Volume Wires : A Framework for Empirical Non-linear Deformation of Volumetric Datasets

S.J. Walton and M.W. Jones
Swansea University
cssimon@swansea.ac.uk m.w.jones@swansea.ac.uk

ABSTRACT

We introduce a new framework for non-linear, non-reconstructive deformation of volumetric datasets. Traditional techniques for deforming volumetric datasets non-linearly usually involve a reconstruction stage, where a new deformed volume is reconstructed and then sent to the renderer. Our intuitive sweep-based technique avoids the drawbacks of reconstruction by creating a small attribute field which defines the deformation, and then sending it with the original volume dataset to the rendering stage. This paper also introduces acceleration techniques aimed at giving interactive control of deformation in future implementations.

Keywords: Volume rendering, Volume deformation, Swept volumes, Curves, Volume Animation, Nonlinear deformation, Attribute distance field

1 INTRODUCTION

Research in the area of volume graphics is mainly concentrated on visualisation techniques. Tools and API's for volume modeling [SK00] and visualisation [WC01] exist, but there is a lack of tools and techniques for interactively manipulating these datasets. For surface-based graphics, a huge variety of tools exist (such as Maya and Character Studio) for the manipulation and rendering of such objects. It would be beneficial to the volume graphics community to bring some of the concepts of such powerful animation tools to working with volume datasets.

Volumetric deformation techniques have been recently documented in the literature [CCI$^+$05]. Deforming volumetric datasets is viewed as a more complex problem than surface-based deformation due to the size of the data. Even if one extracts a subset of this data (a *volume object*) with segmentation techniques [Lak00], the number of voxels to be deformed is still a limiting factor. Some approaches rely on either converting to an intermediate representation (using marching cubes to convert to a mesh structure) and then deforming that representation, or reconstructing (voxelising) a newly deformed volume dataset to be passed to the rendering stage.

This paper introduces a new software-based method to deform a volumetric dataset non-linearly without converting to a mesh geometry or using expensive volume reconstruction techniques. Our work concentrates on empirical deformation with the aim of producing a simple to use volume deformation and animation tool.

2 RELATED WORK

We split the related work into two logical areas - volume deformation and swept volumes.

2.1 Volume deformation

Spatial Transfer Functions [CSW$^+$03] were introduced by Chen *et al* . They define a framework for specifying spatial transformation and deformation for volume objects. A spatial transfer function defines the geometrical transformation of every point in the volume. Typically, a backward-mapping operation must be performed (the inverse of the deforming function) to find out where to sample in the dataset based on the current sample point on the ray. Depending on the complexity of the function, the computational cost can be high.

Similar non-reconstructive approaches involve placing ray deflectors in the scene [KY95] which deform the ray as it passes through the volume, but its use is rather limited, and specifying the deflectors is typically unintuitive as the user must think in terms of the reverse effect. Hardware-accelerated methods that work with isosurfaces exist such as in [WRS01], however, specifying the deformations is still unintuitive for the user, and isosurface property restrictions exist. Other techniques such as the 3D chainmail algorithm [Gib97] rely on moving the individual voxels and then splatting the newly-positioned voxels to the screen [Wes90]. These methods still (e.g. for animation purposes) do not allow for intuitive deformation on a large scale from the perspective of the user.

More recent work by Gagvani [GS01] has allowed for the widely-used IK-skeleton deformation methods to be utilised in volume graphics, whereby an entire new volume is reconstructed and then rendered. The algorithm is costly when the size of the dataset is large (for example, the visible human), as for the case of an animation, a new dataset must be created for each frame. A small animation can easily run over 50GB when stored on disk.

Prakash and Wu [WP99] animated the visible human using Finite Element Methods and clustering for segmenting the dataset into blocks. A hardware accelerated manipulation system called VolEdit [SSC03] allows the user to interactively manipulate the IK-skeleton and see the results in real-time. Since the transformations are linear, cracks can appear at joint areas. The VolEdit system solves this problem using mid-plane geometry. Part of the motivation for our work in this paper has been to solve this problem with a software based, non-reconstructive method.

2.2 Swept volumes

A *swept object* is produced when some template is swept along a trajectory through space. The template to be swept can be a static template such as a 2D image, or a dynamic template that changes through the sweep. Complex swept objects can be achieved by scaling [BvNP89] or rotating the template as it is swept. If the template varies as with slices through an axis of a volume dataset, then the result is a volume dataset swept along a new trajectory.

Much work has been published on swept volumes with an excellent review of techniques given in [AMBJ00]. The amount of work published is a reflection of the difficulty of some of the associated problems with sweeping techniques – in particular, the problem of determining properties of a swept object such as its boundary and volume. Early work on swept solids by Kajiya [Kaj83], and Wijk [vW84] go into some detail on methods for ray-tracing swept solids defined with arbitrary paths. Sealy and Wyvil [Sea97] describe how to voxelise new volume objects by sweeping contours along a curve, which is achieved by recursively subdividing the curve.

In [WC02], 2D images are swept along a path defined by a Bézier curve to reconstruct a volume. The volume is rendered using direct volume rendering. The authors also discuss attempts to directly evaluate the resulting deformation without reconstructing a volume, but unfortunately such evaluation is expensive (since it involves using numerical root finding methods), restrictive, and problematic (e.g, singularity conditions on an axis where an image is swept around the axis).

A *swept volume* is produced when a swept object is voxelised [Sea97]. The new volume can then be rendered using any volume visualisation technique. The disadvantage of reconstructing a volume from a sweep is the space requirement – a new volume must be produced and either stored in memory or on disk. For an animation, this is multiplied by the number of frames if the user wishes to retain the intermediate data to re-render the animation at a later date, with perhaps new view parameters or lookup functions.

3 DISTANCE FIELDS AND ATTRIBUTE PROPAGATION

Since a distance field technique is required for our method, we present a brief overview. Distance fields [SJ01] have been widely used for a variety of applications in the volume domain, such as morphing [BW01], voxelisation [Jon96] [JS00], and skeletonisation [GS01]. A distance field dataset D representing a surface S is defined as $D : \mathbb{R}^3 \to \mathbb{R}$, and for $p \in \mathbb{R}^3$,

$$D(p) = min\{\mid p - q \mid : q \in S\} \qquad (1)$$

where $||$ is the Euclidean norm and q are the nearest points on the surface. Each voxel in the field contains a value that represents the minimum distance to the surface of interest in the data. In the case of volume data, we may be interested in a particular isosurface representing, for example, the bone surface in a medical dataset. We can sign the value depending on whether the voxel is inside the target surface - becoming a signed distance field. A fast method of computing this field is by using the distance transforms [SJ01] to propagate local distances.

It follows that if we can propagate the minimum distances to a surface in this way, any related attributes of the surface (e.g. colour, as in [BM99]) can also be propagated. These additional attributes can be stored at each voxel in the distance field. The field then becomes:

$$D(p) = (min\{\mid p - q \mid : q \in S\}, a_1, \ldots, a_n) \qquad (2)$$

where $a_1, \ldots, a_n$ are our additional attributes. If only the attributes are of interest then the distance value at each voxel may be discarded, thus saving typically 4 bytes per voxel if using floating-point precision. In our method, we discard the distance values as they are not needed in later stages.

4 METHOD DESCRIPTION

Our approach is based on the idea of sweeping a volume object along an arbitrarily defined path, although the approach may also be viewed from the standpoint that the deformed path has the effect of deforming the surrounding volume object. Because no reconstruction takes place, the deformation and rendering stages are closely coupled. Figure 1 gives a high-level overview of the system. The method is not limited to specific classes of curve or any other trajectory definitions, except for the requirement that it can be parametrically evaluated, satisfying the general form:

$$\alpha(t) = (\alpha_x(t), \alpha_y(t), \alpha_z(t)) \qquad (3)$$

The deformed dataset is evaluated at render time via backward mapping utilising an *attribute field*, and can be rendered easily with a standard ray-casting approach. Small modifications are required to the rendering engine which will be discussed in coming sections.

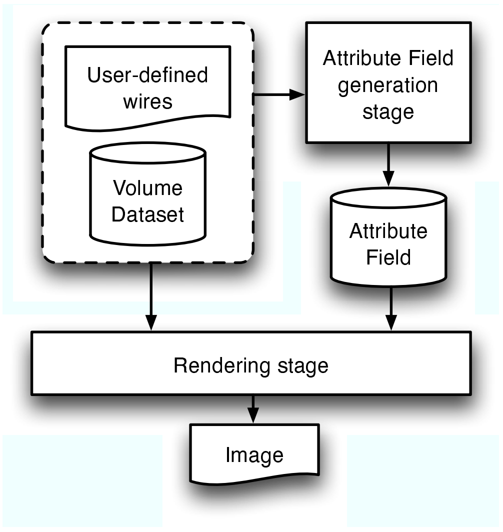


Figure 1: System overview

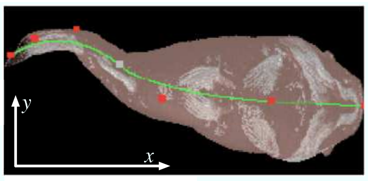
4.1 Specifying the deformation

From the user's point of view, the specification of the deformation is conceptually similar to that of wire deformers in Maya. We are therefore extending this surface-based deformation technique to work neatly with volumetric datasets. Such a transition is not trivial due to the entirely different data representation (discretely sampled vs. surface based). In addition, an important difference is that we are not deforming the data itself and sending it to the rendering stage.

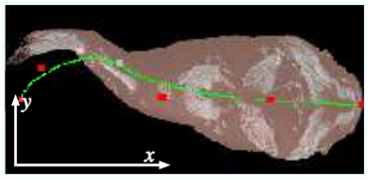
In our method, the user defines a *base wire* close to or inside the object to be deformed, and also an object classification function for the wire $\beta(p \in \mathbb{R}^3) \rightarrow [true, false]$ which determines the associated volume object. The wire is then transformed via translations, rotations, and curve deformation and this has the effect of deforming the volume object defined by β in the wire's specified region of influence.

The method permits scaling and rotation values at arbitrary points on the wire. For example, an angle of $\theta = 0$ at one end of the wire and $\theta = \pi$ at the other will produce the effect of the object being twisted along the path (linearly interpolating the θ values). The length of the base wire and the modified wire need not be equal, which allows for compression and expansion of the data along the trajectory of the wire.

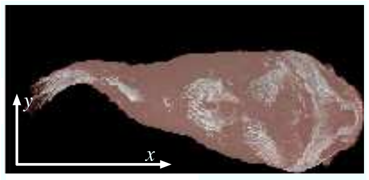
Figure 2(a) gives an overhead view of the user-defined base wire on the CT carp dataset. Figure 2(b) shows that the user has modified the wire, pulling one end of the wire in the negative y-axis direction. Finally, Figure 2(c) shows the resulting render from this deformation.



(a) The base wire



(b) Modifying the wire



(c) The resulting render

Figure 2: Fish deformation

In this example, the wires are Bézier curves. The base wire in this instance acts as the backbone of the carp. Deforming this backbone and then rendering the result would result in a new pose for the carp, as the surrounding soft tissue would be deformed around the backbone. The backbone could be derived semi-automatically using a distance field thinning technique [GS01] or watershed segmentation technique [Lak00].

5 BUILDING DEFORMATION DATA

In this stage, the deformation information from the wire-specification stage is encoded into an *attribute field*, which is then sent with the original volume object to the rendering stage. The attribute field is a volumetric dataset (γ, ε) where γ maps voxels to their corresponding wire and ε maps voxels to the t-value (see equation 3) of the closest point on that wire. For certain classes of curve (e.g. Catmull-Rom splines), the segment number also needs to be stored.

The attribute field need not be the same scale as the volume. In our research we have found that producing an attribute field of 1/8th size (half each dimension) produces results very close visually to the full size field. Additional considerations regarding reduced scale fields and example images are given in a later section.

For each base wire, we associate a set of planes P (see Figure 3) aligned with the trajectory of the wire. The initial size of the plane is governed by the region of influence for the wire, and the plane dimensions are automatically reduced to tightly fit the target object defined by classification function β. To generate the attribute

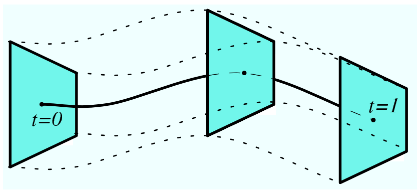


Figure 3: Planes defined along wire

field, an empty field is initialised over the domain of the union of each of the wires' region of influence. A mapping is now defined between planes on the base wire and planes on the modified wire, essentially the planes are copied based on their t-value. For each plane on the modified wire, we look at the attribute field voxels touched by the plane. If the plane touches a voxel, then a flag is set with that voxel.

The optimal number of planes can be calculated from an approximation of the wire's length. For parameterised curves, the length can be approximated with precision p by:

$$|\alpha| = \sum_{i=2}^{p} |\ \alpha(\frac{p}{i}) - \alpha(\frac{p}{i-1})\ | \qquad (4)$$

where $||$ is the Euclidean norm.

5.1 Voxel Initialisation

Each wire is now voxelised into the attribute field. When a new cell is entered by the wire, each of the eight surrounding voxels' γ (the wire reference) and ε (closest t-value) attributes are set, and also the distance d from the voxel to the point defined by ε, as shown in Figure 4. The closest point on the wire is calculated by subdividing the subset of the curve inside the voxel (as in equation 4), and this calculation can be achieved with parametrically-defined precision[1]. If a voxel has already been set in a previous cube (as with v_4), then the new and current minimum distances are compared and the minimum taken. This is denoted by the greyed-out vector in Figure 4 where $d' < d$.

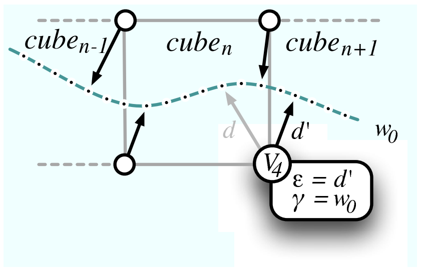


Figure 4: Pre-propagation voxel initialisation

[1] This is a fairly fast and accurate way to approximate the closest point on a curve. Spline implicitization [Sha03] or other methods [Sch90] could be used if more precision is required, at the expense of additional complexity.

Once this process is complete for all wires, the distances and associated attributes are propagated using a distance transform method, and the distance values are discarded. The propagation only takes place with voxels flagged in the previous step, so large areas of the field can be skipped. It is this propagation that removes the need for a costly backward evaluation at each voxel.

6 RENDERING THE DEFORMATION

To render the deformation, a standard ray-casting approach is followed, with rays cast into the attribute field instead of the volume object. We ignore all voxels in the field which have not been flagged. At each sample point p_{sample} on the ray, the wire reference w from the nearest voxel is noted. For the current sample point p_{sample}, the wire parameter value t is trilinearly interpolated from the eight surrounding voxels. The mapping achieved between the base wire and the deformed wire is illustrated in Figure 5. Given the wire reference w and t-value t, we can calculate the closest point on the modified wire and build a vector to it, becoming $p_{sample} \longrightarrow p_{curve}$. To obtain the actual sample point in the volume dataset from this, vector $p_{sample} \longrightarrow p_{curve}$ is mapped onto the base wire, becoming $p'_{sample} \longrightarrow p'_{curve}$ by using the t-value. This is demonstrated in Figure 5 where two sample points on ray_i and ray_j are mapped from the modified wire (a) to the base wire (b). p'_{sample} is now our new sample point in the dataset. The final

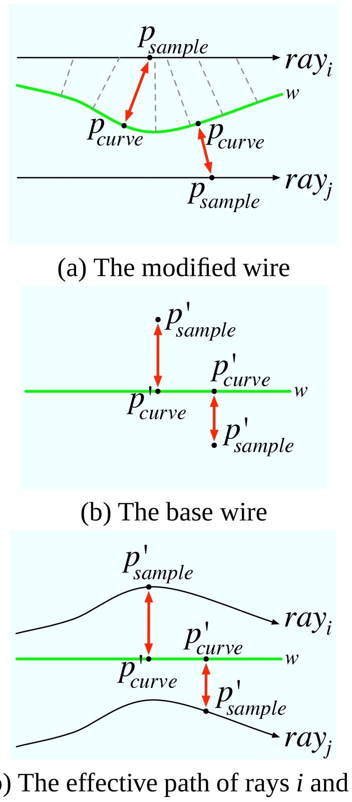


Figure 5: The mapping between wires

effect of ray_i and ray_j's trajectories being deformed is shown in (c).

If the attribute field has been scaled with respect to the volume object, then the density of sample points in the field must be modified accordingly. We also must deal with cases where a cube's eight vertices give different wire references, as in Figure 6. The interpolated t-value at p_{sample} would be inaccurate for either choice of wire (a or b). We have looked at fast methods for recovering a t-value (such as taking averages of the majority wire), but we have found that the decision at these voxels contributes little to the final image quality except with very low scale fields. In the resulting images (given later), we simply choose the wire and t-value at the closest voxel to p_{sample}.

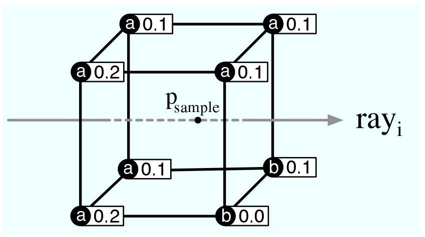


Figure 6: Differing wire reference problem

6.1 Calculating the new normals

Once the the new sample point has been calculated, a new normal at that point is required if we are to accurately light the deformed object. One way to achieve this would be to use central differences using backward-mapped points, but this is clearly an expensive operation. The new normal can be calculated efficiently as follows. First, we compute the normal n at the new sample point p'_{sample} obtained in the backward-mapping stage. This normal is calculated using central differences in the original volume dataset. To obtain a new normal n' for the deformed point, we calculate the difference in wire trajectory of the base wire point p_{curve}, and deformed wire point p'_{curve}. The normal n is then rotated by this amount to obtain n', and can be sent to a lighting equation.

7 OPTIMISATION AND THE DELINEATION PROBLEM

Problems may arise when the user wishes to deform two objects that are in close proximity – perhaps by pulling the two objects apart to separate them. We illustrate this problem in Figure 7. Figure 7(a) shows two objects x and y, and Figure 7(b) shows a slice of the objects (the slice is shown half-way down the objects in (a)). In this case, a plane of target object x's wire (shown as a dotted rectangle) has overlapped object y. Part of object y will therefore be included in the deformation of object x, since y is within x's plane. This is unlikely to be what the user would have intended in this case.

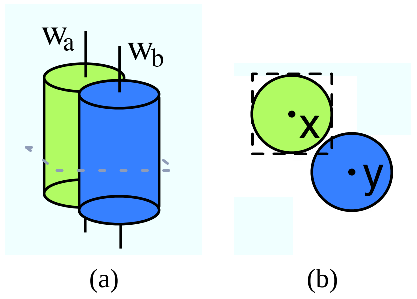


Figure 7: The delineation problem

If the user has defined multiple wires inside the volume dataset, it is likely that they wish to treat the dataset as a set of disjoint volume objects as defined by function β. It would be favourable for the system to be able to automatically delineate the objects in Figure 7 without the user resorting to volume segmentation methods [Lak00], which are typically very difficult to work with.

7.1 Plane masks

To solve this problem, *plane masks* are introduced. Once the planes are defined on the wire, a 2.5D seed fill[2] is performed on each of the planes to generate a 2D bit-mask, which is then stored with the plane.

Figure 8 shows a selection of these masks defined along the wire for the CT carp dataset, with an object classification function β set to identify the outer skin area with a simple value threshold. The resolution of the mask can be varied by parameter s, and the memory requirement for each wire in bytes is calculated as:

$$\sum_{p=1}^{n} \frac{(a(p) * s)}{8}$$

where n is the number of planes on the wire, a gives the area of the plane, and s is a resolution scale multiplier. Values of s below 1 give a sparse mask, values above give a fine mask and therefore greater precision, at the expense of a greater storage overhead and preprocessing time.

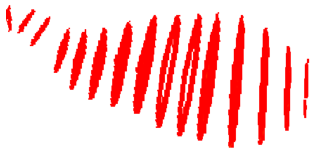


Figure 8: Masks defined along wire

The algorithm automatically hunts for an appropriate seed point by searching inside the plane area outwards from the wire. The condition for a fill at each pixel in the bit-mask is the wire's β function. If a suitable seed

[2] Essentially, a 2D image cutting through the volume dataset - the 2D bit-mask is filled, and the part of the volume touched by the plane used to identify the target object.

point is not found, then the plane is removed from the list, as no object data has been found within the plane's subsection of the volume. To ensure that data at the edges is not skipped, we also apply a morphological dilation operation to the mask. Voxels in the attribute field are now only flagged if the plane mask bit at that point is 1 (See Figure 9).

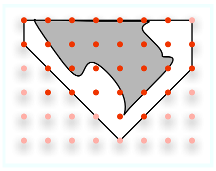


Figure 9: A Plane mask flagging voxels

This solution is effective in that not only does it solve the delineation problem, but it also further reduces the number of flagged voxels in the scene, which reduces rendering time. Backward-mapping operations are now only performed on voxels whose resulting new sample positions lie within the target object (or slightly out). Table 1 gives the number of non-flagged voxels ignored for some example deformations. Note that we do not include samples outside of the field boundaries in the figures.

Dataset	# Sample pts	# Pts ignored	% ignored
CT Carp	26,011,195	15,523,566	59.7%
Visman	45,461,270	39,253,862	86.3%

Table 1: Voxels ignored while rendering

7.2 Speed / Storage / Accuracy trade-offs

Each voxel in the attribute field requires three attributes. The first is γ : the wire reference, the second is ε : the t-value on the wire, and the third is a bit for the flag that denotes a voxel has been swept. If we assume floating-point precision on the t-value, we have a minimum of 5 bytes per voxel including 7 bits for the wire reference with a maximum of 128 wires in the scene. This storage requirement can be reduced by using integer precision on the t-value. Below is an example 2-byte per voxel solution for Catmull-Rom spline wires.

# Bits	Range of values	Data
1	2	swept flag
4	16	w : wire reference
4	16	s : segment index on w
7	128	integer t-value on s

The integer precision on the t-value has another advantage. The points at each integer offset on the wire can be cached before the rendering stage and then used during rendering to avoid expensive curve evaluation at each sample point (the points are chosen by subdividing the curve as in equation 4). The wire point calculation is now reduced to a simple array lookup. If we wish for greater precision still, linear interpolation can be performed between values. The same technique can be used for the wire normals : for each modified wire point p, the difference between the wire normal at p and the normal at the same t-value on the base wire is calculated, and stored.

8 IMPLEMENTATION

The method has been implemented in C++ on GNU / Linux x86. To assist with rapid testing, and to demonstrate the simplicity of specifying deformations, we have built a simple user interface using the GTK+ library. The interface allows the user to view the volume dataset from multiple angles interactively and quickly define and deform wires. The user can also specify an animation by deforming the wire differently for an arbitrary number of frames. The wires can be saved to disk for later retrieval, and rendered into a series of images which can be encoded into a movie.

9 RESULTS

To give a more accurate representation of the overhead of our method implementation, we first give the timings for a software ray-casting volume rendering algorithm written in C++ with very few optimisations (see table 2), and then modify it to work with our method (results in table 3). *Preprocessing* refers to the attribute field generation stage, which also includes mask generation, curve lookup table generation, and other pre-render data discussed in previous sections. The difference in the timings gives the overhead of calculating the attribute field and transforming sample positions in each case.

The base wire and deformed wires are identical to give the same number of sample positions during volume rendering (thus ensuring a fair comparison). The deformation is therefore the identity deformation. The viewing position and image size are also constant. The timings are based on a P4 at 3.2GHz with 512MB RAM.

Dataset	Render time
CT Carp	5.74 secs
Tubes	2.59 secs
Visman torso	4.31 secs

Table 2: Standard rendering times

Dataset	Preprocessing	Render	Total
CT Carp	1.78	12.65	14.43
Tubes	0.96	4.81	5.77
Visman torso	2.97	22.87	25.84

Table 3: Deform/render times

The timings were performed using all acceleration techniques discussed, but the majority of code has not yet been optimised. The tables show that the overhead in the rendering stage is far higher than the preprocessing stage. The biggest factor in the cost of attribute field generation is the size of the field, as more propagation must take place.

Figure 12 shows the visible human rendered with the same deformation (the head has been pulled back), with differing attribute field to dataset ratios. 1:1 (same dimensions) predictably gives the most pleasing reproduction, while a 1:4096 (each dimension is 1/16th the size) field gives a blocky appearance due to trilinear interpolation taking place in the large gaps between voxels. We also give the time for attribute field generation for each.

10 CONCLUSION

We have introduced a new software-based framework for non-linear, non-reconstructive deformation of volumetric datasets. The framework brings a much-needed intuitive deformation method to the field of discretely sampled object representations. The lack of such methods available for volume deformation severely hampers the area, and we feel that this framework goes some way to correct this.

We have shown that the method is not computationally expensive, requires only a small memory storage overhead, and avoids the discussed disadvantages of reconstruction-based methods. The specification of such deformations can be easily defined without knowledge of the internal algorithms that deform the data. The problem of delineating volume objects to deform independently is also handled in a simple manner.

In addition, the standard ray-casting approach to volume rendering can be used to render the result with only minor modifications to the rendering engine. This facilitates the method's integration into the volume deformation and rendering pipeline. Note that our method can also be carried out on the GPU by using the attribute field as a texture - creating a fairly interactive system for deformation and animation.

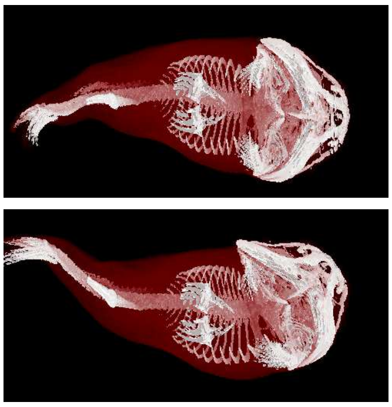


Figure 10: CT Carp deformation

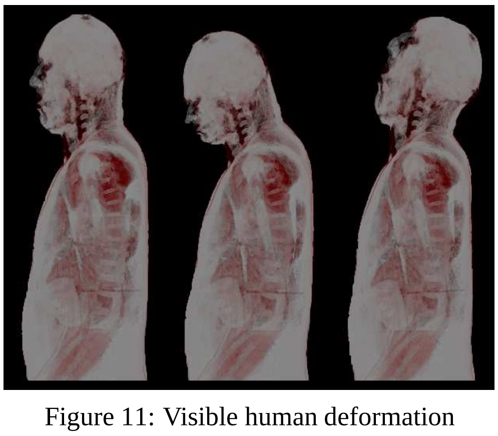
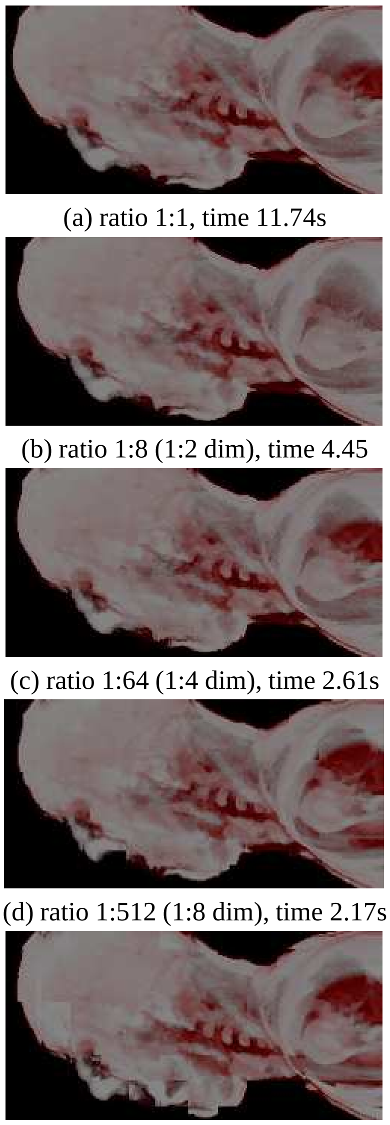


Figure 11: Visible human deformation


(a) ratio 1:1, time 11.74s

(b) ratio 1:8 (1:2 dim), time 4.45

(c) ratio 1:64 (1:4 dim), time 2.61s

(d) ratio 1:512 (1:8 dim), time 2.17s

(e) ratio 1:4096 (1:16 dim), time 2.12s

Figure 12: Attribute field scales

11 ACKNOWLEDGEMENTS

This work has been supported by EPSRC grant GR / S44198. The authors would also like to acknowledge

Stefan Roettger's volume library [Roe], and the National Library of Medicine's Visible Human project.

REFERENCES

[AMBJ00] K. Adbel-Malek, D. Blackmore, and K. Joy. Swept volumes: Foundations, perspectives, and applications. In *International Journal of Shape Modeling*, 2000.

[BM99] D.E. Breen and S. Mauch. Generating shaded offset surfaces with distance, closest-point and color volumes. In *Proceedings of the International Workshop on Volume Graphics*, pages 307–320, March 1999.

[BvNP89] W. F. Bronsvoort, P. R. van Nieuwenhuizen, and F. H. Post. Display of profiled sweep objects. *The Visual Computer*, 5(3):147–157, 1989.

[BW01] D. E. Breen and R. T. Whitaker. A level-set approach for the metamorphosis of solid models. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):173–192, 2001.

[CCI$^+$05] M. Chen, C Correa, S Islam, M. W. Jones, P.Y. Shen, D Silver, S. J. Walton, and P. J. Willis. Deforming and animating discretely sampled object representations. In *Eurographics 2005 STAR Reports*, pages 113–140, Dublin, Ireland, August 2005.

[CSW$^+$03] 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, 2003.

[Gib97] S. Gibson. 3D chainmail: a fast algorithm for deforming volumetric objects. In *Proc. 1997 Symposium on Interactive 3D Graphics*, pages 149–154, April 1997.

[GS01] N. Gagvani and D. Silver. Animating volumetric models. *Graphical Models*, 63(6):443–458, 2001.

[Jon96] M. W. Jones. The production of volume data from triangular meshes using voxelisation. *Computer Graphics Forum*, 15(5):311–318, 1996.

[JS00] M.W. Jones and R.A. Satherley. Shape representation using space filled sub-voxel distance fields. In *Vision, Modeling and Visualization*, pages 316–325, 2000.

[Kaj83] J.T. Kajiya. New techniques for ray tracing procedurally defined objects. In *SIGGRAPH '83*, pages 91–102, New York, NY, USA, 1983.

[KY95] Y. Kurzion and R. Yagel. Space deformation using ray deflectors. In *Proc. 6th Eurographics Workshop on Rendering 1995*, pages 21–32, Dublin, Ireland, June 1995.

[Lak00] S. Lakare. 3D segmentation techniques for medical volumes. 2000.

[Roe] Stefan Roettger. The volume library. http://www9.cs.fau.de / Persons / Roettger / library/.

[Sch90] P. Schneider. Solving the nearest-point-on-curve problem. In *Graphics Gems*, volume 1, pages 607–612. Academic Press, 1990.

[Sea97] G. Sealy. Representing and rendering sweep objects using volume models. In *CGI '97*, pages 22–27, Washington, DC, USA, 1997.

[Sha03] M. Shalaby. Spline implicitization of planar curves and applications, 2003.

[SJ01] R.A. Satherley and M.W. Jones. Vector-city vector distance transform. *Computer Vision and Image Understanding*, 82(3):238–254, 2001.

[SK00] M. Sramek and A.E. Kaufman. vxt : A c++ class library for object voxelisation. In *Volume Graphics*. Springer, 2000.

[SSC03] V. Singh, D. Silver, and N. Cornea. Real-time volume manipulation. In *Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics*, pages 45–52, 2003.

[vW84] Jarke J. van Wijk. Ray tracing objects defined by sweeping planar cubic splines. *ACM Trans. Graph.*, 3(3):223–237, 1984.

[WC01] A.S. Winter and M. Chen. *vlib*: A volume graphics API. In *Volume Graphics 2001*. Springer-Wien New York, 2001.

[WC02] A.S. Winter and M. Chen. Image-swept volumes. *Computer Graphics Forum*, 21(3):441–441, 2002.

[Wes90] L. Westover. Footprint evaluation for volume rendering. *Computer Graphics*, 24(4):367–376, August 1990.

[WP99] Z. Wu and E.C. Prakash. Visible human walk: bringing life back to the dead body. In *VG99*, pages 347–356, 1999.

[WRS01] R. Westermann and C. Rezk-Salama. Real-time volume deformations. In *Comput. Graph. Forum*, volume 20, 2001.