, centeredness is compromised. Therefore, the
centeredness score for the normal diffusion field is only slightly better than the one of the potential field.

<table>
<thead>
<tr>
<th>Table 13. Average centeredness for potential field and normal diffusion field.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Potential Field</td>
</tr>
<tr>
<td>Overall Performance</td>
</tr>
<tr>
<td>91.17%</td>
</tr>
</tbody>
</table>

7.8.2 Topology preservation

Topology preservation however is not guaranteed by any of these approaches and in fact can be compromised by inadequate resolution. We may have loops in the curve-skeletons that do not correspond to any tunnel or cavity in the original object.

<table>
<thead>
<tr>
<th>Table 14. Average topology preservation for potential field and normal diffusion field.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Potential Field</td>
</tr>
<tr>
<td>Overall Performance</td>
</tr>
<tr>
<td>71.78%</td>
</tr>
</tbody>
</table>

The slightly better score obtained by the potential field method can be attributed to better performance on a single object: the helicopter, where the normal diffusion field produces a disconnected skeleton.

7.8.3 Connectivity

Given the adequate resolution, connectivity of the core skeleton is guaranteed by the topology of the resulting vector field. It can be shown that between every pair of neighboring attracting nodes (where the force following stops), there must be a saddle point (because the force vector must change sign as we move from one attracting node to the other). These saddles are the seeds of our core skeleton and the skeleton segments seeded there will connect the attracting nodes.
Connectivity of the level 1 and level 2 skeletons is guaranteed because the force following 
algorithm stops only when reaching a previously visited location.

However, connectivity may be compromised if the voxelization was done at insufficient 
resolution. This causes the resulting vector field to be highly irregular in these areas (if 
insufficient smoothing is applied) and as a result, the force-following algorithm will get stuck. 
This is the case of the ET with noise object, when using divergence seeds. Nevertheless, 
connectivity can be achieved in a post-processing step by connecting disconnected parts of the 
skeleton to the closest part of the main skeleton. This connection can be done using a straight line 
or an active contour whose energy can then be minimized using the underlying vector field [43].

Table 15. Average connectivity for potential field and normal diffusion field.

<table>
<thead>
<tr>
<th></th>
<th>Potential Field</th>
<th>Normal Diffusion Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Performance</td>
<td>94.44%</td>
<td>83.33%</td>
</tr>
</tbody>
</table>

Note that, the better score of the potential field is due to better performance on only four of 
the objects.

7.8.4 Invariance under isometric transformations

Both types of vector field generating functions presented here are invariant under isometric 
transformation. The critical point detection procedure and the force-following algorithms are also 
invariant under such transformations. As a consequence, we expect the resulting curve-skeleton to 
also be invariant under isometric transformation.

Table 16. Average invariance under transformation for potential field and normal 
diffusion field.

<table>
<thead>
<tr>
<th></th>
<th>Potential Field</th>
<th>Normal Diffusion Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Performance</td>
<td>95.28%</td>
<td>97.35%</td>
</tr>
</tbody>
</table>
7.8.5 Robustness to noise

Both the potential field based repulsive force and the normal diffusion field incorporate some amount of averaging (more in the case of the repulsive function, less in the case of NDF) and this improves performance with respect to noise sensitivity. The core skeleton is less sensitive to noise because it is located closer to the center of the object, where the averaging effect is more pronounced. The higher levels of the hierarchy will be more affected by noise if the seeds are selected closer to the surface of the object, where the averaging effect is smaller. This can be controlled by varying the threshold used for selecting the additional seeds, or by dilating the object which also has a smoothing effect of the object’s boundary. Additionally, in the case of the potential field, a lower value of the field strength parameter \( m \) will increase the number of contributions from more distant charges on every interior voxel, thus increasing the averaging effect and reducing the negative effects of surface noise. In the case of the normal diffusion field, obviously increasing the number of smoothing steps in the surface normals will reduce the noise.

<table>
<thead>
<tr>
<th></th>
<th>Potential Field</th>
<th>Normal Diffusion Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Performance</td>
<td>92.28%</td>
<td>96.82%</td>
</tr>
</tbody>
</table>

Table 17. Average robustness to noise for potential field and normal diffusion field.

The score obtained using the normal diffusion field is higher because we usually used quite a few steps of smoothing on the surface normals, thus canceling the artificially added noise. In the case of the potential field, we kept the field strength constant between the object with and the object without noise. A decrease in this parameter could improve the overall performance of the potential field method.

7.8.6 Thinness

The force-following algorithm used to discover new skeleton segments guarantees that each curve-segment in the skeleton is thin. However, due to the limited resolution of the voxel grid, a
discretization of these curves may appear to me more than one voxel thick. Additionally, several curves close together can create the appearance of a surface in a discrete space.

Since the force-following algorithm results in curve segments given as a sequence of points, we consider these segments to be thin by definition. Therefore, the thinness scores are 100% for both methods.

**Table 18.** Average thinness for potential field and normal diffusion field.

<table>
<thead>
<tr>
<th></th>
<th>Potential Field</th>
<th>Normal Diffusion Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Performance</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

7.8.7 Reconstructability

The degree of reconstruction can be controlled by varying the number of additional seed points used in addition to the core skeleton. Clearly, a denser skeleton, using more additional seed points, will reconstruct more of the original object.

**Table 19.** Average reconstructability for potential field and normal diffusion field.

<table>
<thead>
<tr>
<th></th>
<th>Potential Field</th>
<th>Normal Diffusion Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Performance</td>
<td>62.41%</td>
<td>56.50%</td>
</tr>
</tbody>
</table>

The potential field method seems to be able to reconstruct more of the original objects because usually the resulting curve-skeleton segments seem to be slightly longer that those obtained using the normal diffusion field. Clearly this depends on the parameters settings.

7.8.8 Visibility

Visibility (seeing all the boundary points from the skeleton) is not specifically enforced by any of the two methodologies, but most of the object’s boundary is visible from the curve-skeleton.
Enforcing visibility is usually a post-processing step and it can be achieved using the methods described in [94].

**Table 20.** Average visibility for potential field and normal diffusion field.

<table>
<thead>
<tr>
<th></th>
<th>Potential Field</th>
<th>Normal Diffusion Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Performance</td>
<td>99.54%</td>
<td>98.95%</td>
</tr>
</tbody>
</table>

As in the case of reconstructability, the potential field method seems to perform slightly better because the resulting curve segments are usually longer.

### 7.8.9 Smoothness

The resulting curve-skeleton is usually smoother than those obtained using other methods (see Section Chapter 6). This property is given by the smoothness of the underlying vector field, which in turn is given by the averaging effect characteristic to both types of vector fields. Clearly the smoothness of the resulting curve-skeleton curves is also affected by the step size used by the force-following algorithm and the order of the integration scheme used there. For example, a simple Euler scheme will clearly generate less smooth curve-segments than a fourth-order Runge-Kutta scheme because the integration error is larger in the first case.

**Table 21.** Average smoothness for potential field and normal diffusion field.

<table>
<thead>
<tr>
<th></th>
<th>Potential Field</th>
<th>Normal Diffusion Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Performance</td>
<td>40.89%</td>
<td>22.24%</td>
</tr>
</tbody>
</table>

The scores obtained by both methods are low but this is something we’ve also seen when comparing the other implementations. As it can be observed from the images, the potential field produces much smoother curve-skeletons than the normal diffusion field.
7.8.10 Efficiency

The algorithmic complexity of the normal diffusion field is $O(n)$ where $n$ is the number of object voxels, while the algorithmic complexity of the potential field is $O(n^2)$.

7.8.11 Hierarchy

The overall skeletonization framework presented in this chapter is hierarchical, regardless of the particular type of vector field used to extract the curve-skeleton. As long as the vector field remains constant between computations of the different curve-skeletons, the resulting hierarchy is a strict one. This is a major innovation over previous approaches which are not hierarchical (at least not in a strict sense, see [107]).

7.8.12 Junction detection and component-wise differentiation

Junctions are detected automatically in this framework, independent of the particular type of vector field used. For the core-skeleton, the junctions are the critical points of the underlying vector field. When computing the curve-skeletons using additional seed points, junctions are detected where the force-following algorithm reaches an already visited location.

The advantage over other methods is that the junctions are detected before the curve-skeleton is actually constructed, in the case of the core skeleton.

7.8.13 Ability to handle different object representations

We have seen in Section 7.5.1.1 how the potential field can be computed from a set of unorganized point samples mapped to a discrete grid. The normal diffusion field method however cannot handle such point-based representations since it requires knowledge of the surface in order to initially compute the normals. Even though surface normals could be approximated from the neighboring point samples, the propagation step requires that the inside of the object is clearly distinguishable from the outside, as normals are propagated toward the interior of the object.
7.9 Concluding remarks

In this chapter, we introduced a novel curve-skeletonization framework, based on topological analysis of vector fields. We showed how topological elements of the vector field, namely critical points, can be used to extract a curve-skeleton from general 3D objects. We have described two types of vector fields which can be used to extract a curve-skeleton: the repulsive force of the generalized potential, and the normal diffusion field.

We also introduced the concept of hierarchical curve-skeletons, and we showed how the divergence of a vector field can be used to extract skeletons of various complexities for use in a large variety of applications. In addition, the hierarchy can be expanded by including other types of seed points, such as high-curvature points on the object’s boundary.

The curve-skeletons obtained using this framework have many of the desirable properties discussed in this work. In the last part of the chapter, we discussed the performance of the two types of vector fields with respect to each of the properties and we also presented quantitative evaluations of the resulting curve-skeleton with respect to the measurable properties.
PART II – Curve-skeleton Applications

In the second part of this thesis, we will present several applications of curve-skeletons. We start with volume decomposition, where the curve-skeleton can be used to drive the decomposition process, but also as a useful tool in conjunction with the decomposition (Chapter 8). Next, we present an application of curve-skeletons to object matching and retrieval (Chapter 9). We show how the curve-skeletons can be directly used to match 3D shapes from a large database. In addition, we investigate an extension of our curve-skeletonization framework to four-dimensions and we attempt to use the resulting skeletons for indexing and retrieval of time-varying functional MRI data (Chapter 10).
The purpose of object decomposition is to partition the original object into simpler and meaningful components. Studies like [18] indicate that we recognize objects by decomposing them into simpler components (called geons in [18]), recognizing the individual components, and finally, the entire shape. In an attempt to mimic the human perception, many computer algorithms use some form of object decomposition as a pre-processing step for various tasks [254], such as: 3D object recognition [96][223][250], animation, texture mapping [128], collision detection [134], object metamorphosis (morphing) [89][124][125], simplification [254], and compression [104].

These applications were developed mainly for polygonal meshes, and the problem is usually referred to as mesh decomposition or mesh segmentation. However, in the last few years, volumetric objects have become more common due to fast developments in scanning technology. Many of the tasks traditionally working with polygon meshes must now be applied to volumetric objects. As a result, the need for volume decomposition has been recognized by various researchers [48][124][227]. Finite element mesh generation for CAD models is the traditional application that requires volume decomposition to keep the meshing process manageable [143][144][211]. Volume animation [75] uses an IK-skeleton representing the meaningful components of the volumetric object. Focus and context rendering [163][216] requires separation of the volume of interest from the rest of the data.

Volume decomposition is related to 3D mesh decomposition, but there are some differences. Mesh decomposition and mesh segmentation refer to the same operation. However, volume
decomposition is not the same as volume segmentation. Both volume and mesh decomposition attempt to partition the original object (volume or mesh) into a number of disjoint components which cover the entire object (i.e., the union of the components equals the original object). A volume decomposition can be easily derived from a surface decomposition and the opposite is also true.

The first notable difference is that they operate on different object representations: volume decomposition operates on discrete volumetric objects, usually represented by voxels on a cubic grid, while mesh decomposition operates on polygonal representations in continuous space. The discrete nature of the volumetric representation introduces new problems, such as resolution. In the volumetric representation, the interior of the object is also explicitly given, as opposed to the surface-only representation achievable by polygon meshes. This introduces a new task in volume decomposition compared to mesh decomposition: defining the boundary between components in the interior. In this section we describe volume decomposition and discuss various applications using it. In Section 8.3, we show how curve-skeletons can be an effective tool for computing volume decomposition. Finally, in Section 8.4, we present four novel visualization applications of curve-skeleton based volume decomposition: texture mapping, volume compression for hardware accelerated rendering and manipulation, component-based selective transfer functions and generating visualizations for volume decomposition.

### 8.1 Related work

Closely related to volume decomposition, mesh decomposition has received a lot of attention because of its many applications and the proliferation of 3D models described as surface meshes. Mesh decomposition algorithms use curvature-based segmentation [153], simulation of electrical charge distribution [246], polygon clustering based on angular and geodesic distances [107], spherical mirroring and minimum cuts [106], curve-skeletons [39][134][136], or Reeb graphs
These algorithms only consider boundaries between components on the surface of the object. Also related is the decomposition of unorganized point clouds [78][85][247] or unstructured volumetric scalar fields [44]. These object representations, commonly acquired by 3D scanners, provide only sample points, without any connectivity information.

One of the first addressed issues was the decomposition of polyhedral objects into convex components since many operations on solid shapes can be significantly simplified if working with convex objects only [41][13][135]. The sub-surface boundaries between components are detected using cutting planes.

Volume decomposition is a problem of interest in the CAD field, where complex geometries are decomposed into simpler components in order to generate meshes for finite element analysis. The individual components are then matched with the most suitable automatic meshing algorithm to generate a mesh for the entire object. In [143][144], features (components) are detected using the CLoop technique (The CLoop is a closed set of linked edges with the same convexity). The Embedded Voronoi Graph, used in [211], is a graph encoding the Voronoi diagram where nodes contain an approximation of the geometry of the corresponding Voronoi diagram element (vertex, edge or face). This graph provides a natural decomposition of the volume into simpler parts. Although these algorithms operate on surface polygonal descriptions of 3D objects, they go beyond the surface and actually define complete component boundaries (i.e., boundaries extend below the surface).

Decomposition of discrete 3D objects represented by voxels on a cubic grid was also demonstrated using various techniques. In [227], decomposition based on object thickness uses region growing starting from a set of seeds identified in the distance field of the original object. In [61], a class of discrete active models called migration processes are used to define boundary components via constrained migration of curves and surfaces. In [48], an algorithm to extract curve-skeletons from 3D objects using a repulsive force field is presented. It is also shown how the curve-skeleton, in combination with a vector field can be used to decompose the volume. In
this chapter, we extend the work presented in [48] and we discuss using curve-skeletons for decomposition.

Also related, are the space decomposition algorithms where the entire space (inside the bounding box of the volume) is decomposed, rather than only the object, using octrees [29][119], KD-tress [177] or BSP trees [133]. These hierarchical data structures are then used to increase rendering efficiency by skipping empty space regions.

8.2 Volume decomposition

In this section, we give a formal definition of volume decomposition and we discuss the desirable properties of such a process from the perspective of various applications using it. Our discussion is limited to 3D volumetric objects, defined on a cubic grid.

Let $O$ be a volume object embedded in the discrete 3D space $Z^3 (O \subset Z^3)$. A set of sub-volumes $\{C_i\}, i=1,n$ (also called components) is a decomposition of volume $O$ if the $C_i$’s are disjoint volumes and their union is the original object $O$. More formally,

$$C_i \cap C_j = \phi, \forall i \neq j,$$

and

$$\bigcup C_i = O, i = 1,n$$

There are several desirable properties of such a decomposition: meaningful component separation, hierarchy, robustness to noise, invariant under transformation and additional boundary constraints. Some of these properties were also discussed in [227]. It is desirable for the decomposition to detect meaningful components of an object. A credible animation of a human body must use the well-known body parts. Unfortunately, there is no exact definition of meaningful components of an object in the general case. In other works, meaningful components have been described as components which can be perceptually distinguished from the remaining object [107][134], or components of different thickness [227]. Studies have shown that humans detect component boundaries in a complex object along negative minima of principal curvature.
(the minima rule [97]). However, reliable estimation of surface curvature especially in the presence of noise (very common in volumetric objects) is a difficult problem. Furthermore, the ideal decomposition of an object may differ among individuals based on experience with a certain class of objects. For example, a manufacturing engineer may see a decomposition of a CAD object into parts that are easy to manufacture using the machines he is familiar with, while a computer graphics person may see the same object as having a part structure appropriate for meaningful animation.

More importantly, a decomposition must be meaningful for the application that will ultimately use it. For example, for animation of a humanoid shape, the decomposition must provide separate components for each rigid part of the human body: fingers, palm, forearm, arm, torso, and so on. Depending on the pose and quality of the original object, these meaningful components may not be detected following the minima rule; nevertheless, they are essential for a good animation. Thus, meaningful decomposition of an object is application and user dependent. For certain types of objects, the minima rule provides a very good initial decomposition (see [106]) which then be adjusted by the user if necessary.

A hierarchical decomposition [106][107] provides the means to further partition certain components without having to re-compute the rest of the decomposition. This allows one to build a component tree, describing the hierarchical part structure of the object in a very intuitive way: at the top of the hierarchy we have the original object and the simplest components will be the leaves. Such hierarchical descriptions of 3D objects, in combination with graph matching algorithms, can be used for efficient 3D object recognition from a database [96], or for efficient collision detection [134]. Hierarchical decompositions are also useful if the number of required components changes on the fly.

Figure 58 shows an example of hierarchical decomposition. For the first decomposition level, a humanoid shape was decomposed into head, torso, hands and legs (Figure 58(a)). This may be good enough for a very crude animation for example, or for a coarse matching. The
second decomposition level, shown in Figure 58(b), further decomposes the hands into arm, forearm and palm with fingers, and the legs into thighs and legs. Note that the pre-existing boundaries from the first level were maintained in the second level. Also note that, in Figure 58, the boundaries of the different components were generated using knowledge of human body parts, not based on shape boundary characteristics such as curvature.

Two other properties are desirable for most computational processes: the decomposition should be robust to noise and invariant under isometric transformation.

Finally, different boundary constraints are applicable depending on the application. In contrast to mesh decomposition, boundaries between components need to be specified below the surface as well as on the surface. Various methods have been proposed for this: cutting plane [134], region growing starting from a set of seeds identified in the distance transform [227], or vector fields [48]. One requirement is that the boundaries between components be smooth [106][227]. Smooth boundaries could be very useful in a part matching application. Assume we have a database of object parts [213][237] and we would like to recognize the different parts of a complex object and retrieve them from the database. Most likely the parts in the database were created to have smooth boundaries so that they can be easily combined to create more complex objects. For other applications, a smooth boundary is not what is really desired. In component based texture mapping, a texture is applied to each object component independently. Component
based texture mapping is seen as a simplification of the texture mapping problem applied to complex objects: simpler components are easier to parameterize. The main disadvantage of this approach is that it will produce discontinuities at the boundaries between components. In order to reduce the effect of discontinuities, the boundaries between components should pass through concave regions, where the combined effect of lighting will make them less noticeable [128].

Various volume decomposition algorithms achieve the above properties differently. Decompositions for CAD objects achieve meaningful components for finite element mesh generation algorithms but they usually work only for objects with sharp features specific to CAD applications [143][144][211]. The hierarchy property is achievable by most algorithms by recursively applying them to the resulting components. The distance field based approach presented in [227] is different in that it first determines a very detailed decomposition and then a hierarchy can be achieved by merging the smaller components into larger ones. Boundary constraints such as smoothness can be achieved by active models using energy minimization [61], or in an explicit smoothing step [227]. The requirement that boundaries should pass through regions of deep concavities is also achievable by active models.

In the next section, we extend the work presented in [48], and we show how curve-skeletons can be used as an effective tool for volume decomposition.

8.3 Curve-skeletons and volume decomposition

Curve-skeletons have been used previously to drive the decomposition process of surface meshes [39][134][136], unorganized point clouds [85] or volumes [48] but they are not as popular as other decomposition methods. On the other hand, line-like curve-skeletons can be a by-product of the decomposition process [107][136]. In this section, we show how curve-skeletons can be used as very efficient tools for volume decomposition.
Several important curve-skeleton properties are useful for volume decomposition: reduced dimensionality (thin), junction detection and component-wise differentiation, robustness to noise, and hierarchy.

Junction detection and component-wise differentiation can be directly used to decompose the underlying volume based on the computed curve-skeleton. Some curve-skeletonization algorithms can directly detect junction points before extracting the curve-skeleton (general field methods [43][48]), while others (like thinning or distance field) can detect them in a post-processing step provided the curve-skeleton is one voxel thick. We believe the first approach gives more weight to the detected junctions as opposed to detecting them in a post-processing step. If junctions are detected before the skeleton is built, they must have some intrinsic relation to the original shape, while junctions detected in post-processing of the curve-skeleton are simply a by-product of it.

Hierarchical curve-skeletonization algorithms can generate curve-skeletons of different complexities by varying some parameter. A strict hierarchy means that skeleton segments (curves) computed at a certain level of the hierarchy are kept in subsequent levels. The only change that occurs in subsequent levels is the addition of new segments, creating more and more complex curve-skeletons. Finally, robustness to noise ensures that the curve-skeleton is not affected by small changes on the boundary of the volume.

### 8.3.1 Using curve-skeleton for volume decomposition

Curve-skeletons are a nice way to compute volume decomposition. Any curve-skeleton of a volume can determine a decomposition of the original volume, as long as the junctions are identified in the curve-skeleton. Let us consider a volume object \( O \) and its corresponding curve-skeleton \( Sk(O) \). Let \( \{S_i\}, i=1,n \) be the set of disjoint curve-segments forming the curve-skeleton. Each of these segments starts and ends with a junction or an end-point. This generates a decomposition of the skeleton into a set of segments. Next, we can transfer this decomposition to
the entire volume. In the simplest case, for each object voxel, we find the closest skeleton segment $S_i$ and assign that voxel to the $i$-th component, creating a one-to-one correspondence between curve-skeleton and volume components. We will refer to this technique as the minimum distance method.

Now let us remove the constraint that junctions should be identified a-priori. As long as the curve-skeleton is one voxel thick except at joints, we can easily identify junctions and end points by counting the number of neighbors of each curve-skeleton point, or by the algorithm proposed in [39]. As discussed above, these junctions are a by-product of the curve-skeleton and may not be as significant as the ones determined a-priori. Once the junctions and end points are identified, the decomposition can be performed as before.

A second argument for using curve-skeletons for volume decomposition is that it is easier to specify meaningful components using curve-skeletons because it is easier to add or remove a vertex into a curve than to identify all triangles in a mesh that should be coalesced. Curve-skeleton components are defined using a set of junctions (points on a curve). Because of the reduced dimensionality, they can be easily manipulated by the user to add or remove a junction. In Figure 59 we show such an example. After constructing the curve-skeleton of the cow, the algorithm automatically identified 6 junctions (marked with red spheres) and 9 end points (marked with blue spheres) (Figure 59(a)). Figure 59(b) shows the decomposition of the volume using the junctions and end points shown in (a). The decomposition was performed using the minimum distance approach described above. Assuming the decomposition will be used for animation, we would like to divide each one of the legs in two separate segments, add one more segment for the neck and have two segments for the main body. The user can easily create these seven new junctions (shown as green spheres in Figure 59(c)) with one click each; then the new skeleton segments can be computed automatically. The resulting decomposition using the same minimum distance approach is shown in Figure 59(d).
Some curve-skeletonization algorithms (such as general field methods) can identify a set of meaningful junctions directly during the skeletonization process. In this case, user intervention is limited to maybe adjusting the position of the junctions and possibly removing some, if the algorithm detected too many.

When using a curve-skeletonization algorithm which detects significant joint a-priori, the quality of the decomposition may not always be satisfactory, but this is true of any completely automatic method which does not allow user intervention. However, by using the curve-skeleton to drive the decomposition, we can take advantage of the robustness to noise property exhibited by these curve-skeletonization algorithms. This is a definite advantage over surface-based decomposition methods which may be very sensitive to small changes on the boundary.

Finally, the hierarchical property of curve-skeletons can be used to produce hierarchical decompositions. Figure 60(a)(b) and (c) show the first 3 levels of hierarchy for the curve-skeleton of a cow object and the corresponding volume decomposition respectively. Note that components that were not further decomposed in the second level did not change at all. The curve-skeleton
Figure 60  Hierarchical decomposition using the curve-skeleton.

was computed using our hierarchical curve-skeletonization framework, using the repulsive force of the potential field as the vector field function.

8.3.2  Computing component boundaries

One of the new problems introduced by volume decomposition when compared to surface mesh decomposition is defining the boundaries between components below the surface, where the minima rule [97] cannot be applied. Below we describe two possible approaches using the curve-skeleton.

The first and most simple approach is the minimum distance method. Each curve-skeleton segment is considered to be a set of points along the curve defining that segment. For each object voxel (on the surface and interior to the object), we determine the closest curve-skeleton point visible from this location. Let this point be $P_k$. To determine whether an object voxel is visible from a curve-skeleton point, we sample the straight line connecting them. If the line passes through a non-object voxel, they are not visible to each other, otherwise they are. Next, we determine the skeleton segment to which $P_k$ belongs and assign that voxel to the component with the same number as that skeleton segment. Visibility is an important ingredient since we want to
avoid assigning voxels to a component if the line of sight to that component is blocked by the object’s surface.

However, this simple approach suffers from a serious problem: some object voxels may not be visible from any curve-skeleton point. This is identified in Chapter 3 as the visibility property of a curve-skeleton. This problem is often caused by insufficient resolution, noise on the boundary, but it may also be caused by insufficient number of branches in the curve-skeleton. The end result is that some object voxels will not be assigned to any component. A possible solution to this problem is to assign these “orphan” voxels to the closest component.

Another problem with the minimum distance approach is that the closest visible skeleton segment may not represent the closest object component. This is schematically shown in Figure 61. Because the curve-skeleton does not extend all the way to the region where object voxel \( V \) is, the closest visible skeleton segment to voxel \( V \) is \( S_2 \). However, this will create the decomposition shown in Figure 61, where the component corresponding to segment \( S_2 \) is actually represented by two disconnected regions (shown in blue), separated by the component corresponding to the red segment \( S_1 \).

![Figure 61](Image)

**Figure 61** Shortest distance decomposition can create disconnected components.

An alternative solution to the minimum distance method was first described in [48], using the force-following algorithm presented in Section 7.3. For each object voxel, a force following process is started at its location and continued until a skeleton point is reached. Let this point be
Next, we identify the curve-skeleton segment to which $P_k$ belongs and assign the starting voxel to the corresponding component. Note that this is possible only if a vector field was used to extract the curve-skeleton. This can be a repulsive force field, the normal diffusion field (Section 7.5), the gradient of a distance field [33], or any other vector field defined over the object.

This method solves both problems which affect the minimum distance approach. First, the process guarantees that a curve-skeleton point is always reached during the force following procedure because the vector field is oriented towards critical points in the field which are always part of the curve-skeleton. Direct visibility from an object voxel to a curve-skeleton point is no longer an issue since the force-following procedure guarantees we cannot go outside the object. Additionally, the force following process will always reach the closest curve-skeleton segment and so problems like disconnected components will be avoided. In Figure 62 we show a comparison between minimum distance (a) and force-following decompositions (b) for the same objects. Note how the two methods produce different boundaries between components.

![Figure 62](image)

**Figure 62** Distance and vector field based decompositions.
8.4 Applications

In this section, we show how curve-skeleton based volume decomposition can be successfully used for several useful visualization tasks such as: volume texture mapping, volume compression for accelerated rendering and manipulation, component-based volume rendering and generating visualizations for volumetric decompositions.

8.4.1 Volume texture mapping

Texture mapping is an effective way for adding details to the surface of an object, and it has been used extensively in surface-based computer graphics. Surface mapping includes color texture mapping, bump mapping and displacement mapping. Bump mapping [24] uses a 2D texture to modify the normals of the pixels on the surface of an object, simulating wrinkled surfaces, while displacement mapping [46] alters the surface geometry, resulting in a more accurate rendering of surface details. Texture mapping has been extended to volumes by Miller and Jones [159], enabling the simulation of detail and special effects, such as fire and fur, in volumetric objects.

One of the difficulties of texture mapping, for both surface-based and volumetric objects, is the parameterization of the object into texture space. Simple parameterization techniques, such as spherical or cylindrical mapping, are easy to obtain, but they produce unsatisfactory results for complex objects. An example is shown in Figure 63(a), where spherical mapping is used to apply a displacement map onto a volumetric torus knot. The spherical parameterization maps the latitude and longitude of a 3D point into the X and Y coordinates of a 2D texture map. This 2D texture map contains a displacement that is added to the base surface voxels of the torus knot.

Curve-skeleton based volume decomposition provides a more natural way of defining this parameterization. For the torus-knot, for example, we can define a cylindrical mapping for each component, so the texture can be aligned with the object's axis of symmetry. This can be seen in Figure 63(b). Volume decomposition can provide a starting point for more sophisticated parameterizations, such as optimization-based approaches, as in [127] and [128].
8.4.2 Volume compression for hardware accelerated rendering and manipulation

Hardware-accelerated volume rendering is a subject that has received a lot of attention since 3D textures became popular in commodity graphics hardware. One of the drawbacks of texture-based volume rendering is the limited texture memory. Volumetric datasets need to fit into texture memory, which is usually too small in comparison with their increasing size. For instance, a $512^3$ volume requires 128MB of texture memory, and would require a total of 512MB if we wanted to store its pre-computed gradient. Only the latest generation of graphic boards can accommodate this amount of data [167]. For this reason, there have been some research efforts in texture compression, such as quantization-based [166] or multi-resolution approaches [205]. Data compression techniques, such as the S3TC standard, have been implemented in contemporary graphic boards [173], although with a considerable degradation in quality.

Recent efforts have also focused on the solution to this problem from the perspective of empty-space skipping. This technique tries to improve on volume rendering by organizing the texture in such a way that no empty space (typically air) is rendered and/or stored in texture memory, usually without any degradation in quality. Although these methods are usually not

Figure 63  Texture mapping. (a) spherical mapping, (b) component based cylindrical mapping.
accompanied by a data compression technique, the result is nonetheless a reduced memory size, which enables them to accommodate larger volumes in texture memory. These approaches are based on a hierarchical decomposition of the data, using texture hulls for 2D-texture-based volume rendering [132], octrees [29][119], KD-tress [177] or BSP trees [133].

None of these approaches use the structure of the volumetric object to drive compression or empty-space skipping. In this paper, we propose a new way of organizing a volumetric object, based on the volume decomposition. For each component of the volume, we compute the minimum bounding box. Unlike the approaches above, we are able to orient these boxes along the curve-skeleton segments (see Figure 64). For an efficient storage, each bounding box is then resampled along its curve-skeleton segment, separately. Finally, we use an optimal packing algorithm to pack these bounding boxes into the minimum possible 3D volume.

![Figure 64](image)

**Figure 64** Cuboids oriented along the skeleton segments.

To render the new volume, we load the packed volume into texture memory as a single texture and provide the bounding boxes as the proxy geometry to the rendering pipeline, following the cuboid rendering technique in [52] and [217]. According to this method, each cuboid is sliced independently along the view direction. Texture coordinates are defined for the
corners of the cuboid from the packing step, and tri-linearly interpolated automatically by the hardware.

One drawback of this method is that portions of the volume undergo a re-sampling process as they are “rotated” from the skeleton-aligned orientation to axis-aligned orientation, required for texture storage. If no such warping is performed, storage of the different components of the volume would not be optimal, as skeleton segments are not necessarily axis aligned. However, the effects of re-sampling may be acceptable for some applications, and are overcome by the benefits of texture compression and empty space skipping, which allow us to render larger volumes and obtain speed-ups in volume rendering.

Another reason for using this texture organization scheme is that the packed volume can be directly manipulated using hardware based manipulation approaches [216][217] at no extra cost. The curve-skeleton and the individual volume components are already computed from the packing phase. Manipulation of the volume is done by applying an affine transformation to the desired skeleton segments, which cause the corresponding cuboid to move or rotate accordingly. Since they are assigned a texture coordinate, the entire volumetric component is moved or rotated. Special handling at the joints between two skeleton segments is required, as described in [217].

8.4.3 Component-based volume rendering

In traditional volume rendering, transfer functions map color and opacity to individual voxels of the volume based on their value. By using the volume decomposition, we could assign different transfer functions to different components of the volume, allowing for a more targeted exploration of a particular volume. Volume decomposition assigns a component number to each voxel in addition to its value. We add another level of indirection in the transfer function lookup procedure, by first looking up the corresponding transfer function based on the component
number and then using that transfer function to look up the color and opacity corresponding to the voxel value.

This approach can be useful in focus+context scenarios [163][216], where, for example, the feature of interest is assigned a transfer function which presents it in full detail, but the surrounding features have different transfer functions in order to make them visible but not to distract the user from the feature being studied. Taking advantage of the reduced dimensionality of the curve-skeleton, the feature or features of interest can be easily specified by selecting individual components of the curve-skeleton. Transfer functions can then be assigned to individual features or a group of features.

To achieve interactivity, hardware-based rendering uses a set of skeleton-aligned cuboids as proxy geometry. These cuboids are assigned texture coordinates and they define the different object components which are assigned different transfer functions [216]. However, since the cuboids represent a poor approximation of each component, they can overlap when two components are close to each other. This results in portions of a different component being assigned the transfer function of the component of interest. This problem is illustrated in Figure 65(a). Here, the hand is very close to the body and the cuboid corresponding to the hand also includes parts of the torso. The resulting rendering shows parts of the torso being rendered with the same transfer function as the hand. Volume decomposition can be used to solve this problem. The same cuboid-based proxy geometry is used, but we also include the volume decomposition into the rendering pipeline. Although the cuboid corresponding to the hand includes parts of the torso, those parts are marked as belonging to a different component and will be ignored when rendering the hand cuboid (Figure 65(b)).
Visualizing volume decomposition

Throughout this section, we illustrated our volume decomposition work with a number of visualizations where the individual components of the volumes were artificially displaced to allow the user to perceive them as separate entities and show the boundaries between them.

A simple and completely automatic way to generate such visualizations uses the curve-skeleton based decomposition. We can use the curve-skeleton to build a component tree from the decomposed volume as follows:

- Choose a curve-skeleton junction point to be the root of the tree. Let this junction be \( J_r \). The root junction, \( J_r \), can be chosen by the user or by finding the closest junction to the center of mass of the volume.
- Starting from \( J_r \), find all curve-segments \( S_i \) starting from it and add the other end of each segment (let that be \( J_i \)) as a child junction of the root.
- Recursively apply the previous step to each of the children \( J_i \), taking precautions to ignore any loops the curve-skeleton may have.

This procedure generates a tree, where each node in the tree is a curve-skeleton junction or end point and the parent-child links represent the segments of the curve-skeleton. Figure 66(a)
shows an object with different components shown in different colors. A possible curve-skeleton of the object is shown in gray. The junctions in the curve-skeleton are marked with $J_0$, $J_1$, and $J_2$, and the end-points are marked with $J_3$, $J_4$, $J_5$, and $J_6$. The straight-line skeleton, obtained by connecting the junctions and the end-points of the curve-skeleton with straight lines, is shown in blue. Assuming $J_0$ is the root junction, the skeleton tree will have $J_0$ as the root, with $J_1$ and $J_2$ as children. In turn, $J_1$ will have $J_3$ and $J_4$ as children, while $J_2$ will have $J_5$ and $J_6$ as children.

![Figure 66](image)

**Figure 66** Visualizing volume decomposition. (a) different components of the object are shown in different colors; a possible curve-skeleton is shown in gray and a line skeleton, obtained by connecting junctions ($J_0$, $J_1$, and $J_2$) and end-points ($J_3$, $J_4$, $J_5$, and $J_6$) with straight lines, is shown in blue. (b) First transformation step: the children of the root junction ($J_0$) are translated along the directions shown as yellow arrows in (a). (c) Second transformation step: the children of $J_1$ and $J_2$ are translated along the directions shown as yellow arrows in (b).

The skeleton tree constructed above can then be used to determine the direction in which each volume component needs to be translated. For a junction $J_k$, each child junction $J_i$ corresponds to a skeleton segment with $J_k$ and $J_i$ as end points. In addition, each of these
segments corresponds to a volume component. Starting with the root junction $J_r$, we translate each volume component adjacent to $J_r$ along the straight line segment determined by $J_r$ and the corresponding child junction $J_i$. Let this translation matrix be $T_{ri}$. Figure 66(b) illustrates this step, with $J_0$ being the root junction and $J_1$ and $J_2$ its children. The directions in which the corresponding object components are moved are marked with yellow arrows in Figure 66(a). The above procedure is recursively applied to all child nodes of the root junction, while propagating the transformations down the skeleton tree. So at the second level, for every child $J_j$ of $J_i$, we compute the translation direction (let that be $T_{ij}$) given by the straight line segment between $J_i$ and $J_j$ and pre-multiply that transformation with the transformation already assigned to $J_i$ ($T_{ri}$). This step is illustrated in Figure 66(c): the directions in which the object components are moved are indicated with yellow arrows in Figure 66(b), and the result after the transformations is shown in Figure 66(c).
Chapter 9

3D Object Matching

The problem of searching for a specific shape in a large database of 3D models is an important area of research. Text descriptors associated with the 3D shapes can be used to drive the search process, as is done for 2D images [101][210]. However, text descriptions may not be available and furthermore, could not apply for part-matching or similarity-based matching. Matching 3D objects is a difficult problem, with a complex relation to the 2D shape-matching problem. While the 3D nature of the representation helps remove some of the viewpoint, lighting, and occlusion problems in computer vision, other issues arise. Of course, the added dimension and the inherent increase in data size make the matching process more computationally expensive. Furthermore, many of the models are degenerate, containing holes, intersecting polygons, overly thin regions, etc. And there are many different types of matching that may be desirable. Given a query object, one may want to search an entire database for a matching exemplar, if one exists. On the other hand, if the database contains categorical models, one may want to find the category to which the query exemplar belongs.

We will use the curve-skeleton of a 3D shape to drive the matching process. The skeleton provides the following characteristics not present in global shape descriptors:

- **Part/Component Matching**: curve-skeletons incorporate the notion of parts or components and so they can accommodate part matching, where the object to be matched is part of a larger object, or vice-versa. This feature can give the users more

---

1 The work presented in this chapter is the result of collaboration with Dr. Fatih Demirci and Professor Ali Shokoufandeh from Drexel University, and Professor Sven Dickinson from University of Toronto.
control over the matching algorithm, allowing them to specify what part of the object they would like to match or whether the matching algorithm should weight one part of the object more than another.

- **Registration and visualization**: The skeleton can be used to register the two matched objects and visualize the result in a common space. This is very important in scientific applications where one is interested in both finding a similar object and understanding the extent of the similarity [223].

- **Intuitiveness**: The skeleton is an intuitive representation of shape and can be easily understood by the user, providing more control in the matching process.

- **Articulated transformation invariance**: The method presented here can be used for articulated object matching, because the skeleton topology does not change as a result of articulated motion. An example was shown in [223].

The work presented in this section enhances the framework presented in [223] by using our new skeletonization algorithm and an extension of the many-to-many matching algorithm introduced in [110]. The idea of using skeletons for matching was described in [223]; however, only a small database was used for experimentation, and in general, the methodology was not scalable. In this work, the entire methodology has been revised by using our curve-skeletonization algorithm and a more robust matching algorithm. We demonstrate the efficacy of this new matching framework on the objects from the Princeton Shape Database [213]. Our matching algorithm is based on establishing correspondences among two skeletal representations via distribution-based matching in metric spaces. While the performance of our algorithm is comparable to that of other existing 3D matching methods (e.g., [174][213]), the locality of our skeletal representation and matching algorithm has some other benefits, such as allowing part matching, articulated matching, etc.
9.1 Previous work

A number of different approaches have been proposed for the matching problem. Using a simplified description of a 3D model, usually in lower dimensions (also known as a shape signature), reduces the 3D matching problem to comparing these different signatures. The dimensional reduction and the simple nature of these shape descriptors make them ideal for applications involving searching in large databases of 3D models. Osada et al. in [174] propose the use of a distribution, sampled from one of many shape functions, as the shape signature. Among the shape functions, the distance between two random points on the surface proved to be the most effective at retrieving similar shapes. In [213], a shape descriptor based on 2D views (images rendered from uniformly sampled positions on the viewing sphere), called the Light Field Descriptor, performed better than descriptors that use the 3D properties of the object. In [109], Kazhdan et al. propose a shape description based on a spherical harmonic representation.

Most of the previous work on point and skeleton matching has focused on solving one-to-one correspondence problems. Kim and Kak [111] used a combination of discrete relaxation and bipartite matching in model-based 3-D object recognition in computer vision. Pellilo et al. [179] devised a quadratic programming framework for the matching of 2D skeletons using a maximal clique formulation. Siddiqi et al. combined a bipartite matching framework with a spectral decomposition of graph structure to match shock graphs in [214].

The problem of many-to-many matching has also been studied, most often in the context of edit-distance (see, e.g. [206]). In such a setting, one seeks a minimal set of re-labelings, additions, deletions, merges, and splits of nodes and edges that transform one graph into another. Keselman et al. introduced a many-to-many matching algorithm and studied its utility for 2D skeletal representations in [110] using a specific measure, the Earth Mover's Distance, to compute distances between sets of weighted vectors. The matching algorithm used here is an extension of the one presented in [110].
9.2 Approach

Our 3D object matching framework uses the curve-skeleton as a reduced representation of a 3D object. Our goal is, given a query object (exemplar), to retrieve similar objects from a database. We assume that all objects already present in the database have been previously skeletonized using, for example, the hierarchical curve-skeletonization framework described in Chapter 7. The database, stored both the 3D object and its associated curve-skeleton. The matching framework employs the following steps, also illustrated in Figure 67:

- compute the curve-skeleton of the exemplar object and of every object in a database using the hierarchical curve-skeletonization framework described in Chapter 7.
- match the exemplar skeleton against all other skeletons in the database using the algorithm described in the next section.
- rank the results and visualize the best match.

---

**Figure 67** Curve-skeleton matching process.
The curve-skeleton of the exemplar and of all the objects in the database can be computed using any curve-skeletonization algorithms. In our experiments, we used the faster normal diffusion field. Additionally, each curve-skeleton point is equipped with the distance-transform value, which will be used by the many-to-many matching process. The distance-transform value specifies the shortest distance from the skeleton point to the surface of the object.

9.3 Many-to-many matching using EMD

To match two 3D skeletons, we use a distribution-based similarity measure, known as the Earth Mover's Distance (EMD) under transformation \[45][110]. The Earth Mover's Distance is designed to evaluate dissimilarity between two multi-dimensional distributions. The EMD approach assumes that a distance measure between single features, called the ground distance, is given. The EMD then “lifts” this distance from individual features to full distributions. The main advantage of using EMD lies in the fact that it subsumes many histogram distances and permits partial matches in a natural way. This important property allows the similarity measure to deal with uneven clusters and noisy datasets [110].

Computing the EMD is based on a solution to the well-known transportation problem [4], whose optimal value determines the minimum amount of “work” required to transform one distribution into the other. More formally, let \(P = \{(p_1, w_{p1}), \ldots, (p_m, w_{pm})\}\) be the first distribution with \(m\) points, and let \(Q = \{(q_1, w_{q1}), \ldots, (q_n, w_{qn})\}\) be the second distribution with \(n\) points. Let \(D = \{d_{ij}\}\) be the ground distance matrix, where \(d_{ij}\) is the ground distance between points \(p_i\) and \(q_j\). Our objective is to find a flow matrix \(F = \{f_{ij}\}\), with \(f_{ij}\) being the flow between points \(p_i\) and \(q_j\), that minimizes the overall cost

\[
\text{Work}(P, Q, F) = \sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} d_{ij}
\]

subject to the following list of constraints:
The optimal value of the objective function $Work(P, Q, F)$ defines the Earth Mover's Distance between the two distributions.

An extension of the original EMD approach [193] allows us to match point sets that are “non-rigidly” embedded into the Euclidean space by allowing sets to undergo transformations. Assuming that a transformation is applied to the second distribution, distances $d^T_{ij}$ are defined as $d^T_{ij} = d(p_i, T(q_j))$ and the objective function becomes $Work(P, Q, F) = \sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} d^T_{ij}$. The minimal value of the objective function $Work(P, Q, F, T)$ defines the Earth Mover's Distance between the two distributions that are allowed to undergo a transformation.

An iterative process called $FT$ (short for “an optimal Flow and an optimal Transformation”) that achieves a local minimum of the objective function was suggested in [193]. Starting with an initial transformation $T^{(0)}$ from a given $T^{(k)}$, they compute the optimal flow $F=F^{(k)}$ that minimizes the objective function $Work(P, T^{(0)}(Q), F)$, and from a given optimal flow $F^{(k)}$ they compute an optimal transformation $T=T^{(k+1)}$ that minimizes the objective function $Work(P, T(Q), F^{(k)})$. The iterative process stops when the improvement in the objective function value falls below a threshold. The resulting optimal pair $(F, T)$ depends on the initial transformation $T^{(0)}$.

We will use the Earth Mover's Distance under transformation between two curve-skeletons as a measure of their similarity: a small value of this distance indicating high similarity between the two skeletons. Figure 68 shows the distances between the query and the top matched objects.
Figure 68  The top ten best matches for a small set of objects. Models are sorted by similarity with the query object.

Note that an object from the database was not matched against itself. In the case of self-matching, the E.M.D. algorithm return zero, as no work is necessary to transform a skeleton into itself.

Figure 69 shows an example of matching between two objects: in step 1, the curve-skeleton for each object is computed while in step 2, the many-to-many matching establishes the distance and the correspondence between the two skeletal representations. The skeleton regions that were matched to each other are shown in the same color in Figure 69.

9.3.1   Visualizing the Similarity

An important aspect of the matching is the computation of correspondence between the matched objects. Our many-to-many matching algorithm provides a direct correspondence between the skeleton points of the two matched objects. This allows one to register the query object to the database objects and aid in visualization and a better understanding of the match. The correspondences between the matched objects are illustrated with color coded regions in
Figure 69  Computing similarity between two given objects. Step 1: compute skeletons. In Step 2 the correspondence is shown with color coded regions.

Figure 69 and Figure 71. Global shape descriptors perform poorly at this task because global information cannot preserve local correspondences.

9.4 Experiments

To evaluate the utility of our skeletal representation and many-to-many matching algorithm, we performed two sets of experiments: 3D base classification and part matching.

9.4.1 Base Classification and Object Retrieval

We first tested our proposed approach to retrieving similar objects on a subset of 1081 objects from the Princeton Shape Benchmark Database [210], grouped into 99 non-empty classes from both the test and training classifications [213] (the 3D base classification task). In our experiments, we first created 3D curve-skeletons for each object. We used a divergence threshold of 40% for all our skeletons. We then computed the distance from each object to the remaining database entries using our many-to-many matching algorithm. If the conceptual classes correspond to bodies which vary only in scale, or by articulated transformation, our algorithm should return an object that belongs to the same class as the query. We will classify this as a
“correct match”. Based on the overall matching statistics, we observe that in 71.1% of the experiments, the overall best match selected by our algorithm belonged to the same class as the query (also known as the nearest neighbor criterion [213]). In 74.3% of the experiments, the best match belonged to the same parent class as that of query.

In a second experiment, we asked how many of the models in the query's class appear within the top $T-1$ matches, where $T$ is the size of the query's class (first tier [213]). This number was 17.2%. Repeating the same experiment, but considering the top $2\times T - 1$ matches (second tier [213]) covers 22.7% of the members of the class.

Comparing these results with those reported by Shilane et.al. [213] in Table 4 of their work, it should be noted that our method outperforms all methods on the nearest neighbor criterion, but does not do as well on the first and second tier criteria. The sharp drop in the precision-recall curve in Figure 70 illustrates the loss of precision for the first and second tier criteria.

![Figure 70](image)

**Figure 70** Precision/Recall for many-to-many matching algorithm in object retrieval experiment.

The precision-recall plot shows the relation between recall (the ratio of models from the class of the query returned within the top $N$ matches) and the precision (the ratio of the top $N$
matches that belong to the query class) [213]. Figure 70 shows the precision-recall plot averaged over all models and looking at the first 20 best matches only.

In Figure 68 we have presented the matching results for a small subset of objects. The first column of each row shows the query object; the remaining elements of each row represent the top 10 closest objects of the database determined by our matching algorithm. These are all instances where the closest object is an object from a similar class. In some cases, while the algorithm has identified an object with similar structure as best match, it was still penalized for selecting an object from an incorrect category. The query object (race car) in row one and its best matched object are an example of such a case. They can be attributed to the particular hierarchy of categories used by the Princeton Shape Benchmark Database [210]. When similarity of shape is desired, a method which relies on shape would help retrieve objects not normally associated with the exemplar and not typically categorized with it.

9.4.2 Part Matching

Matching of a part within a complex whole is useful for CAD-type databases and also for recognition in laser-scanned images, which tend to cluster objects together. It is presumably also central to medical applications in which a particular configuration is to be found somewhere in a larger object. Specifically, given a part of an object as a query, one attempts to locate objects containing similar subparts. Here, the difficulty lies in the fact that none of the database objects contains an exact copy of the query.

Matching only parts of an object is not possible using global shape descriptors, such as the geodesic distance distribution [174]. The curve-skeleton, in conjunction with EMD can be used to perform part matching, taking advantage of the global and local shape characteristics preserved by the curve-skeleton and of the direct correspondence between skeleton points computed by EMD. As the result of EMD is not only a single number quantifying the similarity between two curve-skeletons but also a list of corresponding skeleton regions, we can aggregate these
individual correspondences to a higher level of abstraction and define correspondences between object components. As we’ve seen in the previous chapter, object components can be easily defined using the distinct segments of the curve-skeleton.

In our next experiment, we used a query part (a torso) and matched it against several simple and composite objects in the database, some containing the query part. The composite objects were obtained by a union operation applied to two simple objects - the kind of composition one would expect to encounter in laser-scanned scenes. Figure 71 shows the query object (and the corresponding skeleton) in (a) and some of the objects that was matched against in (b). In (c) we show the computed correspondence between the query skeleton and the skeletons of the objects in (b). Points that match the query skeleton are shown in red.

Figure 71 Part Matching Example: the query object in (a) is matched against each of the objects in (b). The correspondences between their skeletons are shown in red in (c).
Note that the EMD algorithm produces a correspondence list for every pair of query and database object, including those that do not contain the part. Deciding whether the corresponding regions in two curve-skeletons represent the same object component, (for example a torso of an alien figure), is a difficult problem, given that arbitrary transformations can be present. Furthermore, this requires a-priori knowledge about the semantics of the different parts. The advantage of our approach however, is that the matched parts can be easily identified and visualized, allowing the user to quickly decide whether the correspondence is meaningful or not.

Compared to the previous skeletonization method described in [223], the potential field-based skeletonization algorithm is more robust to reasonable amounts of noise on the boundary of the object. However, the curve-skeleton may not be the best abstraction for all types of objects. For example thin disk-like objects or a mushroom shape are not well represented by a curve-skeleton. The EMD-based matching compensates for a certain amount of differences in skeletonization by looking at loose relationships among the skeleton points.

9.5 Future work

The performance of the 3D shape matching framework could be further improved by improving the quality of the curve-skeletons used as index in the database. The ability to reconstruct the original object seems to be an important indicator of the degree to which the curve-skeleton actually represents the original object. Including the reconstruction criterion into the skeletonization algorithm could produce curve-skeletons that are more effective at the retrieval task. For example, our curve-skeletonization framework could be modified to include user-defined seeds selected from regions that could not be reconstructed from the current version of the curve-skeleton.

New ways of matching curve-skeletons could also be investigates. While EMD provides great flexibility, it does not take into account the natural relation between individual points on
curve segments. An algorithm specifically designed to match curve-segments or parts of segments could take advantage of these relationships, improving efficiency of the matching process and possibly the quality of the retrieval.
Chapter 10

Indexing and Retrieval of Dynamic Brain Images by Content

In the same context of shape matching as the previous chapter, in this chapter we present our work on indexing and retrieval of functional Magnetic Resonance Imaging (fMRI) data by content. We present the challenges posed by the data, and several approaches we took to solving the problem of retrieving this kind of data by content. A content-based retrieval of fMRI data could be used by clinicians for diagnostic purposes. For example, as a new patient receives an fMRI scan, the clinician could present it to such a system and the application would return other fMRI scans that are similar to the presented one. These already classified and diagnosed cases could then be used to help diagnose the current patient. Although such systems do not exist yet, in this chapter we show how we approached the problem and demonstrate that content-based retrieval of this kind of data is possible.

The initial results presented in this chapter follow the traditional way of processing fMRI data and are not based on the curve-skeleton. We present a matching framework based on computing and matching reduced representations of the fMRI data. The reduced representations are computed using the traditional processing of fMRI data using the General Linear Model.

---

2 The work presented in this chapter was done as part of the Novel Indexing and Retrieval of Dynamic Brain Images project, under the supervision of Professor Paul Kantor, at Rutgers University. Novel Indexing and Retrieval of Dynamic Brain Images (NSF Grant # EIA-0205178) is a joint project between Rutgers University New Brunswick, Rutgers University Newark and Princeton University, with Professor Paul Kantor (School of Communication, Information and Library Studies at Rutgers New Brunswick) and Professor Stephen J. Hanson (Department of Psychology at Rutgers Newark) as principal investigators. Additional members of the project include: Professor Deborah Silver (Rutgers), Professor Ali Shokoufandeh (Drexel), Professor Sven Dickinson (Toronto), Professor Jonathan Cohen (Princeton), Ulukbek Ibraev (Rutgers), Jeff Abrahamson (Drexel), Bing Bai (Rutgers) and John Novatnack (Drexel).
(GLM). We investigate several ways to compute similarity between these reduced representations (matching metrics) with increasingly better results. We begin with a short introduction to functional MRI and the analysis methodologies used to interpret these images.

10.1 Background on fMRI and fMRI analysis methods

Magnetic Resonance Imaging (MRI) has allowed scientists and physicians to study the structure of the living human body in a safe and non-invasive manner. A large proportion of the human body (about 70% of its weight) is composed of water, and different tissue types contain different amounts of water. Each of the hydrogen atoms in the water molecule (H₂O) is a tiny magnetic dipole; when placed in a very strong magnetic field (about 50,000 times stronger than the Earth’s magnetic field) each of these tiny dipoles will align themselves with the external magnetic field. A short pulse of energy perturbs these tiny magnets from their preferred orientation and, as they return to their initial orientation, they give off small amounts of energy which are detected and amplified by a receiver coil. As different tissue types contain different amounts of water, the intensity of the MRI signal varies from one region to another, providing a detailed image of the internal structure of the body (the brain in our case). The resolution of the MRI images is very good, with about 0.5mm per image element (pixel or voxel).

Although MRI images provide valuable information about the internal structure of the brain, they do not say anything about how the brain functions – they are static images. Functional MRI (fMRI) is a new technique aimed at mapping brain functionality.

10.1.1 Functional Magnetic Resonance Imaging (fMRI)

Functional Magnetic Resonance Imaging (fMRI) is a medical imaging technique that captures brain functionality by measuring the change in oxygen concentration in the blood that flows into different parts of the brain as the subject performs some mental task. The tasks performed by subjects are carefully designed to emphasize one or few functions or brain areas.
The fMRI methodology provides information about brain activity by measuring and recording the side effects of brain activation. Instead of directly measuring the brain activity, it reveals the local changes in blood oxygenation in the brain tissue, which is the direct result of brain activity. It is widely accepted that (at least some) brain functions are localized in different regions of the human brain. As brain areas become active during a specific task there is a local increase in the volume of fresh oxygenated blood that flows in that area to provide the harder working neurons with the energy they need. However, the increase in local oxygen concentration is not accompanied by an increase of the same magnitude in oxygen consumption by the underlying brain tissue. As a result, an overall increase in blood oxygenation can be observed in the activated area. The magnetic properties of the hydrogen atoms differ slightly between the areas near oxygenated blood and areas near de-oxygenated blood and the fMRI signal reflects this increase in oxygen levels by an increase in signal intensity, which is why the collected signal is also called BOLD (Blood Oxygenation Level Dependent) signal. Figure 72 shows schematically the causal relations between brain activation and the fMRI signal.

The fMRI machine consists of a specially modified MRI scanner (the magnet) that can produce functional images by detecting these small signal intensity variations due to changing oxygenation levels.

Currently, the spatial and temporal resolutions of the functional images are low, typical values for the spatial dimensions of a voxel are 3mm x 3mm x 5mm, and a complete functional image of the brain is taken every 2 to 5 seconds. While it is possible to reduce the time required for a scan by focusing on a narrower region of the brain, the scan time is clearly a few orders of magnitude bigger than the neuron firing time (activation) in a normal brain which is in the order of milliseconds. However, fMRI does not probe the brain activation directly, but one of its aftereffects. The increase in fresh blood flow in an activated area, also known as the
hemodynamic response, takes place gradually, reaching a maximum at 4 to 6 seconds after the start of the neural activity and completely disappearing only 25 to 30 seconds after the activation. This slow hemodynamic process is what makes the activation visible even on such low temporal resolution images. Although there is some debate in the scientific community regarding the exact shape of the hemodynamic response function, a widely adopted model is the double-gamma function model, used in the popular fMRI processing package FSL [62][218], shown in Figure 73:

\[ S_i = \frac{\hat{\lambda}_1 (\hat{\lambda}_1 t)^{(k_1-1)}}{(k_1-1)!} e^{-\hat{\lambda}_1 t} - c \frac{\hat{\lambda}_2 (\hat{\lambda}_2 t)^{(k_2-1)}}{(k_2-1)!} e^{-\hat{\lambda}_2 t}, \]

where \( S_i \) is the signal intensity, \( t \) is the time measured in seconds, with the following parameters: \( \hat{\lambda}_1 = \hat{\lambda}_2 = 1 \), \( k_1 = 6 \), \( k_2 = 16 \), and \( c = 1/6 \).

The subject lies inside the MRI machine and is given a task to perform. First, a high resolution static MRI image (a whole 3D volume) of the subject’s brain is taken; this is also known as the anatomical image and it is used for normalization. Then, the subject is asked to perform a series of tasks, during which the magnet takes 3D functional images of the patient’s head at regular intervals, usually once every 2 seconds. It is common for such a session to last for
a few minutes (five to ten minutes). Depending on the experiment, the task may involve a number of stimuli that the patient is supposed to react to. For example, one common task is known as the oddball task: the subject is presented with a set of pictures at regular time intervals and asked to react to the “oddball” picture by pressing a button. The “oddball” picture may be a face among a series of geometric figures. The time of appearance of the “oddballs” is recorded and provides the stimulus onset times. The onset times are used in most analysis packages to help process the data.

10.1.1.1 The data

An fMRI dataset contains a series of 3D images (volumes) of the subject’s brain, taken over the entire period of the experiment. The size of each volume is machine dependent but a typical value is 64x64x32. Anywhere from tenths to hundreds of volumes can be generated in a single session. For example, our “oddball” test described above contains 205 volumes taken over a period of 410 seconds, each having 64x64x16 voxels. Clearly, each experiment is different and will have a different number of volumes. Every volume is a scalar field representing the signal intensity at each voxel at the moment the image was taken. The series of intensity values corresponding to a single voxel position throughout the experiment forms the time series of that voxel. Thus an fMRI dataset can be thought of as a series of hundreds of volumes or a set of hundreds of thousands of individual voxel time series. Figure 74 shows a slice from one of the images in the “oddball”
dataset and the corresponding time series of three randomly selected voxels. The times when the “oddballs” were shown are represented with vertical lines on the graph.

10.1.2 fMRI analysis methods

Since fMRI measures and records the hemodynamic response to brain activation rather than the brain activity itself, extensive pre-processing and analysis has been employed to recover the original brain activation pattern from the acquired BOLD signal. fMRI analysis methods are usually grouped into two broad categories: hypothesis-driven and data-driven.

In hypothesis-driven methods, an a-priori model of activation is assumed and the data is checked to see how closely it corresponds to this model. The stimulus onset times are recorded and used for this kind of analysis. These onset times represent the initial points at which a change in the acquired signal is expected. In Figure 74, the vertical bars on the graph represent the stimulus onset times for the “oddball” experiment. In addition, a pre-defined hemodynamic response function is used to predict the effects of the brain activity. One example is the hemodynamic function showed in Figure 73. The convolution of the hemodynamic response function and the stimuli onset function provides the expected shape and duration of the BOLD signal for a voxel that shows activation in relation to the experiment that was performed. The next
step uses statistical methods to decide whether the time series of a particular voxel behaves as expected for a task-related activated voxel or not.

The most widely used package for analyzing fMRI sequences is Statistical Parametric Mapping (SPM) [220]. SPM uses hypothesis testing in the form of a generalized linear model (GLM) [72]. In this approach, a parametric statistical model of activation is assumed at each voxel location. The analysis is totally dependent on a hypothesized hemodynamic response function, which relates the dynamics of blood flow in brain capillaries (and the observed oxygen surplus or depletion) to the stimulus function built from the onset-times recorded during the experiment. The parameter values of a linear regression are estimated at each voxel location by analyzing the behavior of the voxel across the entire time sequence of the experiment. A statistical test (F- or t-test) is performed for each voxel to produce the measure of fit between the postulated activation model and the actual behavior of the voxel over time. Results are compiled into a probability map containing a value at each voxel. The 4D data is thus reduced to a 3D scalar field containing probabilities for each voxel. A high value indicates a high probability that the voxel’s behavior over time is linked with the activation model and the performed task. Thresholding the probability map at high values selects those voxels that most likely show task-related activation during the experiment. SPM99 [220] is a self contained and lock-step program that reads in the data and produces the probability map.

There are many other hypothesis-driven functional MRI analysis tools based on the same GLM approach. These include: AFNI (Analysis of Functional NeuroImages) [54], FSL (FMRIB Software Library) [62][218], VoxBo [236], RUMBA (Rutgers University Mind Brain Analysis) [194], etc.

The second category of methods is data-driven or exploratory methods [233] where no prior hypothesis about the activation pattern of a voxel is assumed. Using only the intensity signal over time, the data-driven methods attempt to classify voxels into distinct classes based solely on
signal characteristics. The data driven methods include Independent Component Analysis (ICA), Principal Component Analysis (PCA), Canonical Correlation (CCA) and clustering.

ICA, PCA and CCA attempt to recover independent mixed source signals from recordings at several locations (this is also called the blind separation problem). In the case of brain images, the independent sources could be different functional areas of the brain and the recordings are the intensity of the signal over time as measured at different locations (at every voxel). The methods attempt to identify a number of uncorrelated signals that could have produced the observed data. Additional constraints differentiate between these methods: ICA tries to maximize the statistical independence of the components; PCA imposes the orthogonality constraint on the source, while CCA tries to identify components that have maximum autocorrelation [71].

Clustering methods are extensively used for fMRI analysis both for post-processing the results of the hypothesis-driven analysis (where spatial clusters are found after thresholding the t-test from SPM processing for example) or as a data-driven exploratory method. In the case of exploratory methods, after identifying all the different clusters based on signal patterns, the correlation with the stimulus function (or the activation function) is used to rule out clusters that do not seem to capture task-related activity.

There are several possible bases for clustering. In one, each voxel is taken to be a point in the n-dimensional space where n is the number of scans in the fMRI sequence. A point in n-dimensional space is identified by its n coordinates taken to be the intensities of the BOLD signal at that voxel at each of the n scans in the experiment. The resulting clusters have a meaning in terms of brain functionality, since voxels behaving in a similar manner throughout the scan will map to neighboring points in this space. It was shown in [87] that clustering on the cross-correlation between stimulus and voxel behavior is more robust than clustering on raw fMRI time series. The output is the set of clusters such that all voxels in a set have similar correlations with the stimuli pattern. New clustering techniques are continually being developed, as are improvements to existing methods e.g. [58][86].
10.1.2.1 Normalization

The two data analysis categories do not differ in the extensive pre or post processing that is required in order to formulate a conclusion from such an experiment. Since brain size and shape varies considerably among different subjects it is imperative to have a standardized coordinate system to identify regions of the brain that are considered to be active during a given experiment. The Talairach brain was published by Talairach and Tournoux [229] and introduced three important innovations: (a) a coordinate system fixed to anatomical landmarks in the brain; (b) spatial transformations to map one brain to another; and (c) an atlas of the standard brain with anatomical labels [36].

The Montreal Neurological Institute (MNI) created the MNI template brain based on the average of several hundred MRI scans of normal brains. The International Consortium of Brain Mapping (ICBM) adopted the MNI template as an international standard. Locations in Talairach and MNI brains can also be mapped to the well-known (to neurologists) Brodmann areas, which are a classification of brain areas based on tissue structure.

Mapping the data to the standard space is a two-step process: first, the raw data has to be aligned with the subject’s high-resolution anatomical image (MRI) taken at the beginning of the experiment, a phase called co-registration; then, in the normalization phase, a transformation that maps the high-resolution image into the standardized space is computed. The transformations resulting from each of the two steps (\(T_{\text{MRI to anatomical}}\) and \(T_{\text{anatomical to standard}}\) are concatenated into a single transformation that transforms the fMRI data into the standardized space: \(T_{\text{MRI to standard}}\). Figure 75 illustrates this process.

This process can be done either as a pre-processing step or as a post-processing step; in the later case only the resulting activated regions are mapped to the normalized space.
Whatever the methodology, the ultimate goal of any type of fMRI analysis is to attempt to partition the brain itself into regions that are somehow related to the experiment and regions that are not. As discussed above, a number of different methods have been devised to accomplish this partition and comparing the efficacy of these methods is non-trivial, particularly because the “underlying truth” is typically unknown. Researchers thus face many tools to use, and no good criteria on which to make a selection. Recently, several tools have been developed for enabling interactive comparisons between the results of different analyses [47][66].

The vast amount of pre-processing, statistical processing, and post-processing, with a large number of parameters that need to be adjusted at every run, is likely to affect the final result of the analysis, making it hard or even impossible to reproduce. A recent study, the Functional Imaging Analysis Contest (FIAC), showed significant differences between analysis results on the same fMRI data using different software packages and thus different processing methodologies and parameters [186]. Therefore, an attempt at indexing and retrieving fMRI images by content should employ the same analysis on all the datasets in the database, or work directly with the raw (un-processed) data. Using the same analysis methodology and parameters for all datasets guarantees that the conclusions about areas of activation in different experiments are derived in a consistent way from all experimental data in the database and a direct comparison of these
conclusions does not have to include a detailed discussion of the effects of each parameter setting. On the other hand, directly using the raw data avoids making any assumptions at all about the data, basing any conclusions only on the observed data.

We have investigated both approaches. Initially, we used uniform GLM processing of all the datasets in our database and we used the results of the GLM processing and an index for retrieval. We will review those results first. Finally, we present our initial results on using curve-skeletons extracted directly from the raw data for retrieval.

10.2 Previous work

Previous efforts on retrieving fMRI images were usually based on matching textual information. Experiments collected in a database are accompanied by metadata describing the experiment in detail, with information such as: the name and description of the various conditions, characterization of the subjects participating in the experiment, such as age group, or health condition. Furthermore, once the experimental data is analyzed and activation regions are identified, these can be labeled using a standardized set of labels associated with the different parts of the brain. In addition, the scientists can add extra keywords that describe the experiment. Queries are then formulated as text search terms that will be matched against various fields of the metadata repository. Some well-known data repositories of fMRI experiments which support such queries are: the fMRI Data Center [99], the Brain Image Database [126], Neuro Generator [191], or the BrainMap database [68].

Content based classification of fMRI studies has been a growing area in the fMRI research community. In the data mining approach [241][114][202], large databases of fMRI images are mined to find relationships between brain lesions and functional deficits (e.g. vision or speech deficits) [126][157][156][158], or to classify patients based on fMRI activation patterns [63][64][65]. Parallel with the development of new data mining methodologies for fMRI data, a
series of visualization tools that can also be used to formulate queries have been proposed. With the tool proposed in [47], it is possible to use clustering and visual representation of activated regions in a query by example mode, which will retrieve similar datasets based on actual data similarity not on similar textual description. Also, in [66], Forsberg described a visualization and query tool which allows visual exploration and comparison of several analysis results and can be used to formulate queries in terms of both activated clusters and keywords associated with each experiment.

10.3 fMRI matching framework

Our approach for matching fMRI data is a query by example approach. Figure 76 shows an overview of our approach. We have built a database of several hundred distinct fMRI datasets which we processed uniformly. For processing, we can use standard GLM processing, resulting in an activation map, or some other processing methodology which produces a reduced representation of the original four-dimensional fMRI time series. These reduced representations, are stored in the database, together with the original fMRI data. We refer to the process of obtaining a reduced representation from the fMRI time series as the processing stage.

In a query-by-example approach, a new fMRI dataset is presented to the system, and the system will identify similar datasets that are stored in the database. The first step is to process the new dataset and compute its reduced representation in the same way it was done for the rest of the data in our database.

Next, we match the reduced representation of the new dataset against all the reduced representations in the database. This process is referred to as the matching step. The main component of the matching step is the algorithm that computes the similarity between two reduced representations: the matching metric. There are various definitions of similarity which can be used here, depending on the nature of the reduced representation. For example, if the
reduced representation is a set of labels describing the activation regions, we can use a text-based matching metric, which looks for the same textual labels in two distinct representations. Other application requirements can affect the choice of a particular matching metric, such as speed.

The result of the matching process is a list of similarity scores, representing the similarity between the reduced representation of the query and each of the reduced representations in the database. The list is sorted in decreasing order of similarity scores, such that the first occurrences in the list are more similar to the query than the ones occurring later.

In a real scenario, the results would be presented to the user as they appear in the ranked list, and the user would be allowed to inspect these results, and further investigating the similarity with the query. The visualization tool we developed as part of this project can be used for this task [47]. However, in the development stages of these methodologies, we need to evaluate the quality of the matching, the step referred to as the evaluation step. We evaluate the quality of the matching by using the Receiver Operating Characteristic (ROC). The ROC is a normalized graph of true positives (y-axis) versus false positives (x-axis) encountered as we walk down the ranked
list of retrieved datasets. Figure 77 shows an example ROC curve. Note that the x and y axes of
the graph are normalized to the actual number of false and true positives in the database. We
build the ROC curve as follows: as we go through the sorted list resulting from the matching step,
if the encountered dataset is a true positive (the same condition), we move one increment in the
vertical direction, otherwise, we move one increment horizontally.

![ROC Curve](image)

**Figure 77** ROC Curve.

The area under the ROC curve can be used to evaluate the quality of the retrieval result. An
area of 0.5 is equivalent to random guessing, while an area of 1 is equivalent to perfect retrieval
performance, where all the true positives are retrieved before any false positive is seen.

One ROC curve can be used to evaluate the quality of the retrieval based on one query
dataset being matched against the entire database. In our experiments, we actually used every one
of the datasets in the database as a query and matched it against the rest of the database. An ROC
curve was built for each query and the overall performance of the matching on the entire database
is described as the distribution of areas under the individual ROC curves, qualified by mean and
standard deviation.

Using this framework, we experimented with different reduced representations of the fMRI
data, as well as with different matching metrics, on a database of a few hundred fMRI datasets.
10.3.1 The database

We built a database of more than four hundred datasets from fMRI data collected from our collaborators at several laboratories. Our database includes datasets from five different experiments, each having several conditions and ran on different number of subjects, as shown in the table below:

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Conditions</th>
<th>Number of subjects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oddball</td>
<td>Visual</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Auditory</td>
<td></td>
</tr>
<tr>
<td>Event Perception</td>
<td>House</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>Study</td>
<td></td>
</tr>
<tr>
<td>Recall</td>
<td>Study Face</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Study Object</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Study Location</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Recall Face</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Recall Object</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Recall Location</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Try Face</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Try Object</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Try Location</td>
<td></td>
</tr>
<tr>
<td>Morality</td>
<td>M+E+</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>M+e-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M-e-</td>
<td></td>
</tr>
<tr>
<td>Romantic</td>
<td>Positive Face</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Neutral Face</td>
<td></td>
</tr>
</tbody>
</table>

In the visual oddball experiment, the subject is asked to look at a series of images and react to the “oddball” picture by pressing a button. The oddball can be a face among a series of geometric figures. In the visual oddball, the subject is asked to react to the “oddball” sound. In the event perception experiment [249] the subject is asked to identify events that occur during a short video sequence. The definition of event is left to the subject. For the house condition, the subject watches a cartoon representing the inside of a house with a ball bouncing around the objects in the room. For the study condition, the subject watches a video of a student setting up for study in a library. The oddball and the event perception experiments were provided by our collaborators.
from the RUMBA (Rutgers University Mind Brain Analysis) research group, in the Psychology Department at Rutgers University, Newark.

In the recall experiment [187][188], the subject is asked to study and then recall a number of faces, objects or famous locations. In the study condition, the subject looks at a face, an object or a place. In the recall condition, the subject is asked to remember the faces, or objects, or places that were previously displayed, while in the try condition, the subject is allowed to freely recall whatever he/she can remember that was previously presented. The morality experiment evaluates people’s responses to difficult moral dilemmas, with more or less emotional involvement. The M+E+ condition stands for a moral dilemma with high emotional involvement, while M-e- stands for an easier moral dilemma and no emotional involvement. The recall and morality experiments were provided by our collaborators at the Neuroscience of Cognitive Control Laboratory, at the Center for the Study of Brain, Mind, and Behavior (CSBMB), at Princeton University.

In the romantic experiment [6], the subjects were asked to look at a series of faces among which there was the face of the person they were in love with. The condition positive face signifies that the face of the loved person is showed, while the neutral face condition is one where the face of another person is shown. The romantic dataset was provided by our collaborators at the Functional Neuroanatomy and Basal Ganglia Research Lab at Albert Einstein College of Medicine.

For the following experiments, we selected a subset of 430 datasets from this database, in order to have a balanced number of datasets from each experiment.

10.4 Matching fMRI data using GLM activation maps

We initially approached the problem of matching fMRI datasets, using a reduced representation based on the traditional processing of this kind of data, using the General Linear Model (GLM). As different software packages using the same basic GLM methodology can produce quite
different results on the same data, it is important to analyze all datasets in the database in a uniform way, using the same software package and the same parameter settings. For the processing step, we used the software package FSL. FSL was chosen because of its scripting capabilities, making it easy to ensure that all datasets go through the same set of processing steps and with the same parameters.

The result of the GLM processing using FSL is a statistical map (t-map) corresponding to a condition, a subject and a particular run of an experiment. The t-value of a voxel indicates the degree to which a particular condition is responsible for the activation of that particular voxel. The highest values indicate voxels that are more likely to be activated due to the effect of the respective condition. Following the GLM approach, we threshold the resulting t-maps to obtain the activation regions corresponding to each condition. Choosing an appropriate threshold for the t values is not a trivial matter and a subject of debate in the scientific community, because the underlying truth about which regions of the brain should be active as a result of the experiment is usually unknown. Choosing a threshold that is too large might cause some weaker activated areas to be lost, while by choosing a small threshold too many regions will show up as active even though they are probably just noise. As there is no universally agreed upon optimal value for a t-map threshold, in our experiments, we selected the top 1% highest t-values from the t-map as being significant. The 1% value was chosen after several trial-and-error steps in which we measured the performance of our matching algorithms on the resulting t-maps. 1% seemed to be the best value for the set of datasets we had available. Figure 78 shows an example t-map and the remaining 1% of the highest values after thresholding.
The activation regions resulting after thresholding the t-maps are the reduced representations of the original fMRI datasets. We have a reduction in dimensionality from 4D to 3D, but also a reduction in the number of significant voxels after thresholding the t-maps. Using the above procedure, a reduced representation of an fMRI dataset based on the GLM analysis can thus be regarded as a set of active voxels, or locations (points) in discrete space $R = \{v_1, v_2, v_3, \ldots, v_n\}$. Furthermore, for each location, we have an associated weight, given by the actual t-value at the voxel: $R = \{(v_1, w_1), (v_2, w_2), (v_3, w_3), \ldots, (v_n, w_n)\}$. When we are interested only in the location of the activated regions, not necessarily in the actual t-values, the weights will be ignored.

Using these reduced representations, we investigated several algorithms for computing similarity between two such representations: simple overlap, fuzzy overlap, and EMD. Below we present the details of each of these matching metrics and the matching results on our database of 430 datasets.

Note that in order to directly compare two reduced representations, and depending on the employed similarity measure, they usually must be brought to the same space, by a normalization transformation, as described in Section 10.1.2.1 above. This step was performed on all the reduced representations before the matching step.
10.4.1 Simple overlap

Simple overlap is a very intuitive measure of similarity between two sets of voxels. We simply place the two sets on top of each other and we quantify the amount of overlap between them. Given two reduced representations $R_1$ and $R_2$, viewed as two sets of voxels, we define the overlap between them as the size of the intersection of the two sets, normalized by the size of the union of the two sets:

$$\text{SimpleOverlap}(R_1, R_2) = \frac{\|R_1 \cap R_2\|}{\|R_1 \cup R_2\|}$$

Note that in this case, the actual t-values at the voxels in the two representations are not used. The simple overlap metric is a normalized quantity, with a value of zero indicating that the two reduced representations have nothing in common, and a value of one indicating that the two representations are identical.

Using this metric, we performed the query-by-example test on our entire database of 430 datasets, using each one of the datasets as a query against the rest of the database. The distribution of areas under the ROC curves is shown in Figure 79, with a mean of 0.760 and a standard deviation of 0.134.

![Figure 79](image.png)

**Figure 79** ROC area histogram for simple overlap.
This result is quite promising, as it is quite different from a random guessing which would score a 0.5. It shows that we can indeed retrieve fMRI images by content using these reduced representations.

10.4.2 Fuzzy overlap

Simple overlap is an intuitive similarity metric, but it is very strict on the exact position of the active voxels in a reduced representation. However, the fact is that the normalization process which tries to align the data from each experiment with a template in a standard space is not perfect. The normalization process uses some kind of error minimization approach, and usually the optimal value found is not zero. As a result, the fMRI data cannot be perfectly aligned with the standard template. Usually, the errors are small enough to permit accurate labeling of the activated voxels with respect to well-known areas of the brain (such as the Brodmann regions), but such small errors can severely affect the outcome of the simple overlap similarity metric. Consider the simple case shown illustrated in Figure 80. If the two virtually identical activation regions from two different reduced representations are slightly offset because the datasets could not be perfectly aligned with the standard template, they may end up having no common voxels at all. As a result, their simple overlap similarity score is zero.

Figure 80 Normalization error affecting the outcome of simple overlap similarity.

The fuzzy overlap implementation and results were provided by my colleague Bing Bai.
To account for small offsets resulting from less than perfect normalization of the fMRI data, we investigated two different techniques: fuzzy overlap and EMD. In fuzzy overlap, we also consider as overlapped voxels, those voxels that are closer to each other than a threshold distance $r$, called the fuzzy radius. This is similar to actually dilating the activation regions in both reduced representation and then computing simple overlap, as seen in Figure 81.

**Figure 81** Fuzzy overlap of radius two. Two reduced representations and their dilation with radius two (a) and (b). Overlap of the dilated regions (c).

In the above figure, the solid blue and red regions in (a) and (b) are the original reduced representations. In (c), the dilation of each region is shown as empty squares, with red and blue outlines respectively, added to each of the shapes. When computing the fuzzy overlap similarity between the two representations (in blue and red), we also consider voxels that are at a distance $r$ or closer to each other as overlapping. This is shown in Figure 81(c) as red outlined squares with blue interior, or as squares with blue outline and red interior.

More formally, if $R = \{v_1, v_2, v_3, \ldots, v_n\}$ is a reduced representation, we build the fuzzy representation $R_{F(r)}$ of radius $r$ as:

$$R_F(r) = \{ p \in \mathbb{Z}^3 \mid \exists v_i \in R, \text{ such that distance}(p, v_i) \leq r \}$$

by including in the reduced representation, all voxels at a distance $r$ or closer to any of the voxels in the original representation. The distance between two voxels $p(p_x, p_y, p_z)$ and $v(v_x, v_y, v_z)$ can be
the Euclidean distance (L2-norm), the city block distance (L1-norm), or, as used in our experiments, the L-infinity norm: \( \text{distance}(p, v) = \max( |p_x - v_x|, |p_y - v_y|, |p_z - v_z| ) \).

Then, given two reduced representations, \( R \) and \( P \), the fuzzy overlap score is computed as the simple overlap similarity of their corresponding fuzzy representations \( R_f(r) \) and \( P_f(r) \):

\[
R \rightarrow R_f(r) \\
P \rightarrow P_f(r) \\
FuzzyOverlap(R, P, r) = SimpleOverlap(R_f(r), P_f(r))
\]

Note that, as in the case of simple overlap, the actual t-value of the voxels is not used.

Using the fuzzy overlap metric, we performed the query-by-example test on our entire database of 430 datasets, using each one of the datasets as a query against the rest of the database. We used different values for the fuzzy radius and we found that radius two gives the best results in terms of matching performance. The distribution of areas under the ROC curves for fuzzy overlap with radius two is shown in Figure 82.

![Figure 82](image)

Figure 82  ROC area histogram for fuzzy overlap, of radius two.

With a mean of 0.794 and a standard deviation of 0.131, the result is better than the one obtained using simple overlap.
10.4.3 E.M.D.\textsuperscript{4}

As each of the reduced representations can be regarded as a point distribution in space, we also applied the Earth Movers Distance (EMD) metric to evaluate the similarity between two such representations. Since the quality of the matching improves when going from simple overlap to fuzzy overlap, we believe this is because the overly strict overlap condition is relaxed in the fuzzy overlap case, reducing the effect of normalization errors. In this respect, EMD should be even more flexible than fuzzy overlap, since neighboring voxels are not connected to each other as in the fuzzy overlap case.

The EMD distance between two point distributions was described in Section 9.3. However, we need to incorporate some application specific conditions into the formulation given in Section 9.3. First, we would like to limit the maximum distance a mass can be moved since we only want to match voxels that are close to each other. The maximum distance (radius) a mass is allowed to move will be denoted by $r$. Additionally, we consider all the weights of the points in the distribution to the same (one), ignoring their actual t-values. These two conditions also simplify the formulation of the problem.

![Figure 83 ROC area histogram for EMD radius two.](image)

\textsuperscript{4} The E.M.D. implementation and results were provided by our collaborator John Novatnack, working under the direction of Professor Ali Shokoufandeh, in the Computer Science Department, at Drexel University, in Philadelphia.
Using various values for the radius $r$, we applied our matching algorithm to every dataset in our database. The best performance was obtained using a radius of two. Figure 83 shows the histogram of areas under the ROC curves obtained for matching of individual datasets.

The ROC areas distribution for EMD radius two has a mean of 0.808 and a standard deviation of 0.126, making it the best result in our experiments so far.

### 10.5 Curve-skeletons of fMRI images

All of the above matching metrics were used with reduced representations resulting from GLM processing using FSL. As noted previously, GLM processing incorporates a number of assumptions and a large number of parameters that need to be set at each run. Different settings which are considered reasonable can produce different results, hiding important features in the data, or reporting spurious brain activity. To reduce the effect of this source of errors, in the previous matching experiments, we processed all datasets in our database in a uniform way, using the same software package and the same parameters settings. However, different settings could be considered. For example, although we determined experimentally that choosing 1% of the highest t-values as the significant activation regions gave the best results, it can be argued that this particular value is best for our particular database, and is not scalable.

An alternative to this approach is to go back to the raw fMRI data and compute a reduced representation directly from it. As we have seen in Chapter 9, the curve-skeleton can successfully be used as a reduced representation to match 3D objects. Here, we investigate an extension of our vector field based curve-skeletonization framework presented in Chapter 7 to higher dimensions. Specifically, we would like to extract a four dimensional (4D) curve-skeleton from a raw fMRI (functional Magnetic Resonance Imaging) dataset and use it as a reduced representation for content-based indexing and retrieval of such datasets. Note that the curve-skeleton is still a one-dimensional abstraction, but in this case it will be embedded in a four-dimensional space.
fMRI data shows functional activity in the brain during a specific experiment. We would like our curve-skeleton to extract some of this functional activation rather than the shape of the brain which is quite similar in all humans. Thus, instead of using the shape of the brain, we will use the gradient of the intensity field to define the vector field which we can then use to extract a curve-skeleton with the algorithm given in Chapter 7. However, before using the raw fMRI data to extract the curve-skeleton, we must perform a few pre-processing steps, described in the following section.

10.5.1 Pre-processing

One of the defining characteristics of fMRI data is a low signal to noise ratio. As a result, before using the gradient of the intensity field, we must smooth (blur) the raw data in order to reduce the amount of noise that makes its way into the next phases of processing. In FSL, one of the most popular fMRI processing packages, the default smoothing is using a kernel of 5mm FWHM (Full Width at Half Maximum). Since FSL uses the GLM approach to compute the activation map, the noise is included in the model that tries to explain the observed data. As we will use a model-free approach, we must use a larger smoothing kernel. For the purpose of our experiment, we employed a smoothing kernel of 10mm FWHM. The right amount of smoothing depends on the size of the features we want to preserve. Since the FWHM is about two standard deviations, the total width of the kernel used in this case is about 30mm. Note that smoothing is performed in the spatial domain only (one volume at a time).

A second pre-processing step will remove the mean intensity from each voxel’s time series. This step is necessary because some structures in the brain constantly have a higher intensity than others. For example, the main arteries in the brain continuously carry fresh blood to the brain. Since the fMRI signal is an indication of the degree of blood oxygenation, the voxels corresponding to the main arteries will always have a higher intensity than other parts of the brain. By removing the mean intensity over time from each voxel we bring all voxels at the same
This operation makes small changes in intensity easier to observe. Note that the mean intensity is computed for each voxel as the arithmetic mean of the intensities measured for that voxel over the entire time span of the experiment. At this step, we want to also remove a linear trend from each voxel’s time series which is generated by the machine heating up during the experiment, producing artificially higher intensity readings as the experiment progresses.

Note that although we still have some pre-processing steps in our workflow pipeline, we have significantly fewer parameters that need to be set, compared to a full GLM processing, which can involve tenths of parameters. All of these pre-processing steps, excluding mean intensity removal can be performed using the popular fMRI processing package, FSL.

10.5.2 Significance of the core-skeleton of fMRI data

The core-skeleton extracted from the gradient field of an intensity map traces the ridges and valleys of a terrain map (height map) defined by the intensity field (by taking the intensity to represent the height of the terrain). Let $I^{(n)}$ be an intensity field defined in an $n$-dimensional space, with $I^{(n)}(P)$ representing the intensity at point $P$. This can be visualized as an $(n+1)$-dimensional terrain map $I^{(n+1)}$, by mapping the intensity at each point $P$ to the height of the terrain at that point. This is a one-to-one mapping of a point $P$ from $n$-dimensional space to a point $P' = (P, I^{(n)}(P))$ in $(n+1)$-dimensional space. We will compute an $n$-dimensional curve-skeleton using the gradient of the intensity field $\nabla I^{(n)}$ as the vector field. The core-skeleton is composed of the critical points of the vector field and the streamlines connecting them. We have three general types of critical points: attracting nodes or foci, repelling nodes or foci, and saddles. An attracting node in the gradient field is a local maximum in the intensity field, mapping to a peak in the $(n+1)$-dimensional terrain map. A repelling node represents a pit in the terrain map, and saddles are points that are local maxima in some directions and local minima in other directions. The principal directions where this behavior occurs at a saddle are given by the eigenvectors of the Jacobian of the gradient field at the critical point (equivalent to the Hessian of the intensity field
The streamlines connecting these critical points trace the ridges and valleys (course lines) in the \((n+1)\)-dimensional terrain map. Ridges and valleys are defined in 2D as the loci of minimum gradient magnitude along the relief’s level curves. In [59][11], \((d)\)-dimensional ridges and valleys in an \((n)\)-dimensional height map, \(I\), are defined as the points \(P\) where, if \(\lambda_1 \geq \cdots \geq \lambda_n\) are the eigenvalues of the Hessian matrix at \(P\) and \(v_1, \ldots, v_n\) are the corresponding vectors, the following condition (the height condition) is satisfied:

\[
\nabla I \cdot v_i = 0, \quad \forall i \in \{1, 2, \ldots, (n - d)\}
\]

Additionally, the actual values of the eigenvalues can be used to differentiate between ridges and valleys.

\[
\begin{cases}
\text{ridge if } \lambda_i < 0, \quad \forall i \in \{1, 2, \ldots, (n - d)\} \\
\text{valley if } \lambda_i > 0, \quad \forall i \in \{1, 2, \ldots, (n - d)\}
\end{cases}
\]

In the case of the fMRI images, we view the four-dimensional intensity image acquired from the magnet as a five-dimensional terrain (height) map, using the mapping defined above. We are interested in a subset of the core-skeleton, composed of the peaks of the five-dimensional terrain map and the one-dimensional ridges connecting them. The peaks have a special significance in the fMRI context, representing local spatial and temporal maxima of activations. The one-dimensional ridges connecting the peaks represent the shortest path between these peaks on the 5D terrain map. Since points on a ridge have the minimum gradient magnitude among their neighbors at the same elevation, the ridge also represents the easiest path (in terms of work done by the gradient field) between the two peaks.

While the physiological significance of the peaks in the 5D terrain map as local maxima of brain activity is clear, it is difficult to associate a physiological significance to the one-dimensional ridges connecting them. Under the assumption that the brain uses several distinct areas in a certain sequence to perform a task, and that the sequence can be determined by following the evolution of the activation peaks in time, a possible interpretation of the one-
dimensional ridges in the 5D terrain map is that they represent the most likely path the activation evolves through the brain in time, since they represent the minimum effort path between the peaks. Note that the assumption that the brain uses different areas in a sequence while performing a task is still a debate subject in the scientific community and there are currently no experiments that prove or disprove this theory. As a result, the interpretation given here should be regarded only as an attempt to provide an intuitive explanation the one-dimensional structures extracted from the four-dimensional data.

To compute the curve-skeleton of an fMRI time series, we extend our curve-skeletonization framework to four dimensions. The extension is straightforward: the computed vector field will now be four-dimensional, the critical points can be identified and characterized similarly to the 3D case, while the force-following algorithm remains practically unchanged. The way the vector field is computed is similar to the repulsive force of the potential field in 3D. In 3D, the repulsive force is the gradient of the potential field (a scalar field) generated by charges placed in the surface of the 3D object. In the case of fMRI data, we already have the scalar field given as the intensity measured at each voxel during the experiment, and we directly take the gradient of this scalar field.

To extract the subset of the core-skeleton of interest for the fMRI images, we start our force-following algorithm at saddles which have three negative and one positive eigenvalue. According to the definition of $d$-dimensional ridges given above, such a saddle point will be on a one-dimensional ridge. From these saddles, we follow the outgoing flow direction (direction where the gradient increases in magnitude) until we reach a peak. Since the saddle is already on a one-dimensional ridge, following the gradient direction guarantees that we stay on the ridge since we are always moving in the direction where the elevation increases the most. Another consequence is that when starting at such a saddle point, we can only reach a peak. A proof can be sketched as follows: Let $Ps$ be a saddle point with three negative eigenvalues and a positive one. This places $Ps$ on a one-dimensional ridge, making it a local maximum in the three directions given by the
eigenvectors corresponding to the negative eigenvalues, and a local minimum in the direction given by the fourth eigenvector. Furthermore, the three eigenvectors are perpendicular to the ridge and the fourth eigenvector is parallel to the direction of the ridge at $P_s$. Following this direction and continuing on the path of maximum ascent, we remain on a one-dimensional ridge, where each point is a maximum in three directions perpendicular to the ridge. While tracing the ridge, the fourth eigenvalue (initially positive) can only become negative or zero (the other three remain negative everywhere on the ridge). When the fourth eigenvalue becomes positive, we reached a peak; when the fourth eigenvalue becomes zero, we reached a part of the ridge which is horizontal. In the later case, if the fourth eigenvector is not zero, we can use it to determine the direction in which the next step will be taken; and if the fourth eigenvector is zero, we will use the normal of the hyperplane determined by the other three eigenvectors as the next direction.

The resulting curve-skeleton segments can span multiple time steps of the original experiment time frame. Let $s$ be a curve-skeleton segment, consisting of $k$ points sampled by the force-following algorithm $s = \{s_0, s_1, \ldots, s_{(k-1)}\}$ starting at the saddle point $s_0$ and ending at the attracting node $s_{(k-1)}$. The sequence of points should have monotonically increasing or decreasing temporal coordinates in order to make sense in our interpretation that a curve-skeleton segment represents a migration of activation from one region to another in the brain. This migration can only happen as the experiment progresses (forward in time). However, since the curve-skeleton extraction algorithm starts at saddles and converges toward attracting nodes, effectively in the middle of a migration path, the two resulting curve-skeleton segments originating at a saddle will have increasing and decreasing temporal coordinates for their points respectively. Therefore, both monotonically increasing and decreasing curve-skeleton segments (with respect to their temporal coordinates) are valid. More formally, let $s_i^{(0)}$ be the temporal coordinate of the point $s_i$. We impose the additional constraint that:

\[
\begin{cases} 
  s_0 \leq s_1 \leq s_2 \leq \ldots \leq s_{(k-1)} , \text{ or } \\
  s_0 \geq s_1 \geq s_2 \geq \ldots \geq s_{(k-1)} 
\end{cases}
\]
Being a four-dimensional object, the curve-skeleton of an fMRI time series cannot be directly visualized. For visualization purposes, we project the curve-skeleton to three-dimensional space by “injecting” the time coordinate into one of the spatial coordinates. This will allow us to visualize the curve-skeleton directly and map the temporal component to shifts in one of the spatial coordinates. An alternative is to use color to encode time, but as the number of curve-skeleton segments increases, the view quickly becomes cluttered and unreadable. Figure 84 shows a visualization of the curve-skeleton of a real dataset (Subject 10, of the recall experiment, run 3). To avoid excessive clutter, only the first five seconds of the curve-skeleton are shown. The time coordinate was “injected” into the z coordinate (the up direction) of every curve-skeleton point using the following formula: \( \text{new}_z = z + \text{time} \times \text{sizeofZ} \), where \( \text{new}_z \) is the new z coordinate of the point, \( z \) is the original coordinate, \( \text{time} \) is the time component and \( \text{sizeofZ} \) is the length of the Z edge of the 3D brain volume. Using this formula, a curve-skeleton point at time = 2 seconds will be shown just above the bounding box of the 3D brain volume.

Figure 84  Visualization of a curve-skeleton from a real fMRI dataset.
10.5.3 Matching curve-skeletons of fMRI data

In this section we present the initial results we obtained using the curve-skeleton of the fMRI data as a reduced representation for content-based retrieval. Matching these reduced representations poses some challenges. First, the resulting curve-skeletons are four-dimensional and different experiments will have different lengths. Therefore, a direct matching in four dimensions will have to somehow accommodate skeletons of different temporal length. This can be achieved by scaling both curve-skeletons to a pre-defined temporal length. Secondly, the size of these curve-skeletons is quite large, in the order of millions of points and hundreds of thousands of segments, depending of course on the length of the experiment. The Earth Movers Distance algorithm we previously used for matching 3D skeletons, being an optimization based algorithm, cannot easily scale to such large representations.

Therefore, for our initial test, we used a simplified approach: we project the four-dimensional curve-skeleton to three-dimensional space by simply ignoring the time component of the curve-skeleton points. As many curve-skeleton points will project to the same location, for each voxel we use a counter to measure how many curve-skeleton points project to that location. This produces a skeleton map similar to a t-map used previously, and with similar significance. The voxels with high counter value represent locations in the brain where the curve-skeleton frequently reaches at different points in time. We interpret these locations as important to the performed task, as the curve-skeleton tracks the movement of the activation regions. Similarly to the approach taken when using t-maps, we threshold the skeleton map using 1% as the threshold for significant values. From this point on, we can directly use any of the matching metrics we previously used for matching GLM-based reduced representations.

10.5.3.1 Retrieval experiment

In the following experiment, we used the same database of fMRI data described in Section 10.3.1. Before extracting curve-skeleton from raw fMRI time series, we need to separate the different
conditions present in the same run of an experiment. During a particular experiment, the subject may be presented different stimuli corresponding to different conditions during a single run of the fMRI machine. In practice this means that multiple conditions will be present in a single fMRI time series. While GLM processing is able to separate them because the sequence of stimuli is an input to the program, our approach is completely model free and has no knowledge about the stimulus that was applied. Therefore, we need to manually separate the different conditions present in the same run into different fMRI time series, so that the resulting curve-skeleton of an fMRI time series can be thought of as being representative for a single condition.

To separate the different conditions in a single fMRI time series (run), we used the sequence of different stimuli present in that run. For each stimulus kind $k$, we build a new fMRI time series which includes all the volumes in the original time series from the point in time where stimulus $k$ was applied, up to the point where a different stimulus kind was applied, or up to the end of the original time series. We call these new fMRI time series condition separated runs.

The curve-skeleton is computed from each of the condition separated runs and from the curve-skeleton, we compute the further reduced representation which is the skeleton map (described above) and we threshold it by selecting the top 1% voxels as significant.

**Figure 85** ROC area histogram for matching skeleton maps thresholded at 1%. 
In our initial experiment, we used simple overlap as the similarity metric between two
thresholded skeleton maps. Figure 85 shows the distribution of ROC areas for the full database.

With a mean area under the ROC curve of 0.618 and a standard deviation of 0.162, the result
is not as good as the other methods. It is still better than random guessing, which shows that the
skeleton maps have some discriminating power. As opposed to the t-maps, obtained from GLM
processing, the skeleton maps do not necessarily cover the entire brain. As a result, the 1%
threshold selected for t-maps may not be the best choice for skeleton maps also. Indeed, we found
that the retrieval performance improves slightly as we increase the threshold, and reaches a
maximum at 7%, with a mean area under the ROC curves of 0.633 and a standard deviation of
0.205. Figure 86 shows a graph of the mean area under the ROC curves as a function of the
threshold used to select significant voxels from the skeleton maps. The similarity metric used in
all cases was fuzzy overlap, radius two.

![Mean ROC area vs. Threshold](image)

**Figure 86** Retrieval performance, measured as mean area under the ROC curves, versus the
threshold used to select significant voxels from the skeleton maps.

The table below shows the retrieval results obtained using the different methodologies
discussed in this chapter.
Table 23. Retrieval performance of the different methodologies presented in this chapter.

<table>
<thead>
<tr>
<th>Methodology name</th>
<th>Mean area under ROC curve</th>
<th>Standard deviation of area under ROC curve</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMD, radius 2(^5)</td>
<td>0.808</td>
<td>0.126</td>
</tr>
<tr>
<td>Fuzzy overlap, radius 2(^6)</td>
<td>0.794</td>
<td>0.131</td>
</tr>
<tr>
<td>Simple overlap</td>
<td>0.760</td>
<td>0.134</td>
</tr>
<tr>
<td>4D Skeleton Maps, threshold 7%, fuzzy overlap, radius 2</td>
<td>0.633</td>
<td>0.205</td>
</tr>
<tr>
<td>4D Skeleton Maps, threshold 1%, fuzzy overlap, radius 2</td>
<td>0.618</td>
<td>0.162</td>
</tr>
</tbody>
</table>

10.6 Concluding remarks and future work

The methodology presented in chapter, based on 4D curve-skeletons of raw fMRI data, has the advantage that it does not make any assumptions about the underlying data or physiological processes in the brain. It is a non-hypothesis, exploratory approach. A reduced representation is extracted directly from the raw data after a few pre-processing steps, involving only an handful of parameters, as opposed to the numerous parameters and assumptions used in the full GLM pipeline. Although the final retrieval performance is not as good as of those methods based on hypothesis-driven GLM processing, it performs better than random guessing, showing that there is some discriminating power in the skeleton maps used as reduced representations in this experiment. As mentioned above, the strength of non-hypothesis methodologies is that new and interesting characteristics of the data can be discovered even if they don’t fit a pre-defined model.

Further investigation is necessary past the preliminary results presented in this chapter. One direction could be to improve the performance of the algorithm under the extremely noisy conditions of fMRI data. Another direction to investigate is the development of a true four-

---

\(^5\) The E.M.D. implementation and results were provided by our collaborator John Novatnack, working under the direction of Professor Ali Shokoufandeh, in the Computer Science Department, at Drexel University, in Philadelphia.

\(^6\) The fuzzy overlap implementation and results were provided by my colleague Bing Bai.
dimensional matching algorithm for curve-skeletons, one that does not ignore the time component encoded in the curve-skeleton.
Chapter 11

Conclusion

In this work, we presented a comprehensive framework for curve-skeletonization with the following components:

- A thorough review of numerous applications of curve-skeletons in various scientific domains,
- A framework for evaluation of curve-skeletons based on a number of general criteria complete with evaluation algorithms,
- A comprehensive evaluation of various curve-skeletonization methodologies, with implementation of basic algorithms from each class and comparison of a common set of test objects.
- A novel hierarchical curve-skeletonization extraction framework based on vector fields, which extracts a curve-skeleton set, and
- Several new applications of curve-skeletons:
  - 3D object matching,
  - volume decomposition, and
  - content-based retrieval of four dimensional fMRI data.

In our attempt to bring more formalism to the concept of curve-skeleton, we looked at the numerous applications, in various scientific domains, using some form of curve-skeleton and we distilled a list of desirable properties of this one-dimensional abstraction, properties required or useful in many of the surveyed applications. Our in-depth analysis suggested that a rigorous definition may not be desirable, as it may not achieve all these properties. Starting from these
properties, we developed an evaluation framework, with thirteen criteria, for the curve-skeleton and the curve-skeletonization process. We propose rigorous definitions of these criteria and develop methodologies and algorithms for evaluating a curve-skeleton with respect to each of the nine associated properties. These algorithms would allow one to objectively assess the quality of a curve-skeleton with respect to a set of requirements relevant to a specific application, and provide a standard for evaluating new curve-skeletonization extraction algorithms. Following the initial publication of the thirteen evaluation criteria, recently, researchers have begun to propose more rigorous definitions for the curve-skeleton and evaluate the new algorithms with respect to the desirable properties (see [56]). In this respect, this is one of the first attempts at a systematic treatment of the 3D curve-skeleton.

An important contribution of this work is a comprehensive review of the vast literature on the subject of curve-skeletonization, and the proposal of a simple taxonomy for all the different methodologies. The proposed taxonomy looks at the underlying implementation used in the various algorithms and classifies them into four classes: thinning and boundary propagation, distance-field based, geometric and general field methods. We discussed the advantages and drawbacks of each of these classes of algorithms in the context of the proposed evaluation framework. In addition, we actually implemented a “core” algorithm from each class and we evaluated their performance side by side on a common set of test objects. This “practical” approach has two merits: it augments our qualitative discussion of the strengths and weaknesses of the various algorithm classes with a performance test of actual implementations, and it also demonstrates how our evaluation framework can be used as a benchmark for evaluating the curve-skeletons obtained using different implementations. All the implementations and test objects are publicly available for download on our web site [242].

Another major contribution is the definition of the curve-skeleton set and the development of a novel hierarchical curve-skeletonization framework based on topological analysis of vector fields. The proposed framework can produce one-dimensional curve-skeletons of various
complexities for use in a wide range of applications. We presented two algorithms for computing the vector field, which lies at the foundation of our hierarchical framework: a repulsive force field based on the generalized potential model, and a normal diffusion field, constructed by propagating surface normal toward the interior of the object. We demonstrated the two approaches on a diverse set of three-dimensional objects and we evaluated the results with respect to the desirable properties of the curve-skeleton.

Finally, we presented two novel applications of the curve-skeletonization extraction framework: volume decomposition and object retrieval. We discussed volume decomposition, and showed how the curve-skeleton can be used as an effective tool for driving the decomposition process, but also as a powerful tool in conjunction with volume decomposition for volume texture mapping, volume compression for hardware-accelerated rendering and manipulation, and focus and context rendering. We presented our work on object retrieval using the curve-skeleton as a reduced representation for three-dimensional object matching. We concluded with an extension of our vector field-based curve-skeletonization framework to four dimensions and presented our initial results in using these curve-skeletons for indexing and retrieval of fMRI images by content.

The following publications resulted from the work presented here:

*Journal Publications:*


*Conference Publications:*


**Posters**


**MS Thesis**

• N.D. Cornea (2005). **A Visualization Tool for fMRI Data Mining.** M.S. Thesis, Department of Electrical and Computer Engineering, Rutgers, The State University of New Jersey, Piscataway, NJ, USA.

**In preparation**

• J. Novatnack, N.D. Cornea, A. Shokoufandeh, D. Silver, P. Kantor, B. Bai. **A Generalized Family of Fixed-RADIUS Distribution-Based Distance Measures for Comparing fMRI,** In preparation

• N.D. Cornea, C.D. Correa, D. Silver. *Volume decomposition for visualization,* To be submitted for publication.
References


[62] FMRIB Software Library: http://www.fmrib.ox.ac.uk/fsl


[66] L. Forsberg. Visualization and Query Tool for a Human Brain Database, Master’s Thesis, Royal Institute of Technology (KTH), Numerical Analysis and Computer Science Department, TRITA-NA-E04049.


[137] A. Lieutier. Any open bounded subset of $\mathbb{R}^n$ has the same homotopy type than its medial axis, ACM Shape Modeling and Applications, 2003.


[173] The OpenGL Architecture Revision Board.


[194] RUMBA: http://psychology.rutgers.edu/RUMBA


[220] SPM web site: http://www.fil.ion.ucl.ac.uk/spm/


[236] VoxBo: http://www.voxbo.org/index.html
[242] Vizlab Website, Rutgers, The State University of New Jersey: http://www.caip.rutgers.edu/vizlab_group_files/vizlab.html


Curriculum Vita

Nicu Daniel Cornea

October, 1996 - June, 2001  Technical University, Cluj-Napoca, Romania
Department of Computer Engineering
B.S. in Computer Engineering in June, 2001

Sep, 2001 - May, 2002  Software Engineer,
Siemens VDO Automotive, Timisoara, Romania

September, 2002 - January, 2007  Rutgers University, New Brunswick, NJ, USA
Department of Electrical and Computer Engineering
M.S. in Electrical and Computer Engineering in October, 2005

September, 2002 - May, 2003  Teaching Assistant
Rutgers University, New Brunswick, NJ, USA
Department of Electrical and Computer Engineering

September, 2003 – January, 2007  Graduate Assistant
Rutgers University, New Brunswick, NJ, USA
School of Communication, Information and Library Studies

Publications:

Journal Publications:


Conference Publications:

Posters

MS Thesis

In preparation
- N.D. Cornea, C.D. Correa, D. Silver. Volume decomposition for visualization, To be submitted for publication.