Spatial and Temporal Splitting of Scalar Fields in Volume Graphics

Shoukat Islam[a*] Swapnil Dipankar[b†] Deborah Silver[b‡] Min Chen[a§]
[a] University of Wales Swansea, UK [b] Rutgers, The State University of New Jersey, USA

ABSTRACT

Splitting a volumetric object is a useful operation in volume visualization and volume animation, but is not widely supported by existing systems for volume-based modeling and rendering. In this paper, we present an investigation into two main algorithmic approaches, namely explicit and implicit splitting, for modeling and rendering splitting actions in the context of both volume visualization and volume animation. We consider a generalized notion based on scalar fields, which encompasses discrete specifications (e.g., volume data sets) as well as procedural specifications (e.g., hypertextures) of volumetric objects. We examine the correctness, effectiveness, efficiency and deficiencies of each approach in specifying and controlling a spatial and temporal specification of splitting. We propose methods for implementing these approaches and for overcoming their deficiencies. We demonstrate the use of these approaches with examples of medical visualization, volume animation and special effects.

CR Categories: 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.

Keywords: volume graphics, volume animation, volume visualization, spatial transfer function, constructive volume geometry, volumetric scene graph, volume partition, volume splitting, fire effect, explosion effect.

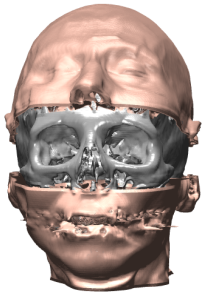
1 INTRODUCTION

Volumetric splitting is used in both scientific visualization and volume graphics. Splitting is the process of dividing a volume into two or more components. Such a process can be governed by data-independent geometric rules, such as when splitting a volume into halves to visualize the interior [4], or by data dependent logical rules, such as when segmenting a volume into different semantic components [18]. Once split, different transformations and rendering algorithms can be applied to each component of the split.

In scientific visualization, splitting can be used to convey more meaningful information, enabling users to browse parts of a volume that are normally occluded and to see both the focus and context (f+C) of the region of a volume that is of interest. Combined with more logical data partitioning (such as segmentation or "semantic layers" [18]), new visual representations can be created which highlight important areas of investigation. In volumetric manipulation and animation, splitting is used to designate different parts of the volume for spatial transformation over time. It can be used to create more intuitive motion in visualization, such as opening the chest of a virtual patient, or to create special effects in volume animation including explosion effects and fusion or merging actions.

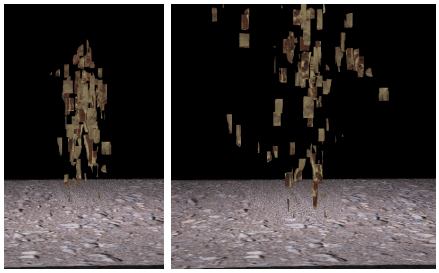
Two sets of examples of splitting are shown in Figure 1. Images (a) and (b) show two logical splits based on semantic contents of the data models. While (a) is a special effect produced for film A.I., (b) is modeled and rendered using volume graphics techniques, which will be part of the focus of this paper. Image (c) shows a partitioning of the visible man data set in volume animation [4], where the splitting could be more dynamic and vibrant. As image (c) simulates an explosion, its realism could be improved significantly by increasing the number of splits, from currently 4 to a large number, for example, 128 as in (d). However, such an increment challenges both modeling and rendering of these splits. This provides us with another motivation to deliver more sophisticated methods for specifying, controlling and rendering spatial and temporal splitting of scalar fields in volume graphics and to examine the scalability of splitting actions.

(a) a special effect in film A.I., Warner Bros., 2001



(b) a similar image that may aid medical visualization

(c) 4 splits



(d) 128 splits (frame numbers 3 and 15)

Figure 1: Two example scenarios where splitting can be deployed in volume visualization and volume animation.

In this paper, we discuss the various approaches to the splitting of a volume and show the different applications of splitting, including for improved volume visualization, volume animation and special effects. After a brief review of the related work in Section 2, we will outline, in Section 3, the concepts of, and general approaches, to splitting in volume graphics. In Sections 4 and 5, we will present two main approaches to the specification of splitting, namely *explicit splitting*, where the volume is explicitly carved into a number of pieces before rendering, or *implicit splitting*, where the "carve" is defined functionally in the modeling stage but only realized in the rendering stage and the volume is never actually partitioned. In

*e-mail: cshoukat@swansea.ac.uk
†e-mail: dipankar@caip.rutgers.edu
‡e-mail: silver@caip.rutgers.edu
§e-mail: m.chen@swansea.ac.uk

Section 6, we will present a few example applications, including logical splits in volume visualization, manipulation of segmented volume data sets, and special effects in volume animation. This will be followed by our concluding remarks in Section 7.

2 Related Work

In [18], a volume visualization technique was presented to "browse" volume data. Different tools were created to "split" the data and then render the splits using different rendering functions. While the work focused on an interactive visualization process, the splitting implemented treated individual voxels independently, which leads to some deficiencies in the quality of rendering.

As splitting can facilitate a "focus + context" view of the occluded regions in data sets, a number of researchers have previously employed this concept in visualization. These include works on volume deformation [14], distortion viewing [3], linked volumes [8], magnification lens [15], two-level rendering [13], hardware-accelerated volume rendering [22], and application in cardiac visualization [5].

Splitting is in fact a kind of *deformation*, where the restriction on continuity is deliberately lifted. Hence, many deformation methods existing in computer graphics (e.g., [1, 12, 19]) may also be relevant to this work in a more general context. The recent introduction of *spatial transfer functions* in volume modeling and rendering was an attempt to build a bridge between the deformation concepts in surface graphics and volume graphics [4].

3 Concepts and Approaches

Consider a volumetric representation of an object that is 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 $\mathbb{E}^3$. Such a representation scheme encompasses both procedural specifications of scalar fields such as solid and hyper-textures, and discrete specifications such as volume data sets. An object defined with such a scheme is referred to as a *spatial object*, which is essentially a tuple, $\mathbf{o} = (A_0, A_1, \ldots, A_k)$, of *attribute fields* defined in $\mathbb{E}^3$. We usually make attribute field A_0 an opacity field which in essence defines the visible geometry of a spatial object.

The action of *splitting* a spatial object in volume graphics spatially and temporally can thus be defined as follows:

> **Definition.** Consider an arbitrary spatial object $\mathbf{o}$ and a set of purposely constructed component objects $\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \ldots, \mathbf{s}_n$. The action of splitting is a series of transformation functions $T_{i,t}$ applied to $\mathbf{s}_i, i = 1, 2, \ldots, n$, over a period of time $t = 0, 1, \ldots, t_{max}$, such that, at $t = 0$, the union of all transformed objects $\mathbf{s}_i (i = 1, 2, \ldots, n)$ is equal to $\mathbf{o}$. In other words,
>
> $$\mathbf{o} = \boxed{\cup}(T_{1,0}(\mathbf{s}_1), T_{2,0}(\mathbf{s}_2), \ldots, T_{n,0}(\mathbf{s}_n)).$$

Note that it is the possible for $T_{i,t}$ to turn an object into an invisible object by either transforming its geometry into a point of zero size, or transforming its opacity to fully transparent. In either case, an invisible object has a zero sum of opacity over its volume V, that is, $\int_V A_0(p)\,dp = 0$ where A_0 is assumed to be the opacity field of the object.

Figure 2 illustrates an action of splitting of a spatial object. With the above abstraction, the modeling of splitting a spatial object becomes primarily two inter-related tasks, namely, the specification of a set of component objects $\mathscr{S} = \{\mathbf{s}_i | i = 1, 2, \ldots, n\}$ and that of a set of transformation functions $\mathscr{T} = \{T_{i,t} | i = 1, 2, \ldots, n; t = 0, 1, \ldots, t_{max}\}$.

The union operation $\boxed{\cup}$ is another critical element in the validity of a splitting scheme. In this work, we adopt a generalized union

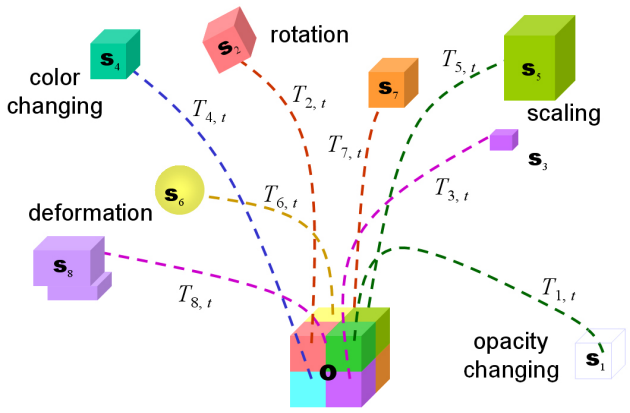


Figure 2: An action of splitting a spatial object.

operator based on [6], that is,

$$\begin{aligned}
&\boxed{\cup}(\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n = (\\
&\quad \textbf{MAX}(A_{1,0}, A_{2,0}, \ldots, A_{n,0}),\\
&\quad \textbf{SELECT}(< A_{1,0}, A_{1,1} >, < A_{2,0}, A_{2,1} >, \ldots, < A_{n,0}, A_{n,1} >),\\
&\quad \ldots,\\
&\quad \textbf{SELECT}(< A_{1,0}, A_{1,k} >, < A_{2,0}, A_{2,k} >, \ldots, < A_{n,0}, A_{n,k} >) \quad)
\end{aligned}$$

where $A_{i,0}$ is the opacity field of the i^{th} spatial object, and $A_{i,j}$ is its j^{th} attribute field. Field operator **MAX** is a pointwise extension of scalar operation $\mathbf{max}(a_1, a_2, \ldots, a_n)$ that returns the maximum value among $(a_1, a_2, \ldots, a_n)$. Similarly, **SELECT** is an extension of scalar operation $\mathbf{select}(< a_1, b_1 >, < a_2, b_2 >, \ldots, < a_n, b_n >)$ that returns the value of b_i if $a_i = \mathbf{max}(a_1, a_2, \ldots, a_n)$ (when there are more than one maximum value, the smallest i is chosen).

Other union operators can also be considered. However, by adopting one operator enables us to maintain the consistency of our discussions throughout the paper, while focusing on the specification of sets $\mathscr{S}$ and $\mathscr{T}$.

There are perhaps many schemes for specifying $\mathscr{S}$ and $\mathscr{T}$. It is, however, necessary for any scheme to address the following technical issues.

- *Explicit or implicit splitting*. We may choose to explicitly split a spatial object $\mathbf{o}$ into a component set $\mathscr{S}$. As long as the union of $\mathscr{S}$ is the same as $\mathbf{o}$ at $t = 0$, the original object $\mathbf{o}$ needs not to be featured in the action of splitting, unless for a specifically desired effect. Alternatively, we may create a component set $\mathscr{S}$ where each element $\mathbf{s}_i$ is mapped onto part of $\mathbf{o}$ using a spatial transfer function. In this case, it is necessary to maintain the original object $\mathbf{o}$, which is partitioned implicitly, as a "ghost" object in the action of splitting.
- *Logical or semantic partitioning*. An action of splitting involves partitioning the geometry of a spatial object $\mathbf{o}$ and moving partitioned component $\mathbf{s}_i$ independently away from others. The logical components of an object $\mathbf{o}$ may mean different properties of the object, such as (i) the bounding volume of $\mathbf{o}$ (e.g., in direct volume rendering [16]), (ii) all points of a specific iso-value or having met a specific functional specification $G(p) = 0, p \in \mathbf{o}$ (e.g., in iso-surface [17] and interval volume extraction [20]), (iii) all non-transparent points in $\mathbf{o}$ (e.g., in constructive volume geometry (CVG) [6]), (iv) the base geometry, or the soft region of $\mathbf{o}$ (e.g., in hypertextures [21].

We organize our approaches to the specification of $\mathscr{S}$ and $\mathscr{T}$ into the following two sections, focusing on explicit and implicit splitting respectively. Within each section, we will address other technical issues wherever appropriate.

4 Explicit Splitting

A straightforward approach to splitting a spatial object $\mathbf{o}$ is to divide $\mathbf{o}$ into a set of independent component objects, $\mathscr{S} = \{\mathbf{s}_i | i = 1,2,\ldots,n\}$, that is,

$$\mathbf{o} = \boxed{\cup}(\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n).$$

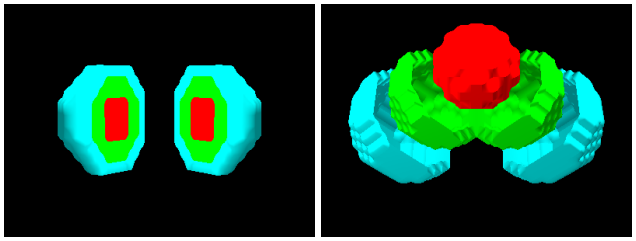
With a transformation set $\mathscr{T}$ where all functions at $t = 0$ are unit transformations, i.e., $T_{i,0}(\mathbf{s}_i) = \mathbf{s}_i, i = 1,2,\ldots,n$, we have completed the specification of sets $\mathscr{S}$ and $\mathscr{T}$ that meet the definition given in Section 3. However, unlike surface or solid objects that have a homogeneous interior, the construction of the component set $\mathscr{S}$ is not as straightforward as one would expect. We discuss this issue by considering discrete spatial objects and procedural spatial objects separately in the following two subsections.

4.1 Discrete Spatial Objects

Objects defined upon discrete volume data sets are spatial objects. Typically, the basic geometry of such an object is bounded by a cuboid in $\mathbb{E}^3$, within which attribute values are specified at a finite number of voxels that are commonly organized into a 3D grid. An interpolation function is normally used to transform a set of discrete data points to a continuous scalar field.

A simple way to split such a volume is to divide the voxels into a set of non-overlapping cuboids. For example, this method was used to create an animation of visible man in [4], where the body was divided up into about 26 different sized cuboids based upon the volumetric skeleton determined [11] and the distance field associated with each bone of the skeleton. Figure 3(a) illustrates this approach with a low resolution data set of 12x12x12 voxels, which is purposely constructed to make individual voxels identifiable. The data set contains three interconnected layers of non-zero voxels, shown in blue, green and red respectively.

Although *geometrical splitting* is useful in many applications, *logical splitting*, based on the semantic partitions in a volume, can offer intuitive visualization which highlights important areas of investigation. To achieve a logical splitting, one can explicitly construct separate objects for individual semantic partitions, and apply appropriate spatial transformations to them. For example, with the simple data set shown in Figure 3, we can construct three separate objects, each specifies a segment based on voxel values. This allows us to manipulate three segments independently as shown in Figure 3(b).

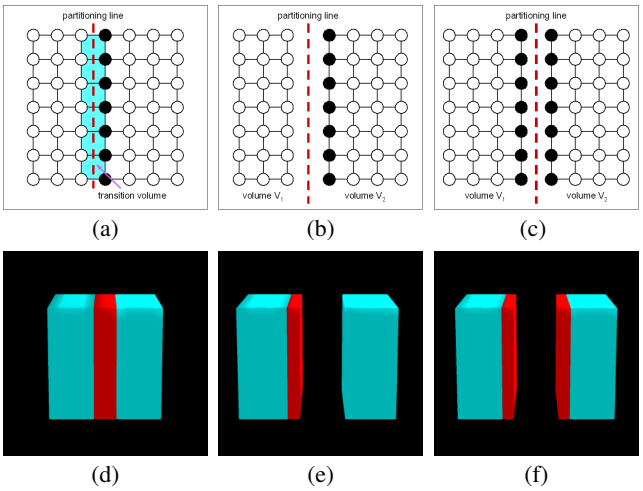


(a) a geometrical splitting (b) a logical splitting

Figure 3: Splitting a volume of 12x12x12 voxels by partitioning the original data set explicitly into two components which are re-jointed together using a volume scene graph.

One imperative problem is that simply distributing voxels among different components, $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n$, is not sufficient to satisfy the definition in Section 3 in terms of the equivalence condition. As illustrated in Figures 4(a) the regions between component objects, which was defined and (b), in the original object, $\mathbf{o}$, become undefined in $\boxed{\cup}(\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n)$. Figures 4(d) and (e) shows the splitting of a purposely constructed data set of 11x11x11 voxels into two objects, of 6x11x11 and 5x11x11 voxels respectively. As the original data set contains one middle y-z slice (the 6th slice, shown in red) that has different voxel values from the rest of the volume. The transition volume between the 6th and 7th slices have been lost during the splitting.

Hence, it is necessary to replicate an interpolation region along the boundary of division among all relevant component objects. Figure 4(c) shows a simple scheme suitable for volume data sets defined with tri-linear interpolation, and Figure 4(f) shows a correct splitting along the 6th slice. For logical splitting based on semantic partitions, a similar duplication scheme according to the interpolation method can also ensure the correctness of splitting actions.



(a) (b) (c)

(d) (e) (f)

Figure 4: Splitting a volume of 12x12x12 voxels by partitioning the original data set explicitly into two components which are re-jointed together using a volume scene graph.

4.2 Procedural Spatial Objects

Procedurally defined spatial objects do not suffer from the problem associated with volume data sets. We can easily construct the procedural specifications of n component objects, the union of which at $t = 0$ is equivalent to the original object. Once it is split, arbitrary spatial transformations can be applied to its components independently.

However, this simple approach cannot be applied to some common but complex spatial objects such as hypertexture objects. A *hypertexture* [21] is a spatial object defined upon a *object density function*, $D(x)$, which is a 3D distance field – a type of scalar fields commonly used in volume graphics [23, 2, 24]. For every point p in three-dimensional space $\mathbb{E}^3$ (or a bounded domain), $D(p)$ indicates whether or not p resides in a "soft" boundary region of an object. If p is in the soft region, $D(p)$ gives a distance from p to a surface representing the core of the object; and if p is not in the region, $D(p)$ indicates if p is inside the core, or outside the entire object, that is, the union of the soft region and the core. Combined with a *noise function* $D_N(p)$, a *turbulence function* $D_T(p)$, a *bias function* $D_B(v)$, and a *gain function* $D_G(v)$, where $p \in \mathbb{E}^3$ and $v \in \mathbb{R}$, we

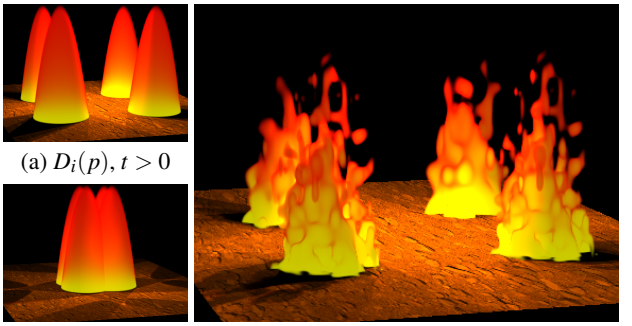
have a hypertexture function:

$$H(p) = \Theta(D(p), D_N(p), D_T(p), D_B(D(p)), D_G(D(p))).$$

From $H(p)$, with appropriate opacity and colour transfer functions, we can obtain a spatial object:

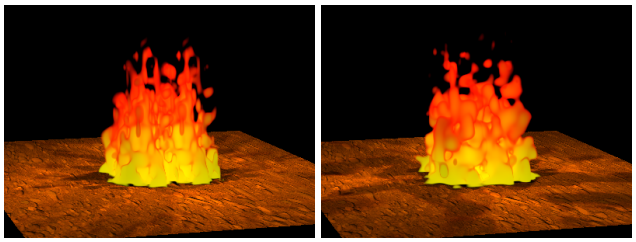
$$\mathbf{o} = (A_0(H(p)), A_1(H(p)), \ldots, A_k(H(p))).$$

Using the strategy of explicit splitting, we can construct a hypetexture to be partitioned by defining its n components separately using separate object density functions, $D_1(p), D_2(p), \ldots, D_n(p)$, or in a more general form, using separate hypertexture functions, $H_1(p), H_2(p), \ldots, H_n(p)$. In Figure 5, (a) shows four separate soft regions defined by a similar density function placed at different locations, and (b) shows four fire objects built from $D_1(p), D_2(p), D_3(p), D_4(p)$ by applying appropriate hypertexture functions and transfer functions.



(a) $D_i(p), t > 0$

(c) union, $t = 0$ (b) $\mathbf{s}_i, i = 1,2,3,4, t > 0$



(d) union after, $t = 0$ (e) union first, $t = 0$

Figure 5: A hypertexture of a fire is specified with four separate soft regions, and a splitting action is defined as the reverse of a merging action. (a) $D_i(p), i = 1,2,3,4$, at $t = 12$, (b) $\mathbf{s}_i, i = 1,2,3,4$, at $t = 12$, (c) $\mathbf{max}(D_i(p) | i = 1,2,3,4)$, (d) Applying a union operation to $\mathbf{s}_i, i = 1,2,3,4$, at $t = 0$, (e) Applying a **max** operation to $D_i(p), i = 1,2,3,4$, at $t = 0$, prior to the application of hypertexture and transfer functions.

Let us define the original object $\mathbf{o}$ as the union of its pre-partitioned components $\boxed{\cup}(\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n)$. This may seem to be a fudge, but it satisfies the definition in Section 3, and in principle, it uses the exactly the same approach as employed for Figures 3. The only feeling of inadequacy is that we did not actually start with a single $D(x)$ as a reference, though there was nothing to prevent us to do so. In fact, we will base our discussions on a single $D(x)$ when discussing implicit splitting methods for hypertextures in Section 5.2. Nevertheless, for explicit splitting of hypertexture objects such as fires, this predefined reference is unnecessary and does not bring in any extra technical issue. Hence, we focus our discussions on an original object defined as the union of a set of components.

Consider a set of overlapped density functions, $D_1(p), D_2(p), \ldots, D_n(p)$ at $t = 0$ (Figure 5(c)). We can simply treat them as individual soft regions, then apply appropriate hypertexture functions and transfer functions to create individual fires, and finally use the union operator given in Section 3 to construct a composite object as shown in Figure 5(d).

Alternatively, we may first combine the set of density functions, for example, using a **MAX** field operator over $D_1(p), D_2(p), \ldots, D_n(p)$ as they are in effect a set of scalar fields. We can then apply a single set of hypertexture functions and transfer functions to the combined density function (Figure 5(e)).

Comparing (d) and (e) in Figures 5, we can observe more vibrancy in (d), where the noise and turbulence of different objects help create a more realistic fire. This additional dynamism is more apparent in animation. The main advantage of applying the union operation after hypertexture and transfer functions is the extra degree of freedom in specifying noise, turbulence, bias and gain functions which do not need to maintain the uniformity over the combined soft region. This suits particularly well the dynamic and stochastic nature of most hypertexture objects. The union operator $\boxed{\cup}$ used in this work places a strong emphasis on the opacity, which helps effectively the visual integration of multiple fire objects. Hence, the second approach of combining density functions prior to the application of hypertexture and transfer functions does not bring in any extra benefit, while incurring an extra degree of complexity in the implementation.

5 Implicit Splitting

Given a spatial object $\mathbf{o}$ built upon either a discrete volume data or a procedurally defined scalar field, we can split $\mathbf{o}$ implicitly by introducing a set of "mobile" component objects, $\mathscr{S} = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n\}$, each of which is mapped onto a portion of the original object. Such a "mobile" object $\mathbf{s}_i$ can be defined using a *spatial transfer object* $\mathbf{t}_i$, and a generic *spatial transfer operator* $\boxed{\mapsto}$, as

$$\mathbf{s}_i = \boxed{\mapsto}(\mathbf{t}_i, \mathbf{o}).$$

A *spatial transfer object* $\mathbf{t}$, which is of a specific type of spatial objects, consists of three attribute fields, $A_x(p), A_y(p), A_z(p)$. Like in conventional spatial objects, these attribute fields can be defined using mathematical, procedural and discrete specifications [7]. Hence an object $\mathbf{t}$ defines a spatial mapping $\Psi : \mathbb{E}^3 \to \mathbb{E}^3$ for every point p in $\mathbb{E}^3$. Given an arbitrary p in $\mathbb{E}^3$, the three scalar fields, $A_x(p), A_y(p), A_z(p)$ collectively define the coordinates of another point q:

$$q = [q_x, q_y, q_z] = [A_x(p), A_y(p), A_z(p)] = \Psi(p).$$

Given an arbitrary scalar field $A(p)$, we can transform it into a different field as

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

and given an arbitrary spatial object $\mathbf{o}$, we can transform it into a different spatial object as

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

Hence, an evaluation of A' (or $\mathbf{o}'$) at p implies the evaluation of A (or $\mathbf{o}$) 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. It can be used to map a component object $\mathbf{s}_i$ to parts of the original object $\mathbf{o}$. It can also be used to modify the geometrical shape and position of the mapped partition of $\mathbf{o}$, hence serving as a specification for a transformation function $T_{i,t}$, which is essential to a splitting action.

In this section, we focus on the use of spatial transfer operations for specifying the implicit splitting of a spatial object. We will discuss its use for specifying spatio-temporal transformation functions, $\mathscr{T} = \{T_{i,t} | i = 1,2,\ldots,n; t = 0,1,\ldots,t_{max}\}$, in Section 6.

Similarly, we discuss the splitting using spatial transfer functions by considering discrete and procedural spatial objects separately in the following two subsections.

5.1 Discrete Spatial Objects

A spatial object built from one or more discrete volume data sets can be split implicitly by defining one or more spatial transfer objects. For example, a simple cuboid split, similar to Figure 3, can be achieved by a volume scene graph in vlib [26], which can be represented algebraically as:

$$\boxed{\cup}(\boxed{\mapsto}(\mathbf{t}_1,\mathbf{o}),\boxed{\mapsto}(\mathbf{t}_2,\mathbf{o})),$$

where $\mathbf{o}$ is the original object built upon a 12x12x12 volume, and $\mathbf{t}_1$ and $\mathbf{t}_2$ are two "mobile" objects that spatially transform two parts of $\mathbf{o}$ to achieve an opening effect. In fact, more conveniently, this splitting action can be realized using just one spatial transfer object, in the form of $\boxed{\mapsto}(\mathbf{t},\mathbf{o})$, where $\mathbf{t}$ defines a discontinuous transformation moving each half of $\mathbf{o}$ in a different direction. Figure 6 shows two splitting actions applied to a discrete volume data set using different spatial transfer functions.

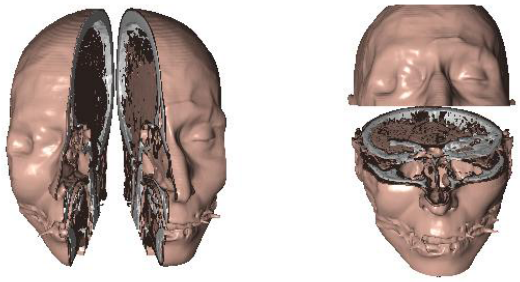


Figure 6: Two simple spatial transfer functions are applied to a CT data set.

The use of spatial transfer functions does not suffer from the problem of missing transition volume as discussed in 4.1. This is because the entire data set of $\mathbf{o}$ is available to any spatial transfer object $\mathbf{t}$ if they are coupled in a volume scene graph as $\boxed{\mapsto}(\mathbf{t},\mathbf{o})$, even when $\mathbf{t}$ defines only a partial mapping of $\mathbf{o}$. The problem of missing a transition volume between slices, or missing data for a slightly demanding interpolation scheme (i.e., other than nearest neighbor) disappears naturally.

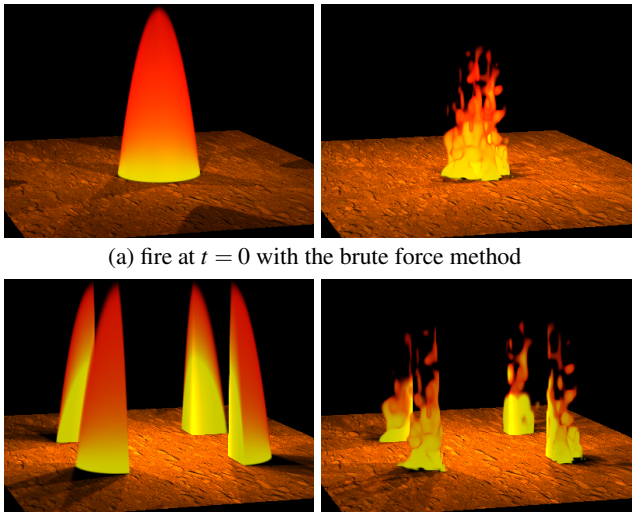
Spatial transfer objects can also be used for logical splitting, though directly defining such an object could be rather challenging. However, one approach is to use masking objects which can be specified either procedurally or with masking volume data sets commonly available in segmentation processes. For example, to achieve the splitting shown in Figure 3(b), we can simply define three masking objects, $\mathbf{s}_1,\mathbf{s}_2,\mathbf{s}_3$, each with an opacity field representing one of the semantic layers in $\mathbf{o}$. The splitting action is thus defined by a volume scene graph:

$$\boxed{\cup}(\boxed{\mapsto}(\mathbf{t}_1,\boxed{\cap}(\mathbf{o},\mathbf{s}_1)),\boxed{\mapsto}(\mathbf{t}_2,\boxed{\cap}(\mathbf{o},\mathbf{s}_2)),\boxed{\mapsto}(\mathbf{t}_3,\boxed{\cap}(\mathbf{o},\mathbf{s}_3))),$$

where $\boxed{\cap}$ is an intersection operator.

5.2 Procedural Spatial Objects

With the support of a volume scene graph in software such as vlib, the splitting of procedurally specified spatial objects demands much less effort from the users. For example, using four simple spatial transfer objects (or a combined one), we can easily split a fire as shown in Figure 7.

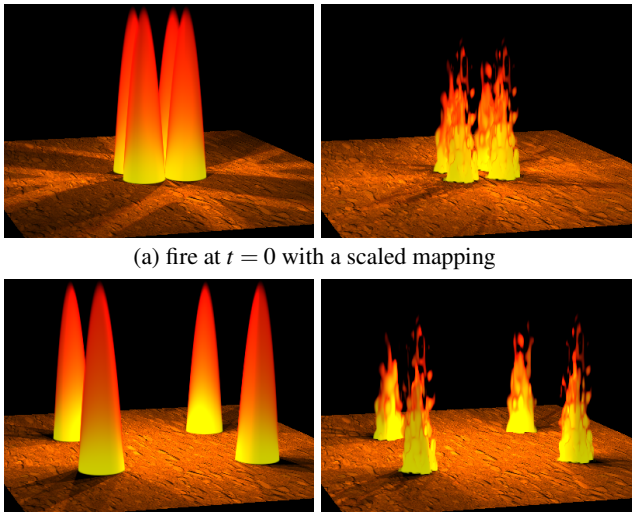


(a) fire at $t = 0$ with the brute force method

(b) fire at $t > 0$ with the brute force method

Figure 7: A hypertexture of a fire is specified with a soft object, and a splitting action is defined using spatial transfer functions that partition the fire implicitly.

However, noticeably, the splits are far too "perfect" for amorphous objects such as a fire. It is hence desirable to maintain some geometrical characteristics of the original soft region. Instead of mapping each component object, $\mathbf{s}_i = \boxed{\mapsto}(\mathbf{t}_i,\mathbf{o})$, to a quarter of the spatial domain of $\mathbf{o}$, we can map $\mathbf{s}_i$ to the entire $\mathbf{o}$, while maintain the size of the corresponding spatial transfer object $\mathbf{t}_i$. Note that it is the size of $\mathbf{t}_i$ that determines the size of $\mathbf{s}_i$. This in effect facilitates a scaling transformation from $\mathbf{o}$ to $\mathbf{s}_i$. Figure 8 shows a much realistic set of fires at $t > 0$ in comparison with Figure 7.

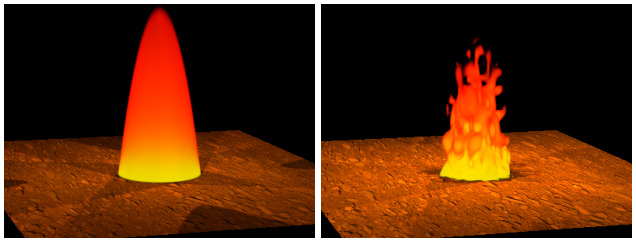


(a) fire at $t = 0$ with a scaled mapping

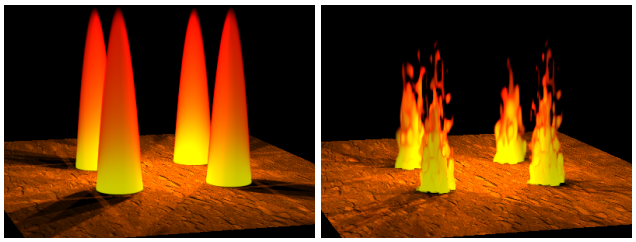
(b) fire at $t > 0$ with a scaled mapping

Figure 8: Instead of mapping each component to a quarter of the original $\mathbf{o}$, each spatial transfer object $\mathbf{t}_1$ is mapped onto the entire domain of $\mathbf{o}$, while maintaining its quarterly spatial occupancy.

Recall our previous discussions on a reference object at $t = 0$ in 4.2, and we do have a desirable reference object as a single fire in this case. Based on this reference object, the fire at $t = 0$ in Figure 8 does not satisfy the equivalence condition as outlined in Section 3.



(a) fire at $t = 0$ with a resizable mapping



(b) fire at $t > 0$ with a resizable mapping

Figure 9: Each spatial transfer object $\mathbf{t}_1$ is mapped onto the entire domain of $\mathbf{o}$, and changes its spatial occupancy from the same as $\mathbf{o}$ at $t = 0$ to its quarterly spatial occupancy at a specific $t > 0$.

The flexibility in the specification of spatial transfer objects can again help resolve this problem. We can dynamically change the size of each spatial transfer object, $\mathbf{t}_i$, as part of spatial and temporal transformation $T_{i,t}$. At $t = 0$, $\mathbf{t}_i$ not only maps $\mathbf{s}_i$ to the entire spatial domain of $\mathbf{o}$, but also occupies the exact same space as $\mathbf{o}$. During a splitting action over time, each spatial transfer object $\mathbf{t}_i$ (hence $\mathbf{s}_i$ as well) resize itself, gradually, to a desired size at a specific time $t > 0$. Figure 9 shows a satisfactory transformation from $t = 0$ to $t > 0$.

At $t = 0$, there are multiple component objects occupying the same space. As the union operator used in this work is based on the MAX field operator (see Section 3), this ensures that the union of these component objects is equivalent to $\mathbf{t}$. This is not always guaranteed with other union operators used in the field of volume graphics and implicit surface as many such operators do not satisfy the identity law.

6 Results

In this section, we describe the implementation of four reasonably complex splitting actions, and report briefly our study on the scalability of a complex splitting action. Our first case study is partially inspired by the special effects in film AI as shown in Figure 1. We combined the explicit splitting approach outlined in 4.1 with the implicit approach outlined in 5.1 to achieve an animation that opens the facial skin in a CT data set. Figure 10(a) shows some frames extracted from two animation sequences of the same splitting action.

With the explicit splitting approach, the CT data set is segmented into two semantic layers, namely the bones and the rest of soft tissues, which are represented by two separate spatial objects. The opening of the facial skin is achieved using the implicit approach as it is relatively easy to specify geometrical splitting using spatial transfer objects once a logical splitting is defined. The specification of this splitting can easily be extended to an object containing many semantic layers.

Our second example is shown in Figure 10(b), where the bones of a visible man's foot is segmented and slips out the skin. Again, we considered two semantic layers. Our third example, shown in Figure 10(c), is simulation of fuel injection into the combustion chamber. It involves two semantic layers. Fourth example is shown

Figure 11: Two semantic layers of a volume teapot model.

in Figure 11,where a teapot is segmented into three semantic layers, namely exterior of the teapot, interior of the teapot and a lobster. Each of these are represented by a spatial object.

To examine the scalability of the approach presented, we attempted a much more complex splitting action, which simulates an explosion effect with both discrete volume objects (i.e., the visible man data set) and procedural volume objects (i.e., fires). We utilize a particle system to define the trajectories of the component objects. For each component object $\mathbf{s}_i$, we determine its motion path as:

$$x_{i,t} = v_i \cos^2(\theta_i)$$
$$y_{i,t} = v_i \sin(\theta_i)t + 0.5gt^2$$
$$z_{i,t} = v_i \sin(\theta_i)\cos(\theta_i)$$

where g is the gravity constant. v_i is the velocity of each component which is randomly chosen from $[1,90]$ at the beginning. θ_i is the initial angle of motion, which is again randomly chosen from $[30,89]$ degree.

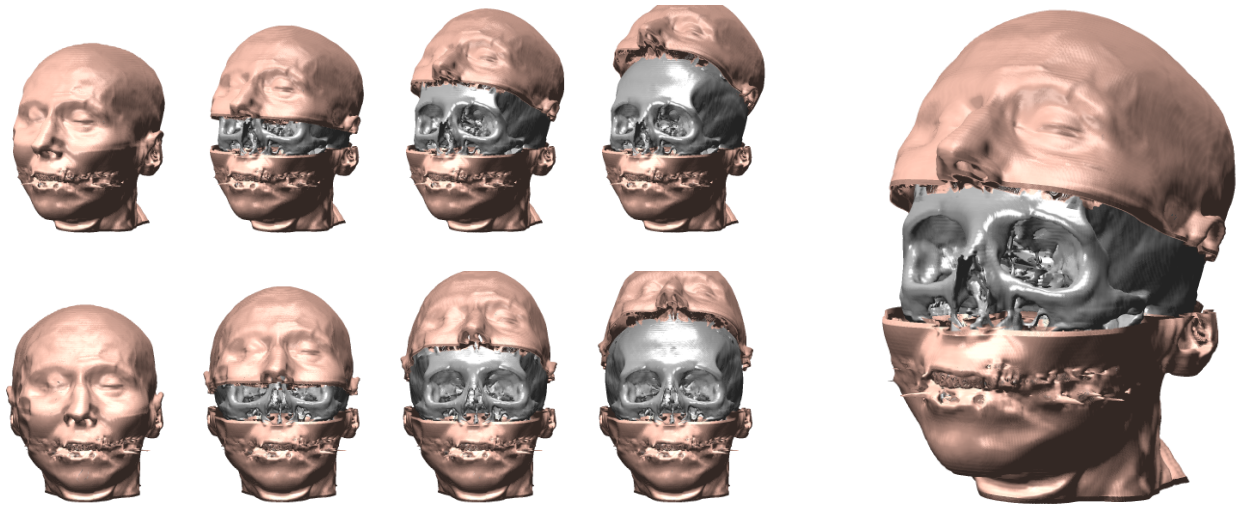
For the splitting of the visible man data set, we employed the implicit approach in 5.1 directly, with which each component is mapped onto a portion of the original object using a spatial transfer object. The fire components are specified using the explicit approach in 4.2, which gives more flexibility to associate each component of the visible man with a fire component.

All components are combined into a volume scene graph, where geometrical transformation is a function of time, and which is rendered using vlib [26]. Figure 12(a) shows a purposely constructed frame with two visible human data sets, one rendered as usual, and the other split into 32 pieces which are spatially transfered, and combined with hypertexture fire also split into 32 pieces. Figure 12(b) shows frames extracted from three separate sequences of splitting the visible human data sets with 8, 32, and 128 splits respectively.The timing shown in the figure indicates that the rendering time relates to the number of splits almost linearly.

Most of the images in this paper were rendered on a 128-node PC cluster without any graphics hardware. Depending the complexity of the volume scene graphs and image resolution, the majority of the images, except the visible man animation, take between 1 - 30 seconds to render. For simple scene, the most time is in fact the communication overhead in the parallel processing.

7 Conclusions

In this paper, we have presented a comprehensive study on a particular graphical operation, namely *splitting*, in the context of volume visualization, volume animation and special effects. We have



(a) digitally splitting a CT head based on two semantic layers, namely bones and soft tissues.

(b) digitally extracting bones from a visible man's foot.

(c) digitally opening a combustion chamber.

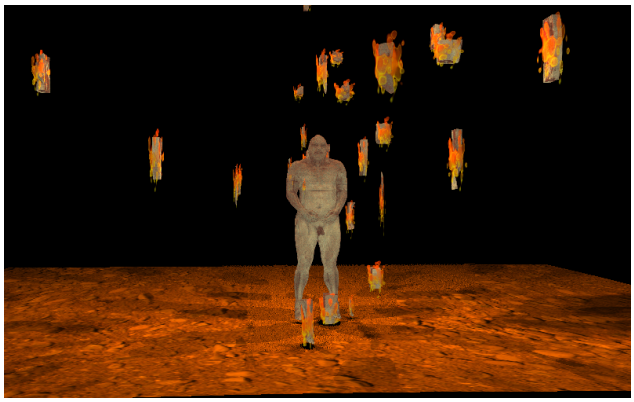
Figure 10: Three sets of frames selected from different example animation sequences.

described two main approaches to realize splitting actions in a volume graphics pipeline. We have introduced a definition of "correct" splitting, which provides basic guidance to the evaluation of different splitting methods as well as enables us to consider both objects constructed from discrete volume data sets and those defined procedurally, often in a continuous and amorphous manner.

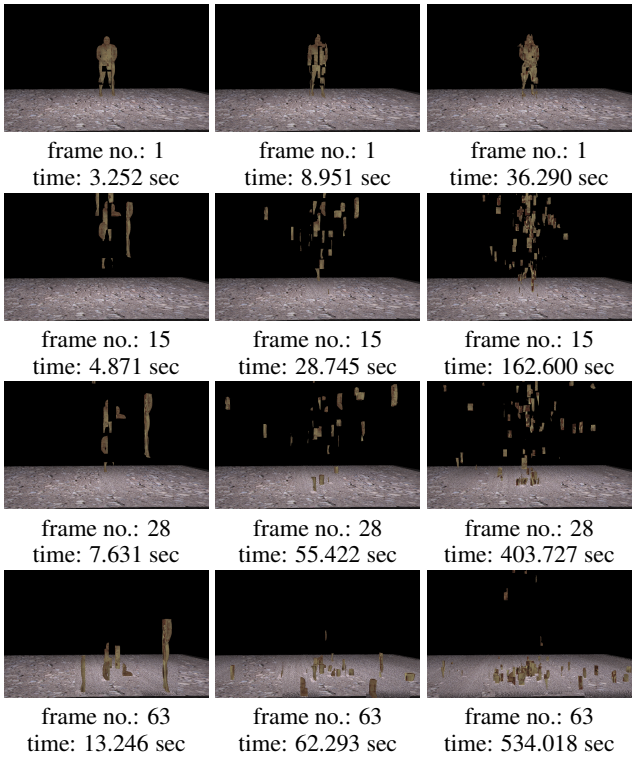
We have shown that splitting can be specified explicitly as well as implicitly. The latter demands less effort from users in the specification process, and does not suffers from the issue of missing boundary data as with the former in dealing with discrete data sets. We have also shown that both explicit and implicit splitting can effectively facilitate geometrical splitting as well as logical splitting. The integration of spatial transfer functions into volume scene graphs allows us to specify complex splitting in a scalable manner. We have also considered the splitting of hypertexture as an example of complex volumetric objects. We have found operative methods with both explicit and implicit approaches. To support our findings, we have provided a set of high quality results which have demonstrated the correctness, effectiveness, and scalability of the approaches presented in this paper.

ACKNOWLEDGMENTS

Authors gratefully acknowledge the individual support from different funding organizations, including a research grant to D. Silver, a UK EPSRC grant to M. Chen, and a UK EPSRC PhD studentship to S. Islam. 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;Foot from Philips Research, Hamburg, Germany, and Teapot with lobster from Terarecon Inc, MERL, Brigham and Women's Hospital.



(a) A split visible man and fire with the original visible man.



(b) from left to right: 8, 32 and 128 splits.

Figure 12: (a) Splitting the visible man data set with a fire hypertexture, together with the original visible man in the center. (b) Scalability test with varying number of splits.

REFERENCES

[1] A. Barr. Global and local deformation of solid primitives. *Computer Graphics (Proc. SIGGRAPH 84)*, 18(3):21–30, July 1984.

[2] D. E. Breen, Sean Mauch, and R. T. Whitaker. 3d scan-conversion of CSG models into distance, closest-point and colour volumes. In M. Chen, A. E. Kaufman, and R. Yagel, editors, *Volume Graphics*, pages 135–158. Springer, London, 2000.

[3] M. S. T. Carpendale, D. J. Cowperthwaite, and F. D. Fracchia. Extending distortion viewing from 2D to 3D. *IEEE Computer Graphics and Applications*, 17(4):42–51, 1997.

[4] M. Chen, D. Silver, A. S. Winter, V. Singh, and N. Cornea. Spatial transfer functions – a unified approach to specifying deformation in volume modeling and animation. In *Proc. Volume Graphics 2003*, pages 35–44. Tokyo, Japan, Eurographics/ACM Publications, 2003.

[5] M. Chen, J. V. Tucker, R. H. Clayton, and A. V. Holden. Constructive volume geometry applied to visualisation of cardiac anatomy and electrophysiology. *International Journal of Bifurcation and Chaos*, 13(12):3591–3604, 2003.

[6] M. Chen and J.V. Tucker. Constructive volume geometry. *Computer Graphics Forum*, 19(4):281–293, 2000.

[7] M. Chen, A. S. Winter, D. Rodgman, and S. M. F. Treavett. Enriching volume modelling with scalar fields. In F. Post, G.-P. Bonneau, and G. Nielson, editors, *Data Visualization: The State of the Art*, pages 345–362. Kluwer Academic Press, 2002.

[8] S. Frisken-Gibson. Using linked volumes to model object collisions, deformation, cutting, carving, and joining. *IEEE Transactions on Visualization and Computer Graphics*, 5(4), 1996.

[9] N. Gagvani and D. Silver. Parameter controlled volume thinning. *Graphical Models and Image Processing*, 61(3):149–164, 1999.

[10] N. Gagvani and D. Silver. Animating the visible human dataset. In *Proc. the Visible Human Project Conference*, 2000.

[11] N. Gagvani and D. Silver. Animating volumetric models. *Graphical Models*, 63(6):443–458, 2001.

[12] S. Gibson and Mirtich. A survey of deformable modeling in computer graphics. Technical Report TR97-19, MERL Technical Report, 1997.

[13] M. Hadwiger, C. Berger, and H. Hauser. High-quality two level volume rendering of segmented data sets on consumer graphics hardware. In *Proc. IEEE Visualization 2003*, pages 301–308. Seattle, WA, 2003.

[14] Y. Kurzion and R. Yagel. Space deformation using ray deflectors. In *Proc. the 6th Eurographics Workshop on Rendering*, pages 21–32, Dublin, Ireland, June 1995.

[15] E. LaMar, B. Hamann, and K. I. Joy. A magnification lens for interactive volume visualization. In *Proc. IEEE Pacific Conference on Computer Graphics and Applications*, pages 223–231, 2001.

[16] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(5):29–37, 1988.

[17] W. Lorensen and H. Cline. Marching cubes: a high resolution 3d surface construction algorithm. *ACM/SIGGRAPH Computer Graphics*, 21(4):163–169, 1987.

[18] M. J. McGuffin, L. Tancau, and R. Balakrishnan. Using deformations for browsing volumetric data. In *Proc. IEEE Visualization 2003*, pages 401–408. Seattle, WA, 2003.

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

[20] G. M. Nielson and J. Sung. Interval volume tetrahedrization. In *Proc. IEEE Visualization 1997*, pages 221–228. Phoenix, AZ, 1997.

[21] K. Perlin and E. Hoffert. Hypertexture. *ACM/SIGGRAPH Computer Graphics*, 23(3):57–64, 1989.

[22] S. Roettger, S. Gutthe, D. Weiskopf, T. Ertl, and W. Strasser. Smart hardware-accelerated volume rendering. In *Eurographics/IEEE Symposium on Visualization 2003*, pages 231–301. Seattle, WA, 2003.

[23] R. Satherley and M. W. Jones. Extending hypertextures to non-geometrically definable volume data. In M. Chen, A. E. Kaufman, and R. Yagel, editors, *Volume Graphics*, pages 211–225. Springer, London, 2000.

[24] M. Sramek and A. E. Kaufman. vxt: A class library for object voxelisation. In M. Chen, A. E. Kaufman, and R. Yagel, editors, *Volume Graphics*, pages 119–134. Springer, London, 2000.

[25] M. Tory and C. Swindells. Comparing ExoVis, orientation icon, and in-place 3d visualization techniques. In *Proc. Graphics Interface*, pages 57–64, 2003.

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

[27] Y. Zhao, X. Xie, Z. Fan, A. Kaufman, and H. Qin. Voxels on fire. In *Proc. IEEE Visualization 2003*, pages 271–278. Seattle, WA, 2003.