Volume Graphics (2003)
I. Fujishiro, K. Mueller, A. Kaufman (Editors)

Spatial Transfer Functions — A Unified Approach to Specifying Deformation in Volume Modeling and Animation

M. Chen*, D. Silver†, A. S. Winter*, V. Singh†, and N. Cornea†

* Department of Computer Science, University of Wales Swansea, Singleton Park, Swansea SA2 8PP, UK
† Department of Electrical and Computer Engineering, Rutgers, The State University of New Jersey, Piscataway, NJ 08855-0909, USA

Abstract

In this paper, we introduce the concept of spatial transfer functions *as a unified approach to volume modeling and animation. A spatial transfer function is a function that defines the geometrical transformation of a scalar field in space, and is a generalization and abstraction of a variety of deformation methods. It facilitates a field-based representation, and can thus be embedded into a volumetric scene graph under the algebraic framework of constructive volume geometry. We show that when spatial transfer functions are treated as spatial objects, constructive operations and conventional transfer functions can be applied to such spatial objects. We demonstrate spatial transfer functions in action with the aid of a collection of examples in volume visualization, sweeping, deformation and animation. In association with these examples, we describe methods for modeling and realizing spatial transfer functions, including simple procedural functions, operational decomposition of complex functions, large scale domain decomposition and temporal spatial transfer functions. We also discuss the implementation of spatial transfer functions in the* vlib *API and our efforts in deploying the technique in volume animation.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism; I.3.8 [Computer Graphics]: Applications.

1. Introduction

Geometric deformation is an important aspect of computer graphics and computer animation. With traditional surface representations, the control of deformation is usually achieved by moving the vertices of a polygonal mesh, or the control points of a parametric surface. Many methods have been proposed for specifying and controlling surface deformation. During the past decade, we also witnessed the extensive use of image distortion techniques, which played an important role in creating some stunning visual effects in the entertainment industry. Because of the extra dimension of information in volumetric representations and the increasing availability of volume datasets, it is desirable to deform and animate volumetric objects directly. Hence one of the critical measurements for the practical usefulness of volume graphics as a general purpose technology will be its capability of specifying intuitively, representing effectively, realizing accurately and rendering efficiently geometrical deformation of volumetric objects.

A collection of techniques for deforming volumetric objects have been developed for volume visualization [18], volume morphing [20, 8], and volume animation [14, 33]. Whilst many of these techniques were deployed successfully in specific applications, they lack in a unified conceptual framework to bring these techniques together and lack in a consistent approach to their implementation.

In this paper, we present the notion of *spatial transfer functions* as a unified approach to volume modeling and animation in the context of deformation. In Section 2 we will give a brief review of various deformation techniques, focusing on those used in volume visualization, volume morphing, volume animation and application areas such as medical imaging. In Section 3 we will define the concepts of spatial transfer functions in a broad scope of volume graphics, highlighting the role of *field-based representations* and embracing a generalized approach to the representation of spatial transfer functions. In Section 4, we will demonstrate the use of spatial transfer functions in volume visualization,

volume sweeping, volume deformation and volume animation. In particular, we will describe the methods for modeling and realizing a range of spatial transfer functions, including simple procedural functions, operational decomposition of complex functions, large scale domain decomposition and temporal spatial transfer functions. In Section 5, we will discuss a small collection of implementation issues. We will offer our observations and concluding remarks in Section 6.

2. Related Work

There has been extensive research effort in areas of geometric deformation, and a large collection of major developments were captured in several surveys. These include surveys by McInerney and Terzopoulos [23] on deformable models in medical image analysis, Gibson and Mirtich [16] on deformable modeling in computer graphics, Lemos and Wyvill [19] on deformable models in animation of articulated figures. Noh and Neumann [25] on facial modeling and animation. In respect to volumetric representations, a number of techniques have been developed, and applications in visualization, medical image analysis and computer animation have been explored.

In particular[†] feature-based techniques [20, 8] and physically-based and physically-plausible techniques [11, 15] were developed for deforming volume datasets. Deformable models were applied to volume visualization [18], tensor visualization [35], segmentation and registration [5, 24], motion analysis [26], and surgical simulation [29, 22]. Volume deformation also constitutes an integral component of many volume morphing techniques [20, 7] and perhaps more significantly it is an indispensable component in volume animation [33, 12, 13, 14].

Given a deformation specification that defines how an object is to be deformed to another (typically in the form of control information such as control points), it can be realized using either *forward mapping* (e.g., revoxelization [14]) or *backward mapping* (e.g., space transformation [18]). The backward mapping approach can be traced back to the 1980's when Barr first introduced the concept of space warping for deforming solids and parametric and implicit surfaces [3, 4]. In 1995, Kurzion and Yagel extended this generic approach for specifying and rendering volume deformation [18]. Both employed vector fields as the underlying representation in modeling deformation, and both utilized a ray-based mechanism in rendering deformation. Whilst Barr focused on a global space transformation, Kurzion and Yagel placed their emphasis on local transformation by introducing the notion of "ray deflectors".

Previous research effort has resulted in a variety of methods for constructing and realizing deformable models. However, we do not seem to have a unified approach to the specification and representation of geometric deformation. Without such an approach, inevitably, we are unable to address the following set of the questions:

- how to define deformation for a composite object built upon a set of simpler objects using a constructive term,
- how to incorporate deformation into a scene graph,
- how to combine global and local space transformation,
- how to combine different deformation techniques,
- how to reuse and modify a deformation specification in the same manner as an object specification.

3. Definitions and Concepts

In this section, we introduce the concepts of *spatial transfer function* as a unified approach to specifying deformation in volume modeling and volume animation. We first define the notion of *spatial objects*, and then show how a spatial transfer function can be specified and manipulated as a spatial object. Finally, we discuss the merits of a particular type of objects, namely *spatial displacement objects*.

3.1. Spatial Object

Let $\mathbb{R}$ denote the set of all real numbers, and $\mathbb{E}^3$ denote 3D Euclidean space. A *scalar field* is a function $F : \mathbb{E}^3 \rightarrow \mathbb{R}$. Conceptually a scalar field $F(p)$ is a generalization of a surface function that models only those points on the surface. A volumetric representation of an object can hence be specified as a set of scalar fields, $F_1(p), F_2(p), \ldots, F_k(p)$, that define the geometrical and physical properties of the object at every point p in three-dimensional space. Such an object is called a *spatial object*, which is a tuple, $\mathbf{o} = (A_0, A_1, \ldots, A_k)$, of *attribute fields* defined in $\mathbb{E}^3$. In volume visualization and volume graphics, it is common to define an attribute field upon another using a *transfer function*, usually in the form of $A_i(p) = \Phi(A_j(p)), p \in \mathbb{E}^3$ where $\Phi : \mathbb{R} \rightarrow \mathbb{R}$. Transfer functions are an intrinsic part of volume visualization, and in particular, direct volume rendering. It is also common to apply transfer functions during rendering, and this extends their role beyond the modeling of attributes.

A *volume dataset* is a discrete specification of a scalar field, which consists of up to three essential components: a set of samples $\mathbf{V}$, their topological relationships $\mathbf{T}$ and an interpolation function $\mathbf{I}$. The set $\mathbf{V} = \{(p_i, v_i) | i = 1, 2, \ldots, n\}$ defines the known value v_i at each sampling location p_i in $\mathbb{E}^3$. Some volume datasets (e.g., regular 3D grids) can be represented simply by a 3D array of values with implicit geometry and topology, whilst others (e.g., curvilinear grids, tetrahedral meshes) may require an explicit specification of sample positions, $\{p_i | i = 1, 2, \ldots, n\}$ or topological connectivities between the samples, $\mathbf{T}$. The role of the interpolation function $\mathbf{I}$ is to define the scalar values at all the points in $\mathbb{E}^3$ other than those sample positions $\{p_i | i = 1, 2, \ldots, n\}$. Typical interpolation functions include trilinear interpolation for

[†] It is not possible to give a substantial list of literatures on deformation and its applications. Here we highlight only those works that are particularly relevant and some representative applications.

regular grids, and barycentric interpolation for tetrahedral meshes.

Hence objects built from volume datasets are merely a subclass of spatial objects. As the concepts to be introduced can be applied to objects defined upon volume datasets, those specified mathematically or procedurally, and a combination of both, we will refer to all spatial objects in their general from $\mathbf{o} = (A_0, A_1, \ldots, A_k)$ though many objects illustrated will be built from volume datasets.

3.2. Spatial Transfer Function

A *spatial transfer function*, $\Psi : \mathbb{E}^3 \rightarrow \mathbb{E}^3$, is a function that defines the geometrical transformation of every point p in $\mathbb{E}^3$. It is used to modify the geometrical shape and position of a scalar field A usually in the form of

$$A'(p) = A(\Psi(p)),$$

or a spatial object in the form of

$$\mathbf{o}'(p) = \mathbf{o}(\Psi(p)) = (A_0(\Psi(p)), A_1(\Psi(p)), \ldots, A_k(\Psi(p))).$$

According to the above definition, an evaluation of A' at p implies the evaluation of A at $q = \Psi(p)$. As the actual point q is transferred to p during the evaluation, Ψ is in fact an backward mapping from p to q. This definition is well suited for the sampling process in both voxelization and ray-based direct volume rendering. For projection-based rendering methods, however, it would be desirable to specify a spatial transfer function as a forward mapping from q to p. To maintain the consistency in our discussion, we assume that all spatial transfer functions presented are backward mappings from p to q unless it is stated otherwise.

The use of spatial transfer functions differs from that of conventional transfer functions in a number of ways:

- In relation to a scalar field $A(p)$, a spatial transfer function Ψ is applied to p prior to the evaluation of A, whilst a conventional transfer function Φ is applied after the evaluation.
- The essential output of Ψ is a point $p' \in \mathbb{E}^3$, and the essential output of Φ is a scalar value $v' \in \mathbb{R}$. Typically, Ψ is a tri-variate function $(p'_x, p'_y, p'_z) = \Psi(p_x, p_y, p_z)$, whilst Φ is a univariate function $v' = \Phi(v)$. Both Ψ and Φ can of course be extended to include additional input variables.
- In relation to a spatial object $\mathbf{o}$, Ψ is normally applied uniformly to all of its attribute fields, whilst Φ is commonly employed to specify an undefined attribute field through a mapping from another, or modify the value ranges of an attribute field through re-mapping.

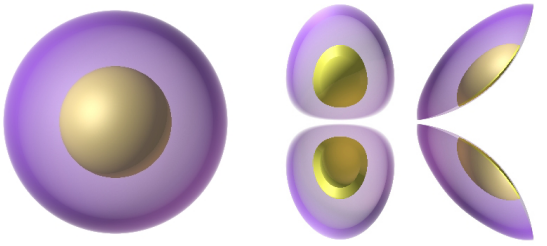
A spatial transfer function Ψ is in fact composed of three sub-functions, Ψ_x, Ψ_y, Ψ_z as:

$$\begin{pmatrix} p'_x \\ p'_y \\ p'_z \end{pmatrix} = p' = \Psi(p) = \begin{pmatrix} \Psi_x(p_x, p_y, p_z) \\ \Psi_y(p_x, p_y, p_z) \\ \Psi_z(p_x, p_y, p_z) \end{pmatrix}$$

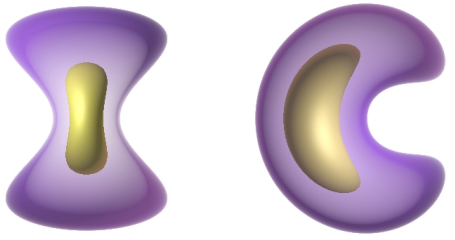
We notice a useful feature that $\Psi_x, \Psi_y, \Psi_z : \mathbb{E}^3 \rightarrow \mathbb{R}$ are essentially scalar fields. Hence all spatial transfer functions of a form $\Psi = (\Psi_x, \Psi_y, \Psi_z)$ can be considered as a class of spatial objects [9]. We use $\mathbf{s} = (A_x(p), A_y(p), A_z(p))$ to denote this particular class of spatial objects, and conveniently call them *spatial transfer objects* or *ST objects*. In the following discussions, the terms "ST object" and "spatial transfer function" are used interexchangeably, though the former places more emphasis on the specification of a spatial transfer function.

3.3. Specifying Spatial Transfer Functions

In much the same way as conventional spatial objects, the attribute fields, $A_x(p), A_y(p), A_z(p)$ (i.e., sub-fields $\Psi_x(p), \Psi_y(p), \Psi_z(p)$), of an ST object $\mathbf{s}$ can be defined using mathematical, procedural and discrete specifications [10]. An example of a discrete specification is a volume dataset. Figure 1 shows three simple spatial transfer functions applied to a spherical spatial object, which comprises a translucent shell and a solid core.



(a) the original spatial object (b) split (two views)



(c) squeeze (d) sweep (rotational stretch)

Figure 1: *The application of three simple spatial transfer functions in (b), (c) and (d) respectively, to a spatial object shown in (a).*

To model a volume graphics scene, a spatial transfer function can be incorporated into most types of volumetric scene graphs. In the case of the *vlib* API [31], a spatial transfer function is associated with and applied to a spatial object in a way similar to that for geometrical transformations such as rotation and scaling. In the case of the CSG tree used in implicit surface modeling [34], pre-defined spatial transfer functions

are built into the system, and applied to objects, as unary operators.

Under the algebraic framework of Constructive Volume Geometry (CVG) [9], we can introduce *spatial transfer* as a generic operator $\boxed{\mapsto}(\mathbf{s},\mathbf{o})$ where $\mathbf{s}$ is an ST object and $\mathbf{o}$ is the object to be spatially modified. Such an algebraic notion unifies

- the mechanism for specifying ST objects and that for other spatial objects (i.e., through the use of scalar fields),
- the mechanism for manipulating ST objects and that for other spatial objects (i.e., through the use of conventional transfer functions),
- the mechanism for applying a spatial transfer operation to a spatial object and that for other CVG operations (such as union and intersection),
- the mechanism for defining a global space transformation (i.e., an unbounded ST object) and that for a local space transformation (i.e., a bounded ST object).

Like other spatial objects, we can modify an ST object $\mathbf{s_1}$ using another ST object $\mathbf{s_2}$ by applying a spatial transfer operation $\boxed{\mapsto}(\mathbf{s_2},\mathbf{s_1})$. This allows a complex spatial transfer operation $\Psi(p)$ to be constructed from a set of simpler operations $\Psi_k(\ldots\Psi_2(\Psi_1(p))\ldots)$ as a CVG term:

$$\boxed{\mapsto}(\boxed{\mapsto}(\mathbf{s_k},\boxed{\mapsto}\ldots\boxed{\mapsto}(\mathbf{s_2},\mathbf{s_1}),\mathbf{o})$$

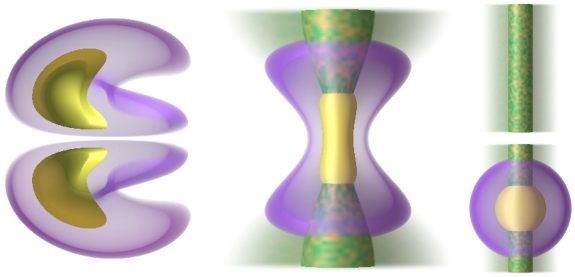
It also allows a spatial transfer operation $\Psi(p)$ to be applied to a complex spatial object $\mathbf{o}$ constructed from simpler objects $\mathbf{o_1},\mathbf{o_2},\ldots,\mathbf{o_n}$, in the form of a CVG sub-term Ω, that is,

$$\boxed{\mapsto}(\mathbf{s},\mathbf{o}) = \boxed{\mapsto}(\mathbf{s},\Omega(\mathbf{o_1},\mathbf{o_2},\ldots,\mathbf{o_n}))$$

Figure 2 shows an example of two spatial transfer functions applied to a spatial object, and an example of a spatial transfer function applied to the union of two simple spatial objects. In the first example shown in Figure 2(a), the object in Figure 1(a) is first swept rotationally as in Figure 1(d) and then split open in a manner similar to Figure 1(b). In the second example, a composite object, shown in Figure 2(c bottom), is first constructed by applying the CVG union operator $\boxed{\cup}$ to the spherical object in Figure 1(a) and a cylindrical object that has an amorphous outer layer (Figure 2(c top)). A squeezing function (similar to that in Figure 1(c) is then applied to the composite object, resulting in a spatially transferred composite object as shown in Figure 2(b).

3.4. Spatial Displacement Object

Given $q = \Psi(p)$, we can introduce a *spatial displacement function* $\Delta(p) = \Psi(p) - p = q - p$. Similarly we can specify $\Delta(p)$ as a type of spatial object $\mathbf{d}$, which we call a *spatial displacement object* or an *SD object*. We can define a generic constructive operator $\boxed{\mapsto\!\!|}$ that takes an SD object $\mathbf{d}$, and applies to an spatial object $\mathbf{o}$ in the form of $q = p + \Delta(p)$, where $\Delta(p)$ is defined by $\mathbf{d}$.



(a) $\boxed{\mapsto}(\boxed{\mapsto}(\mathbf{s_2},\mathbf{s_1}),\mathbf{o})$ (b) $\boxed{\mapsto}(\mathbf{s},\boxed{\cup}(\mathbf{o_1},\mathbf{o_2}))$ (c)

Figure 2: *Constructing CVG terms involving ST objects.*

The operation $\boxed{\mapsto\!\!|}(\mathbf{d},\mathbf{o})$ seems to have introduced additional work in evaluating $\Psi(p)$, and especially specifying $\Delta(p)$ (or an SD object) is less intuitive than that for $\Psi(p)$ (or an ST object). However, $\Delta(p)$, which defines the relative displacement, exhibits some useful features that $\Psi(p)$ does not. Like other ordinary spatial objects, conventional transfer functions can be used to define or modify attribute fields of an SD object, and combinational operators, such as union, intersection and blending, can also be applied to SD objects.

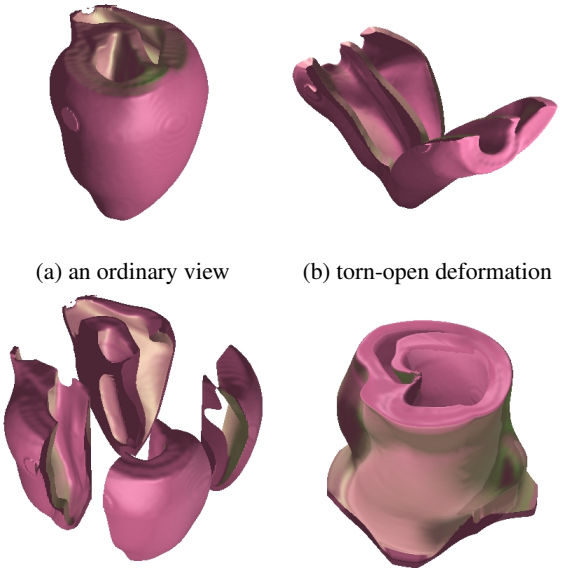
4. Spatial Transfer Function in Action

In this section, we demonstrate the use of spatial transfer functions with the aid of examples in volume visualization, sweeping, deformation and animation. In association with volume visualization, we show how procedural spatial transfer functions can be used effectively to assist in visualization. In association with volume sweeping, we discuss the specification of complex spatial transfer functions, and the modeling of backward mapping for a forward sweeping. In association with volume deformation, we show how a complex spatial transfer function can be decomposed into a set of relatively simpler operations. In association with volume animation, we discuss the large scale domain decomposition when spatially transform a complex animation model.

4.1. Volume Visualization

One of the most important uses of volume visualization is to examine the interior of volumetric objects embedded in volume datasets. Although the introduction of transparency and amorphous effects through the use of transfer functions is an effective means to display the internal structure, it is sometimes more intuitive, and hence more effective, to employ "direct actions" such as simply cutting an object open. Figure 3 shows three such "direct actions" digitally on a rabbit heart dataset in a non-invasive manner.

The spatial transfer function Ψ_1 in Figure 3(b) tears open the heart, revealing the internal structure clearly. The spatial transfer function Ψ_2 in Figure 3(c) digitally dissects the heart to four quarters and translates each quarter away



(a) an ordinary view (b) torn-open deformation

(c) quartile dissection (d) inside out

Figure 3: *Four visualizations of a heart dataset.*

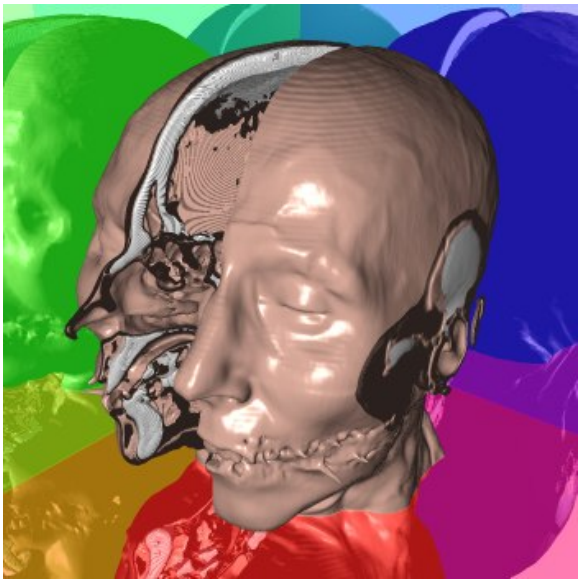


Figure 4: *Opening a CT dataset.*

from the center axis. The spatial transfer function Ψ_3 in Figure 3(d) takes an axis passing through one of its chambers, and flip the heart inside out about the axis.

Figure 4 shows the application of an opening spatial transfer function Ψ_4 to a more complicated volume dataset. From the opening, one can clearly visualize the internal structures, including bones and soft tissues, in relation to the external skin surface.

Two example spatial transfer functions, Ψ_2 and Ψ_4, which are defined in a normalized volume coordinate system where $p \in [0,1]^3$, are given below:

$$\Psi_2(p) = (\psi_2(p_x, 0.3), \psi_2(p_y, 0.3), p_z),$$

$$\psi_2(a,b) = \begin{cases} a/(1-b) & 2a \leq 1-b); \\ 1-(1-a)/(1-b) & 2a > 1+b); \\ -1 & \text{otherwise.} \end{cases}$$

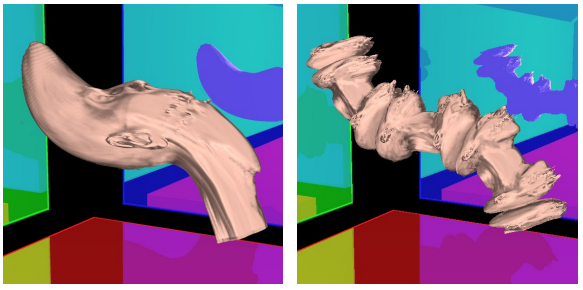
$$\Psi_4(p) = \begin{cases} (p_x + 0.2 - 0.2p_z, p_y, p_z) & p_x < 0.3 + 0.2p_z; \\ (p_x - 0.2 + 0.2p_z, p_y, p_z) & p_x < 0.3 + 0.2p_z; \\ (-1,-1,-1) & \text{otherwise.} \end{cases}$$

4.2. Volume Sweeping

Volume sweeping is a modeling method for defining a spatial domain. Winter and Chen recently reported that imagery templates (i.e., 2D images and 3D volume datasets) can be used to construct a swept volume effectively [32]. Conceptually, a sweep is a way of specifying an ST object, and an imagery template is composed of one or more discrete scalar fields, with which a spatial object can be defined. The construction of a swept volume is hence an application of a spatial transfer function Ψ to a spatial object. Although only spatial objects built from images and videos were considered in [32], the method can be generalized to any spatial objects.

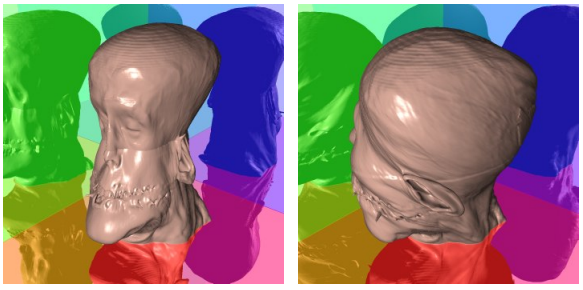
Technically, however, it is not always straightforward to obtain a spatial transfer function Ψ, in the form of a backward mapping, from the description of an arbitrary sweep ρ, which may include the specifications of a sweeping trajectory, a sweeping direction-vector, a sweeping up-vector, and a scaling function. Hence, a specification of a sweep is considered *simple* if we can obtain a spatial transfer function Ψ which is an inverse mapping of ρ, can be evaluated with reasonable accuracy and a reasonable amount of computing resource, and facilitates a one-to-one or a many-to-one mapping from the swept space to the template space. If Ψ is a one-to-many or many-to-many mapping, a constructive operator must be applied to the multiple values sampled from the template space. If Ψ cannot be evaluated with reasonable accuracy and a reasonable amount of computing resource, voxelization will likely be a better option [32].

Figure 5 shows two sweeps with a volume dataset as the sweeping template. One "stretches" the volume dataset along a Bézier trajectory, whilst the other sweeps the dataset in sections using a more sophisticated function for specifying the volume/path association. The Bézier sweeps used in the images are directly evaluated using a numerical root finder [32].



(a) stretching repeating(b)

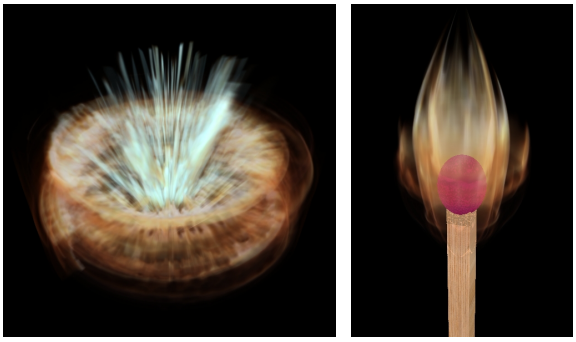
Figure 5: *Sweeping a CT dataset along a Bézier trajectory.*



(a) squeeze (b) twist

Figure 7: *Relatively simple procedural deformation.*

Figure 6 shows two examples of sweeping a volume dataset of a small flame captured as a video. In (a), two separate sweeps are combined together using a CVG union operation, creating a small explosion. In (b), a sweep, which uses a smaller number of video frames, is combined with other objects modeling a matchstick. Both demonstrate the use of spatial transfer functions in simple volumetric scene graphs.



(a) union of two sweeps (b) matchstick

Figure 6: *Sweeping a video of a flame.*

4.3. Free-Form Volume Deformation

For simple deformation of a volume dataset such as those shown in Figure 7, the most effective way is to define a procedural spatial transfer function. However, most procedural specifications are not suitable for direct manipulation in a user interface though they can be controlled using some parameters.

One well-established deformation technique in surface graphics is *Free-Form Deformation (FFD)* originally developed by Sederberg and Parry [27]. It allows a user to embed an object in a lattice of grid points, and manipulate the grid points to achieve desired deformation. It is therefore important to deploy this technique in volume graphics.

Consider a global FFD where a bounded spatial object is embedded in the parameter domain of a free-form volume $P(u,v,w)$, where $(u,v,w) \in [0,1]^3$. The role of a spatial transfer function in this case is, for a given point $p \in \mathbb{E}^3$, to determine the corresponding parametric coordinate (p_u, p_v, p_w) if p resides inside or on the surface of $P(u,v,w)$. From (p_u, p_v, p_w), a point q in the original spatial object can be easily obtained. However, for most commonly-used deformation prescription, such as Bézier and B-spline, it would be incredibly difficult to determine a backward mapping from the Euclidean space $\mathbb{E}^3$ to the parameter space.

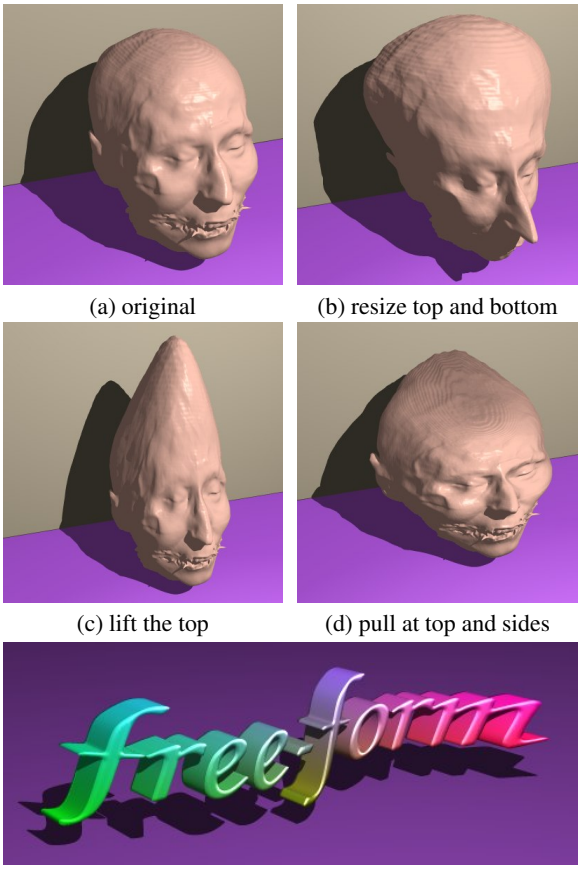
We thus adopt an approach similar to that of Joy and Duchaineau [17] who discretized the boundary faces of a parameter volume. Consider Bézier as the deformation prescription. We discretize parametric space by imposing on it a 3D rectilinear grid comprising cubic or cuboidal cells in parametric coordinates (which are hexahedral regions (i.e., irregular six-face boxes) in the the deformed Euclidean space). Each vertex in the grid, where a parametric coordinate, (v_u, v_v, v_w), is known, is transformed into Euclidean space using the Bézier volume equation:

$$P_B(v_u, v_v, v_w) = \sum_{k=0}^{3} \sum_{j=0}^{3} \sum_{i=0}^{3} C_{i,j,k} B_i(v_u) B_j(v_v) B_k(v_w),$$

where $(v_u, v_v, v_w) \in [0,1]^3$ is the parametric coordinate of the vertex, $C_{i,j,k}$ is a Bézier volume control point and B_i, B_j, B_k are the Bernstein basis functions. Each vertex in the grid maintains a record of both the original parametric coordinate (v_u, v_v, v_w) and the computed Euclidean coordinate (v_x, v_y, v_z). Hence each cell encloses a small parametric domain, which is further divided into six tetrahedra. Given the information stored on the vertices of each tetrahedron, an approximate local spatial transfer function can be obtained by using barycentric interpolation. The problem of determining (p_u, p_v, p_w) is thus transformed to the search for a tetrahedron containing p — a point location problem. Hence a highly complex spatial transfer function is decomposed into many simpler spatial transfer functions operating on individual tetrahedral domains.

Figure 8 shows a set of five free-form deformations of

volumetric objects. A detailed description of a brute force rendering method, its acceleration using a custom octree, and control specifications used in the examples can be found in [30].



(a) original (b) resize top and bottom

(c) lift the top (d) pull at top and sides

(e) horizontally bend the text string

Figure 8: *Free-form volume deformation.*

4.4. Volume Animation

In [12, 13, 14] a system for creating volume animations was presented. It was based upon a *volumetric skeleton*, which was extracted from a volumetric representation and was animated. New volumes were reconstructed about the transformed skeletons and then rendered using a ray caster. In [28], a system called *VolEdit* was presented which allowed a user to manipulate or animate a volume in real-time. In *VolEdit*, the skeleton is used only to help specify deformation and logically subdivide the volume into components. Each skeleton "limb" defines a *bounding box* about a portion of the volume. The bounding boxes are determined using mid-plane geometry [6] which maintains connected rectangular regions during animation by adding planar polygons at the joints. The skeleton can be deformed interactively, through the *VolEdit* interface, or off-line, for example, using a standard animation package such as *Character Studio* [1] or *Maya* [2]. In [28], the volume was rendered using the texture mapping hardware by slicing the bounding boxes and mapping back to the original volume. The bounding boxes are of a simple geometry that can support interactivity. This approach to volume animation well suited to be realized using spatial transfer functions in conjunction with a volume scene graph.

A *temporal spatial transfer function* is a spatial transfer function that defines geometrical transformation over a temporal domain, i.e., $p[t] = \Psi(p[0], t)$, $t_{min} \leq t \leq t_{max}$. A kinematic specification typically defines a series of sub-functions, $\Psi_1, \Psi_2, \Psi_3, \ldots$ over a discrete set of time steps. Given a spatial transfer function for animating an actor built from one or more volume datasets, one can create new discrete volumetric representation of the actor at each time step using voxelization. However, this will incur an excessive amount of storage for any complex actor models. An alternative approach is to maintain a temporal spatial transfer function (or a series of sub-functions). Together with the original static model, such a spatial transfer function can be rendered directly. Hence there is no need to generate explicitly discrete volumetric representations for a deformed actor.

There are two different cases for animation. The simple case is that the each volume dataset (part of a scene) has a set of temporal spatial transfer functions governing its motion. The temporal spatial transfer functions are applied directly, just as the regular spatial transfer functions, resulting in an animation. A more complicated case arises for animation within volume datasets (i.e., animating parts of a volume) or for a set of volume datasets which must maintain connectivity over the animation (i.e., animating a composite object).

For the latter case, we utilize the "skeleton" concept from [12, 14] which is an intuitive mechanism to define kinematic animation. A skeleton is of a set of skeletal segments, each of which defines a box-like bounding region of the volume that should normally deform together. As the animation progresses, the box-like hexahedral regions change to maintain connectivity about the "joints". This representation can also be used to keep objects which are composed from different volume datasets connected. Whilst the connectivity is useful in many circumstances, it is not an essential requirement. As shown in Figure 9, different regions of a volumetric actor (e.g., a spatial object built from the Visible Human dataset) can be made to deform in a disjointed manner though the original skeleton and the volumetric representation are both connected. Although this is of a creative nature in the context of volume animation, it shares the same spatial transfer mechanism with Figure 3(c), which is of a investigatory nature in the context of visualization.

Because many of the tools for standard computer animation (such as motion capture) use skeletons to control kinematics, one of the main advantages of using a skeleton in volume animation is that the dynamic specifications

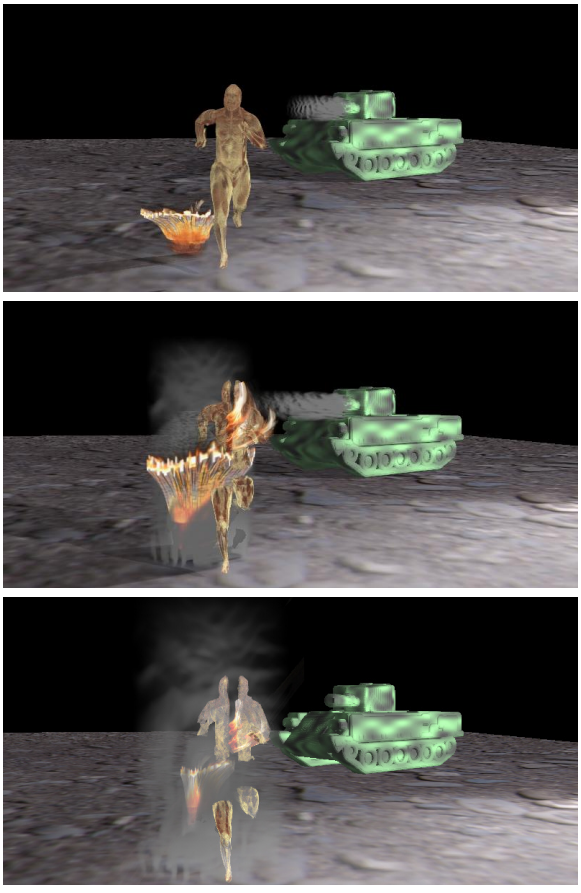


Figure 9: *Regions of a volumetric actor moving apart.*

produced by these tools can now be incorporated in a volume graphics pipeline such as the *vlib* pipeline [31]. Figure 10 shows five frames extracted from an animation of the visible human is shown through a scene modeled and rendered using *vlib*. The Visible Human actor was decomposed into about 15 hexahedral regions, each is associated with a spatial temporal transfer function. The joining of these hexahedral regions is achieved using the union operator in *vlib*.

5. Ray-based Software Implementation

In the above sections, we have discussed the modeling aspect of spatial transfer functions. To render a volumetric scene graph involving spatial transfer functions, one may (i) voxelize the scene graph and then render the resultant volume dataset, (ii) project elements of the scene graph (where spatial transfer functions are forward mappings) with the aid of a visibility determination algorithm, or (iii) evaluate the scene graph using a ray-based renderer. Approach (i) may not always be practical due to the huge storage space requirement, and the likely degeneration of data quality. Approach (ii) could be hindered by unpredictable collisions between different sub-graphs caused by spatial transfer functions. Approach (iii), though it may not be the fastest in rendering speed, inherently facilitates direct evaluation of a volumetric scene graph and associated spatial transfer function during rendering time.

To evaluate a spatially-transformed spatial object $\mathbf{o}'(p) = \mathbf{o}(\Psi(p))$, we need first compute the actual point to be sampled $q = \Psi(p)$, then sample all attributes of object $\mathbf{o}$ at position q and finally render the sampled attributes at p for object $\mathbf{o}'$.

To render the animation shown in Figure 10, a combination of a backward and forward mapping algorithm was used. The entire volume is rendered as a backward mapped process, and this represents the gross motion as the Visible Human actor is animating through the dataset. Within these bounds, individual limbs are defined in a manner of forward mapping, that is, the control nodes of the "bounding regions" (hexahedral for this implementation) are transformed in line with time. A ray hitting one of these transformed boxes must then perform an inverse transformation to determine the actual scalar value of that intersection point. The specification and implementation of this particular spatial transfer function is similar to the ray-box intersection in TROVE [21]. Alternatively, entire space of each individual hexahedral region can be backward mapped, using a backward spatial transfer function, to the corresponding region in the original dataset for rendering.

The advantage of using a forward mapping process for moving regions is that it can take advantage of hardware assisted rendering. This can be achieved by rendering forward-mapped limbs by inverse mapping the sliced polygons. This would enable skeleton based interactive multi-volume scene editing. Furthermore, the use of hexahedral bounding regions (simple geometry) makes it suitable for achieving higher interactivity with hardware rendering.

6. Conclusions

We have presented the concept of spatial transfer functions, and the method for specifying them as spatial objects. This has provided us a unified approach to the specification of deformation in volume modeling and animation, and facilitated the use of volume scene graphs for constructing complex deformable models. We have demonstrated the capability of this unified approach through examples of its deployment in volume visualization, volume sweeping, free-form deformation and volume animation.

This generalization has brought a range of deformation techniques into a common software framework, and it has provided a unified interface to the implementation of these techniques, for example, procedural deformation in volume visualization, backward mapping in template sweeping, operational decomposition and domain decomposition in free-form deformation and volume animation. Some of these

techniques can be used effectively in volume visualization. Many have the potential to be deployed in virtual environment for supporting complex interactive manipulation of volumetric objects. All techniques presented can play a role in volume graphics for supporting creative modeling and animation using volumetric representations. We believe that this unification will help shape the future generation of volume graphics software.

Acknowledgments

Authors gratefully acknowledge the individual support from different funding organizations, including a US NFS grant (0118760) to D. Silver, a UK EPSRC grant (GR/R25286/01) to M. Chen, and a UK EPSRC PhD studentship to A. S. Winter. The authors would like to thank Alfie Abdul Rahman (University of Wales Swansea) for her help in technical communications between Swansea and Rutgers. The authors also wish to acknowledge the sources of some datasets used in the paper. They are the Visible Human from U.S. National Library of Medicine; the CT head from University of North Carolina, Chapel Hill; the rabbit heart from University of California, San Diego, the tank from Miloš Šrámek and Arie Kaufman; and the fire dataset captured by Mark Kiddell.

References

1. *Character Studio Animation Toolkit*. http://www.discreet. com/products/cs/, 2002.
2. *Maya: A 3D Animation and Visual Effects Tool*. http://www. aliaswavefront.com/en/products/maya/index.shtml, 2002.
3. A. Barr. "Global and local deformation of solid primitives". *Computer Graphics (Proc. SIGGRAPH 84)*, **18**(3):21–30, July 1984.
4. A. Barr. "Ray tracing deformed surfaces". *Computer Graphics (Proc. SIGGRAPH 86)*, **20**(4):287–296, July 1986.
5. I. Carlbom, D. Terzopoulos, and K. Harris. "Computer-assisted registration, segmentation, and 3D reconstruction from images of neuronal tissue sections". *IEEE Transactions on Medical Imaging*, **13**(2):351–362, 1994.
6. J. Chadwick, D. Haumann, and R. Parent. "Layered construction for deformable animated characters". *Computer Graphics (Proc. SIGGRAPH 89)*, **23**(3):243–252, July 1989.
7. M. Chen, M. W. Jones, and P. Townsend. "Methods for volume metamorphosis". In Y. Paker and S. Wilbur, editors, *Image Processing for Broadcast and Video Production*, 282–292. Springer-Verlag, 1995.
8. M. Chen, M. W. Jones, and P. Townsend. "Volume distortion and morphing using disk fields". *Computers and Graphics*, **20**(4):567–575, 1996.
9. M. Chen and J. V. Tucker. "Constructive volume geometry". *Computer Graphics Forum*, **19**(4):281–293, 2000.
10. M. Chen, A. S. Winter, D. Rodgman, and S. M. F. Treavett. "Enriching volume modelling with scalar fields". In F. Post, G. Nielson, and G.-P. Bonneau, editors, *Data Visualization: The State of the Art*, 345–362. Kluwer Academic Publishers, 2003.
11. Y. Chen, Q. Zhu, and A. Kaufman. "Physically-based animation of volumetric objects". In *Proc. IEEE Computer Animation '98*, 154–160, 1998.
12. N. Gagvani and D. Silver. "Parameter controlled volume thinning". *Graphical Models and Image Processing*, **61**(3):149–164, 1999.
13. N. Gagvani and D. Silver. "Animating the visible human dataset". In *Proc. the Visible Human Project Conference*, http://www.nlm.nih.gov/research/visible/vhpconf2000/, 2000. Online proceedings.
14. N. Gagvani and D. Silver. "Animating volumetric models". *Graphical Models*, **63**(6):443–458, 2001.
15. S. Gibson. "3D chainmail: a fast algorithm for deforming volumetric objects". In *Proc. 1997 Symposium on Interactive 3D Graphics*, 149–154, April 1997.
16. S. Gibson and Mirtich. *A survey of deformable modeling in computer graphics*. Technical Report TR97-19, MERL Technical Report, 1997.
17. K. I. Joy and M. Duchaineau. "Boundary determination for trivariate solids". In *Proc. Pacific Graphics '99*, 82–91, Seoul, Korea, October 1999.
18. Y. Kurzion and R. Yagel. "Space deformation using ray deflectors". In *Proc. the 6th Eurographics Workshop on Rendering*, 21–32, Dublin, Ireland, June 1995.
19. R. Lemos and B. Wyvill. *A survey of layered construction techniques using deformable models in the animation of articulated figures*. Technical Report 1998-637-28, University of Calgary, 1998.
20. A. Lerios, C. D. Garfinkle, and M. Levoy. "Feature-based volume metamorphosis". In *Proc. SIGGRAPH'95*, 449–456, Los Angeles, CA, August 1995.
21. A. Leu and M. Chen. "Modelling and rendering graphics scenes composed of multiple volumetric datasets". *Computer Graphics Forum*, **18**(2):159–171, 1999.
22. W.-T. Lin and R. A. Robb. "Dynamic volume texture mapping and model deformation for visually realistic surgical simulation". In *Proc. Medicine Meets Virtual Reality*, 62:198–204, 1999.

23. T. McInerney and D. Terzopoulos. "Deformable models in medical image analysis: a survey". *Medical Image Analysis*, **1**(2):91–108, 1996.

24. J. V. Miller, D. E. Breen, W. E. Lorensen, R. M. O'Bara, and M. J. Wozny. "Geometrically deformed models: a method for extracting closed geometric models from volume data". *Computer Graphics (Proc. SIGGRAPH 91)*, **25**(4):217–226, July 1991.

25. J.-Y. Noh and U. Neumann. *A survey of facial modeling and animation techniques*. Technical Report 99-705, University of Southern California, 1998.

26. J. Park, D. Metaxas, A. A. Young, and L. Axel. "Deformable models with parameter functions for cardiac motion analysis from tagged MRI data". *IEEE Transactions on Medical Imaging*, **15**(3):278–289, 1996.

27. T. W. Sederberg and S. R. Parry. "Free-form deformation of solid geometric models". *Computer Graphics (Proc. SIGGRAPH 86)*, **20**(4):151–160, 1986.

28. V. Singh and D. Silver. "Hardware accelerated volume manipulation". 2002. Submitted for Publication.

29. G. J. Wiet, R. Yagel, D. Stredney, et al. "A volumetric approach to virtual simulation of functional endoscopic sinus surgery". In *Proc. Medicine Meets Virtual Reality*, 167–179, 1997.

30. A. S. Winter. *Volume Graphics: Field-based Modelling and Rendering*. PhD thesis, University of Wales Swansea, United Kingdom, 2002.

31. A. S. Winter and M. Chen. "vlib: a volume graphics API". In K. Mueller and A. Kaufman, editors, *Proc. Volume Graphics 2001*, 133–147, New York, 2001. Springer Wien New York. see also http://vg.swan.ac.uk/vlib.

32. A. S. Winter and M. Chen. "Image-swept volumes". *Computer Graphics Forum*, **21**(3):441–450, 640, 2002.

33. Z. Wu and E. C. Prakash. "Visible human animation". In M. Chen, A. Kaufman, and R. Yagel, editors, *Volume Graphics*, 243–252. Springer, London, 2000.

34. B. Wyvill, A. Guy, and E. Galin. "Extending the csg tree. warping, blending and boolean operations in an implicit surface modeling system". *Computer Graphics Forum*, **18**(2):149–158, 1999.

35. X.-Q. Zheng and A. Pang. "Volume deformation for tensor visualization". In *Proc. IEEE Visualization 2002*, 379–386, 2002.

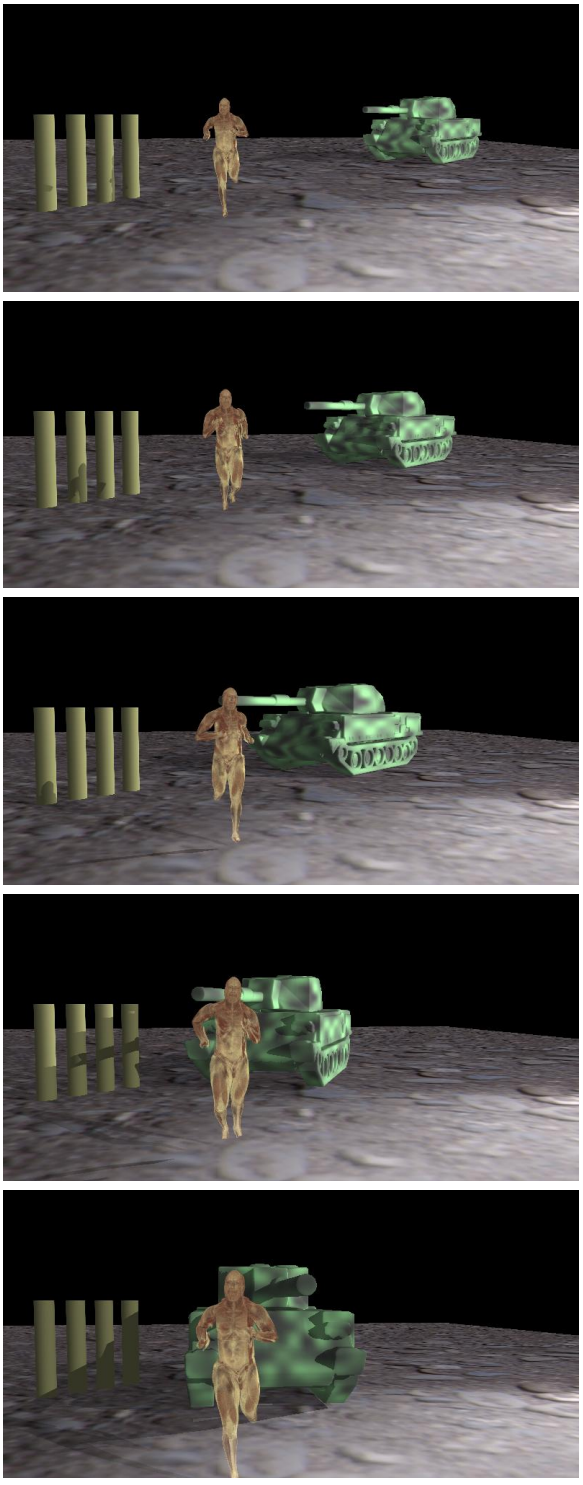


Figure 10: *A volume animation sequence.*