Fast Volume Deformation Using Inverse-Ray-Deformation and FFD

H. Chen, J. Hesser, R. Männer
Lehrstuhl für Informatik V, Universität Mannheim,
Mannheim, Germany

ABSTRACT

In this paper we present a new approach for free-form-deformation of volume object. Free-form-deformation (FFD) is used as method for calculating the deformation. Rendering is based on the inverse deformation of viewing rays using approaches adapted from Barr [1] and Kurzion [10]. Thus this new approach doesn't suffer from the overhead of the intermediate step to reconstruct the deformed volume as traditional approaches do. Compared to these earlier methods we integrated algorithmic optimization like distance-coding based space-leaping and early-ray-termination. This is achieved by approximating the deformed viewing rays with polylines and by treating the linear segments as conventional rays.

The performance of this optimized rendering method depends on the length of the ray segments. The higher the local curvature, the smaller the segment length which is determined automatically during rendering. For semi-transparent volumes we thus achieve speedups of the optimized algorithm vs. the non-optimized algorithm between 3.13 and 4.43. In the case of opaque objects and empty space in-between the speedup decreases from 14.16 of non-deformed volumes to 4.52 for severely deformed volumes. Thereby the decrease of speedup is mainly determined by less efficient space-leaping due to shorter segments in the polylines.

Keywords: *volume deformation, inverse ray deformation, volume graphics, , volume animation, volume rendering, algorithmic optimization.*

1. INTRODUCTION

In computer graphics, an object is traditionally modeled using polygon-based meshes. The rendering of polygon models can be accelerated by graphics hardware. The real time rendering of polygon-models is therefore commonly applied in a wide range of fields, such as computer games, film industry, and TV. However, applications such as medical surgical simulation require the visualization of both the objects' surface and inner structure. In such applications, objects can not be easily described with a polygon-model, especially when there is fuzzy edges-transition between different parts of the objects. Volume rendering is more suitable for such applications.

When volume rendering was first introduced in the 1980s, it was notorious for its enormous requirements on the computer resources. The improvement of techniques results in the steady increase of computing power and turns impossible things now possible. Today volume data with reasonable size($\geq 256^3$) can be rendered in real time[5][23]. Nevertheless current available interactivity of volume rendering can only be achieved for the static volume data. Many applications need to deform the volume as response to the external stimulation assigned on the volume objects. To deform the objects in the scene, the volume must be resampled once whenever a snapshot of the volume is rendered. Such a process slows down the available frame rate, since resampling the original volume is of the complexity of $O(n^3)$. In addition, extra memory space is necessary to hold the intermediate deformed volume.

To render the deformed volume with the same complexity as rendering static volume, we propose a rendering scheme based on inverse bending of rays. In this scheme the deformed volume is not reconstructed, instead, in the rendering phase the ray is bend to the inverse direction of the deformation, thus generating the image of the deformed volume. This volume deformation approach does not generate the intermediate deformed volume. The extra memory for holding the intermediate deformed volume is therefore not required. In principle, this method is independent of the deformation definition. FFD[2][3][4], functional deformation[1] or other special operators[10] all can work well.

The paper comprises 6 sections. Section 2 outlines some related work on volume deformation. In section 3 we shortly describe the background of volume rendering and two popular rendering acceleration algorithms. Section 4 contains the implementation of the new deformation algorithm. The experimental results and analyses can be found in section 5. The outlook in section 6 concludes this paper.

2. EARLIER WORK

A rich literature on object deformation can be found since the 1980s. However, volume deformation is less discussed, since interactive volume rendering became available only after 1995.

Volume morphing[13] by Hughes generates a smooth transition from a source volume model to a target through a scheduled interpolation of two objects' Fourier frequency. It provides a method to reconstruct the intermediate shape of the deformed volume, just like the popular keyframe interpolation method for mesh models used by computer animators. However, in interactive applications the target volume object itself is not available, therefore volume morphing cannot be used in such applications, besides, the computing power requirement is enormous to carry out twice the 3D Fourier transform each frame.

Schiemann and Höhne[14] described a true volume transformation method. The transformation is based on surface models which fit the objects detected in the volume. First the surface models are deformed interactively, the deformation of the surface models is then spread to the whole volume through interpolation, this way they achieved real time interactivity for volume deformation.

Gagvani [15] proposed a Skeleton-Tree based volume animation method. The volume is first thinned to generate a skeleton-tree. At the same time the shortest distance of each voxel on the skeleton tree to the object boundary is estimated and saved. The volume deformation is then realized by manipulating the skeleton-tree followed by reconstructing the deformed volume

using the skeleton-tree and the distance information. However the voxel's gray value information is lost during this process. Chandru's volume keyframe animation system [16] creates volume sequences through sculpting operations. Similar to Gagvani's method, the animated volume is binary and the voxel in the volume has no gray value information.

Approaches with physical fidelity are also proposed to deform volume objects, for example the Finite-Element-Method(FEM) [7][8][9], Mass-Spring systems [11][12]. These methods share a common disadvantage: before the rendering of the deformed volume a separate process to reconstruct the volume with the deformation information is necessary. This additional reconstruction process is redundant and is the main factor that makes the volume deformation far more time-consuming than static volume rendering.

Kurzion and Yagel introduced ray deflectors [10] to deform any ray-traceable objects in the scene. The ray deflectors are derived from Barr's idea of inverse-ray- deformation [1]. The ray deflectors define a region in the 3D space inside which the ray is bent to the opposite direction of the deformation. When the ray is followed along its deformed trace, the object in the scene will look as being deformed, although the original object in the scene is in fact not modified at all. However, complex deformations are difficult to be implemented by deflectors.

The volume deformation approach we present in the following sections uses also the inverse deformation of rays. Instead of ray deflectors we use a FFD grid to define the space deformation, because the FFD has been a powerful tool for interactive object modeling and animation. Many users of popular graphics software are familiar with it. Since the intermediate step to reconstruct the deformed volumetric data is no more necessary in our approach and the algorithmic optimizations, like early-ray-termination and space-leaping, are easily incorporated in the deforming and rendering process, the complexity increase of our deformation approach is very limited compared to the static volume rendering. We begin the presentation of our deformation method with the background of volume graphics.

3. BACKGROUND

Traditionally polygon meshes are used to describe objects in 3D space in computer graphics. Since a polygon mesh has zero thickness, it can not fully describe the content of a 3D objects. A complete presentation of a 3D object uses volumetric data. The volumetric data stores the 3D objects information in a 3D lattice of points, each point on the lattice is called a voxel which is the basic element of volume graphics. Each voxel stores a constant amount of information, for example, the density of a material at that location. Through volume rendering the virtual scene of the 3D volume object is projected on the 2D image plane. Volume rendering simulates the propagation of light through volume space. For each pixel a ray is cast into the virtual scene. Using the low albedo model the intensity I(a,b) reflected by a ray travelling from point *a* to point *b* is given by the volume rendering equation:

$$I(a,b) = \int_a^b e^{-\int_a^s \sigma(s')ds'} q(s)ds \qquad (1)$$

Where $\sigma(s')$ is the absorption function, $q(s)$ is the emission function. The standard solution for equation (1) uses discretization in equally spaced sample points and applies the rectangle rule:

$$I(a,b) = \sum_{i=1}^{n-1} e^{-\sum_{j=0}^{i-1}\sigma_j \Delta s} q_i \Delta s = \sum_{i=1}^{n-1}\left(\prod_{j=0}^{i-1} e^{-\sigma_j \Delta s}\right) q_i \Delta s = \sum_{i=0}^{n-1}\left(\prod_{j=0}^{i-1}\theta_j\right) q_i \Delta s \qquad (2)$$

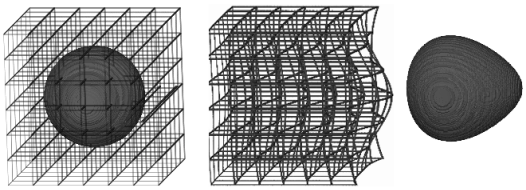
θ_j in equation (2) is the sample point's transparency, it equals *1-*α_j. where α_j is the opacity of the sample point, which is determined by the opacity-transfer-function. The sample point's emission q_i is evaluated by the product of opacity and shading function. To evaluate a pixel's intensity, at each sample point, the opacity is estimated by using e.g. tri-linear interpolation and shading is calculated by using local gradients. Then the sample points' contribution to the pixel is accumulated by the compositing operation. To avoid artifacts, the sample step Δs is chosen to be equal or less than the voxel size, thus the complexity of standard volume rendering is of or larger than the volume size. This makes volume rendering a computation intensive process. To implement volume rendering in real time, algorithmic optimization is necessary. Early-ray-termination and space-leaping are two efficient methods for acceleration.

Early-ray-termination is used to terminate the further sampling and compositing along a ray when the remaining voxels' contribution to the pixel intensity becomes neglectable. Return to equation (2), when the transparency becomes too low, i.e. $\prod_{j=0}^{i-1}\theta_j < \theta_{th}$, the remaining voxels along the ray are occluded, the processing of the ray therefore can be terminated. Similarly, when a voxel contribution to the pixel, i.e. the item q_i in equation (2), is too small, it is called an empty voxel. The sampling and compositing for the empty voxel is unnecessary. Since the emission function is the product of opacity and shading function, we can determine before rendering if a voxel is empty or not when an opacity-transfer-function is selected. By determining for each empty voxel the shortest distance to a non-empty voxel in its 3D space, the empty space in the volume can be efficiently leaped over. This is how space leaping works. The distance for space leaping is calculated view-independently in a preprocessing stage to make the acceleration fully available in the rendering phase.

4. FFD AND INVERSE-RAY-EFORMATION BASED DEFORMATION APPROACH

4.1 Define space deformation using FFD

We consider only the volumetric data with voxels located on a regular lattice. Correspondingly we use the uniform B-spline presentation of FFD to define space deformation. The FFD is in principle a R^3 to R^3 mapping which transforms any point inside the boundary of the FFD grid to its new position. Through the embedding of the volume object in the FFD grid, the deformation of the volume object can be determined by only manipulating the control points of the FFD grid. Figure 1 shows the procedure to deform an object using FFD. Unlike standard applications of FFD, we do not use the FFD to directly deform the volume object, instead, we only use the space deformation defined by FFD to inversely transform viewing rays in the rendering phase.



a) b) c)

Figure 1. Deform an object with FFD
a). Embed the object in the FFD grid
b). Change control points on FFD grid to define deformation
c). The deformed object

4.2 Inverse-Ray-Deformation

Inverse-ray-deformation [1][10] is a way to generate an image of deformed objects without really deforming the original objects. To generate the image of a deformed object, we can transform the ray path in the space to the opposite direction of deformation. Figure 2 shows how the inverse-ray-deformation works. Inverse-ray-deformation is of great meaning for volume deformation, since the volume rendering itself is very time-consuming. When the deformation is merged in the rendering procedure, the expensive intermediate step to reconstruct a deformed volume is no longer necessary. The deformation is therefore of high efficiency.

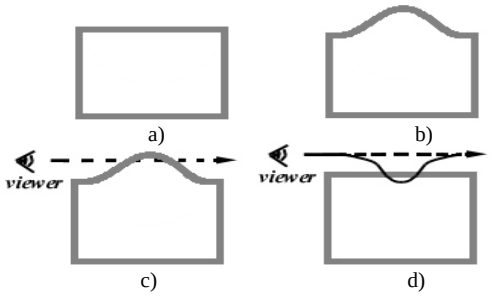


Figure 2. Inverse-ray-transformation for object deformation
a). the original object shape; b). the deformed object shape;
c). normal deformation—deform the object model then render it;
d). inverse ray transformation based deformation—let the object keep its original shape, during rendering follow the inversely transformed ray path to produce an illusion of the deformation.

4.3 Calculate the ray path in the deformed space

Let $\{P(i)|i=0,1,2...n\}$ in figure 3 denote the point set on a viewing ray which locates inside the boundary of the FFD grid. $\{P_{ffd}(i)|P_{ffd}(i)=T_{ffd}(P(i)),\ i=0,1,2...n\}$ denotes the transformed version of these points. The inverse transformed points set of the ray $\{P_{inv}(i)|i=0,1,2....n\}$ is then defined with:

$$Pinv(i) = P(i) - [Pffd(i) - P(i)] = 2P(i) - Pffd(i) \qquad (3)$$

The sequence of inversely transformed points are connected by linear segments. Thus the whole deformed ray is represented by a polyline.

The original points of the undeformed viewing ray, i.e., $\{P(i)|i=0,1,2...n\}$, are not selected arbitrarily. In order to follow the known Sampling Theorem, the intervals between these points are chosen according to the magnitude of the local deformation.

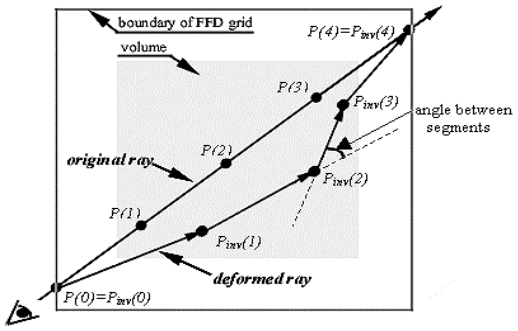


Figure 3. The division of the original ray and its inverse transformation

4.4 Ray-casting in the deformed space

Since the ray is subdivided into a series of linear segments, each segment is processed in the same way as rays in standard ray-casters [18][19]. Along these segments, the ray is sampled at equidistant positions using e.g. tri-linear resampling. Gradients are estimated and shading is performed followed by compositing.

An important issue is the opacity correction. Since the ray path is approximated by piecewise line segments having arbitrary length, there is no guarantee that the sample point interval length is uniform. In general, the last interval is shorter (see figure 4). In this case, we use opacity correction [20] based on a look-up table to compensate the mismatch of sample distance.

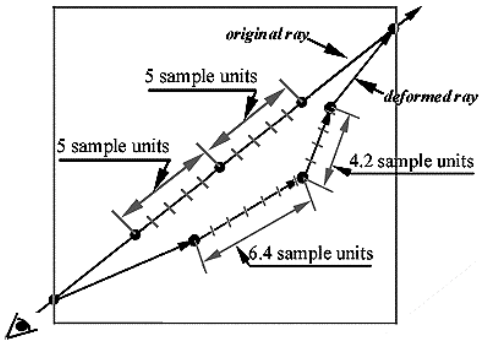


Figure 4. Mismatch of deformed ray segments to the sample interval

The ray casting procedure in the deformed space is described as follows:

Ray-casting in the deformed space

```
nitialize scene settings
manipulate the FFD grid's control points   //defining the desired deformation
for every pixel on the image plane do
begin
  generate a ray originated from the viewer and passing the pixel;
  divide the ray into small segments with proper interval length;
  transform ray-segments into a polyline approximating the deformed ray;
  Ray_opacity   =0 ;
  Ray_RGB    =0 ;
  Ray_position  = polyline's start point;
  for each deformed ray segment do
   begin
    Ray_direction =  the segment's direction;
    sampleNumber=(integer)length(raySegment)/sampleStep;
    distanceToCompensate=length(raySegments)-sampleNumber*sampleStep;
    for sample =0 to sampleNumber-1  do
      begin
        sample_opacity = tri-linear_interpolation(Ray_position);
        sample_RGB   = PhongShading( light,  Ray_direction, Ray_position);
        Ray_RGB      = Ray_RGB   + sample_opacity *sample_RGB * (1- Ray_Opacity)
        Ray_Opacity     = Ray_Opacity+ sample_opacity * (1- Ray_Opacity)
        Ray_position    = Ray_position+ Ray_direction   *  sampleStep
      End
    if(distanceToCompensate not equal 0) do
      begin
        // compensate the mismatch between
        // ray segments' length and sample interval
        sample_opacity = tri-linear_interpolation(Ray_position);
        sample_opacity=opacityCorrect(distanceToCompensate, sample_opacity);
        sample_RGB   = PhongShading( light, Ray_direction, Ray_position);
        Ray_RGB      = Ray_RGB   + sample_opacity *sample_RGB * (1- Ray_Opacity)
        Ray_Opacity     = Ray_Opacity+ sample_opacity *(1- Ray_Opacity)
        Ray_position    = Ray_position+ Ray_direction    *distanceToCompensate;
      end
   end
  save pixel's RGB and Opacity values;
end
```

Adaptively select the length of ray segments

```
oldLength    =  minimalSegLength          // initialize oldLength
oldDirection = directionOfOriginalRay     // initialize oldDirection
oldPointIndex=0;
P[oldPointIndex]=StartPointOfTheRay
P_inv[oldPointIndex]=InverseTransform(P[oldPointIndex]);
while( not end of the ray) do
begin
  currentSegmentLength=2*oldLength;
  currentPointIndex=oldPointIndex+1;
  greedy-heuristic module:
  begin
    P[currentPointIndex]=P[oldPointIndex]+directionOfOriginalRay*
                         urrentSegmentLength
    P_inv[currentPointIndex]=inverseTransform(P[currentPointIndex]);
    CurrentSegmentDirection=direction( P_inv[currentPointIndex],
                                       P_inv[oldPointIndex]);
    if(angle(currentSegmentDirection, oldDirection)>5 degree)
      begin
        currentSegmentLength=currentSegmentLength/2;
        if(currentSegmentLength<=minimalSeglength)
          currentSegmentLength=minimalSeglength;
        else
          redo greedy-heuristic module
      end
    if(currentSegmentLength>maximalSegLength)
      currentSegmentLength=maximalSegLength;
     oldPointIndex= currentPointIndex;
     oldDirection  = CurrentSegmentDirection;
     oldLength     = currentSegmentLength;
  end
end
```

4.5 Adaptively select the length of ray segments

In our ray-casting procedure the ray path in the deformed space is approximated by a polyline. The polyline is generated by dividing the original viewing ray into small segments and then inversely transforming them as described in the section 3.1. On one hand, the shorter the segments the original ray is divided into, the better the polyline approximates the deformed ray path. On the other hand, more segments cost more computations to perform the deformation. An adaptive segment length is therefore necessary.

The ray segment length is determined by a greedy-heuristic algorithm [17]. It uses the angle between two consecutive segments to recursively subdivide them. This method works well if an upper bound on the maximal curvature is known. The procedure to divide a ray segment (refer to figure 3) is realized as follows:

During dividing the original ray, we set an upper limit to a single ray segment's length, i.e. maximalSegLength. The maximalSegLength is determined by the radius of the largest curvature of the deformed ray. This maximal curvature is obtained by direct computation from the FFD-grid and the given spline function. The minimal segment length is given by 0.5*voxel size due to the Shannon Theorem.

4.6 Algorithmic optimizations

Volume rendering is a very computationally expensive procedure. It achieves interactive rates if algorithmic optimization techniques [21][22][23] are used. In the following we show that algorithmic optimization can be integrated in our deformation approach without any problems.

The two techniques for algorithmic optimization we implemented are early-ray-termination [19] and space-leaping [24][25]. They are integrated in the following way:

- Early-ray-termination

Although the ray path is approximated by a polyline, the rays are still followed from front to back through the volume, so early-ray-termination can still be implemented by checking the transparency

of the ray as the polyline is followed and stopping the processing of the current ray if the transparency becomes lower than an user-defined threshold.

- Space-leaping

For space-leaping we rely on distance transform [24] that encodes for each voxel the distance to the nearest non-transparent voxel in its 3D neighborhood. Since each ray segment is a straight line in the undeformed volume, we can apply the distance values to determine the leap distances. There is only one restriction: the leap distances must be shorter than the length between the current sample point and the end point of the segment.

5. EXPERIMENTAL RESULTS AND ANALYSES

We tested the volume deformation approach with different volumetric data sets as examples. Since we are primarily interested in the efficiency of the new volume deformation approach, we examined the effect of deformation amplitude on the performance. To measure the performance, we use three values: the average length of segments, the average number of non-transparent sample points/ray (sampling/ray) and the average number of transparent sample points/ray that are hit during space-leaping (distance/ray). **We use these measures instead of other statistics like rendering time or frame rate because they are independent of the platform and the level of code optimization**. We got similar results for different data sets. We present and analyze here only the results for the volume data engine. In the experiments, a semi-transparent mapping and two different opaque mappings differing in the percentage of empty space in the volume are tested. Images in figure 5 shows the deformation of the volume engine ($256^2 \times 113$ voxels) with different magnitudes of deformation and different opacity mappings.

The curves in figure 6 show the performance change of the deformation approach as the function of deformation amplitude when early-ray-termination and space-leaping are used. As comparison, the curves in figure 7 shows the performance change of the deformation approach without algorithmic optimization. For the non-optimized ray casting the opacity mapping has no impact on the performance, therefore the curves in figure 7 are all identical. On the contrary, for the optimized method the impact of the opacity mappings on the performance is obvious, as the curves in figure 6 show.

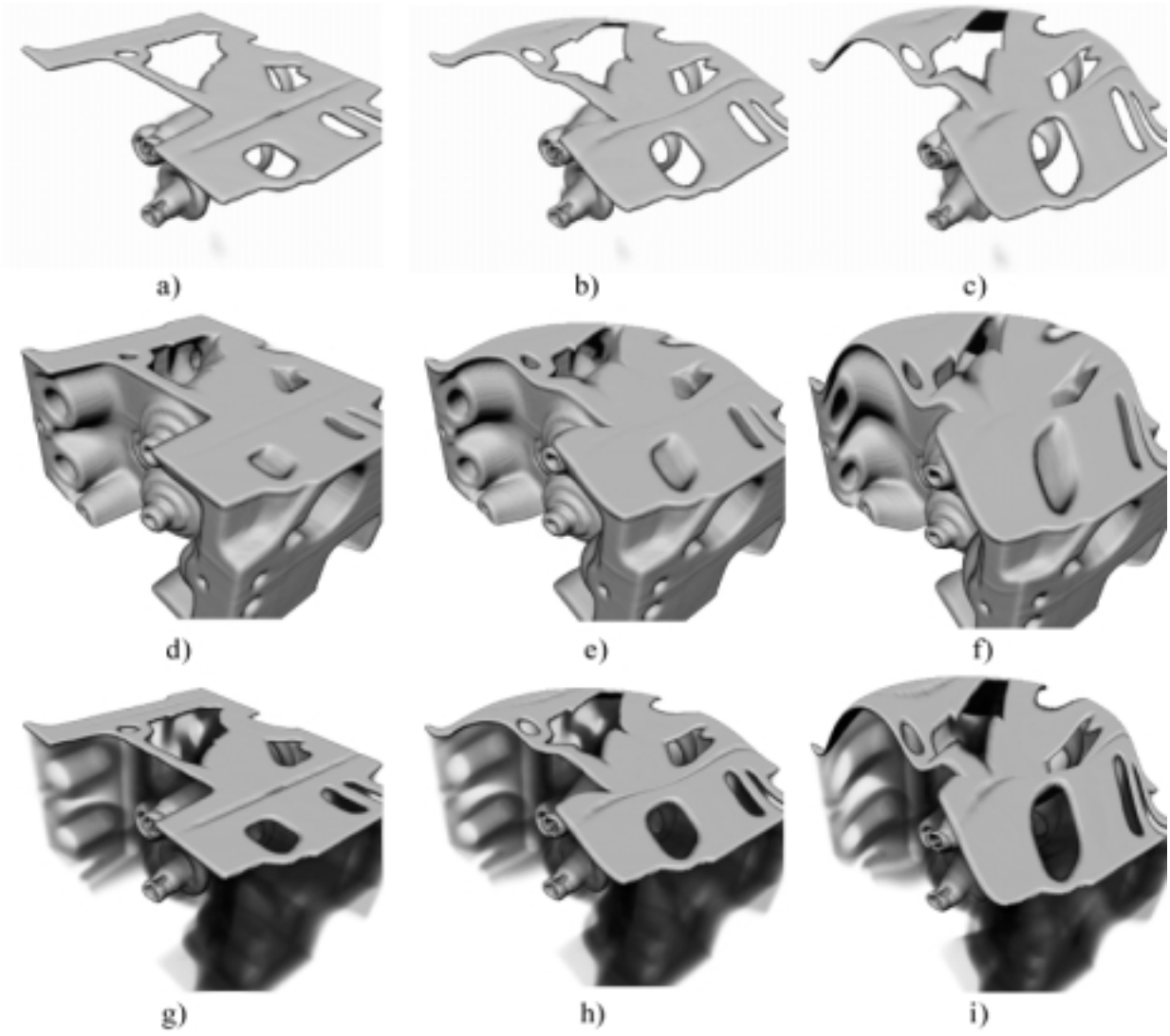
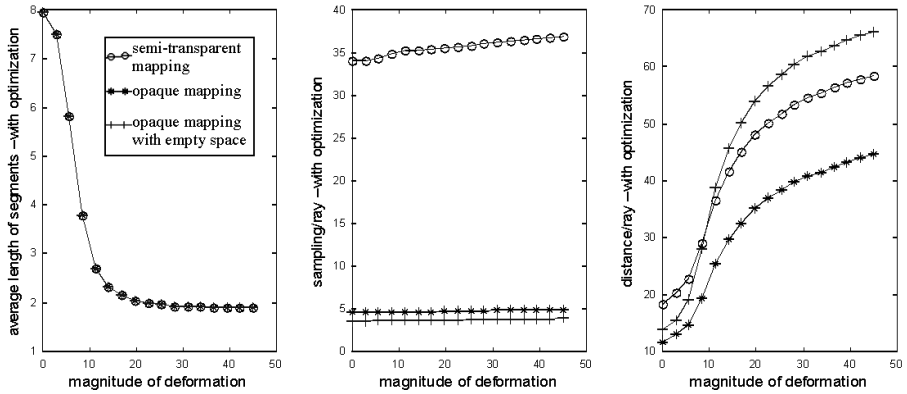
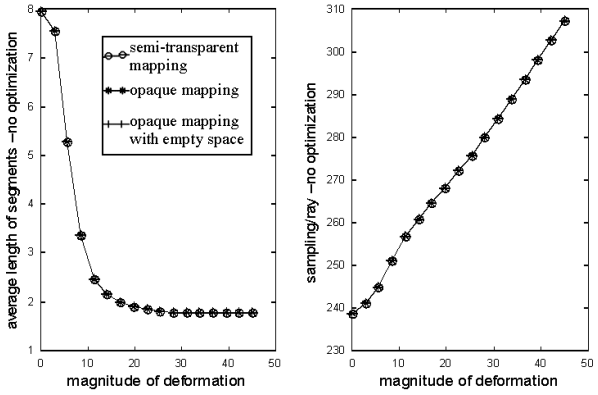


Figure 5. Deformed engine with different deformation magnitudes and different opacity mappings



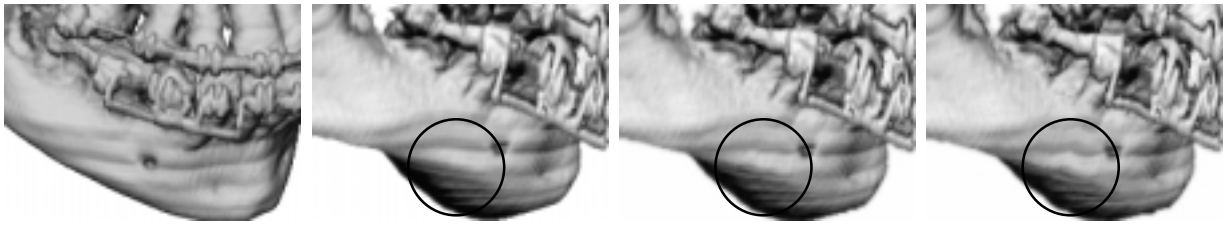
a) b) c)

Figure 6. Results for ray casting with algorithmic optimization



a) b)

Figure 7. Results for ray casting without algorithmic optimization



a. no deformation

b. severe deformed using adaptive division of ray (average length is 2)

c. severe deformed using fixed ray segment length of 2

d. severe deformed using fixed ray segment length of 4

Figure 8. Comparison of the deformation continuity

The values minimalSegLength and maximalSegLength are chosen to 0.5 and 8 (voxel size). According to figure 6a/7a, the average length of the ray segments decreases with the deformation magnitude from 8 to a value between 1.8 and 2.0. It shows how the ray is adaptively divided into smaller segments to match the deformation magnitude. Thus the extra computation necessary to deform the viewing ray does not increase further when the limiting value of the average length of ray segments is approached. In addition, the adaptive segment length selection guarantees high-quality rendering results. This can be found in the rendered images in figure 8. In this figure image *b*, *c* and *d* are rendered with the same magnitude of deformation using adaptive division of ray segment and fixed segment with length of 2 and 4 separately. As a reference, image *a* in figure 8 is rendered with the same camera settings but without deformation. The deformation in image *b* is smooth, but in image *c* and d the deformation is rough and discontinuous in the severe deformed region as marked by the circles, showing the superior performance of the adaptive selection of the ray segment length.

For the optimized ray casting, the trends of the sampling/ray change with the deformation magnitude is quite different compared to that of the non-optimized method. For the non-optimized method, as shown in figure 7b, the sampling/ray increases steadily with the increase of the deformation magnitude. The reason is that the ray path in the deformed space is longer due

to the increase of the deformation magnitude, but along the deformed ray the volume is still sampled equidistantly. For the optimized case, shown in figure 6b, the increase of sampling/ray is only gradual. Theoretically, there should be no increase of samples/ray if we use space leaping and adaptive ray termination together. The reason is that the longer ray path in the deformed space does not change the condition for early-ray-termination. Nevertheless, since the length of the deformed ray segment is often not an exact multiple of the sampling distance, the length for the last sample interval is usually shorter than the standard sample distance (which is compensated as described in section 4.4). This leads to additional samples and increases the number of samples/ray with the deformation. This is more striking for the semi-transparent mapping. Here, for a ray, a long path through the volume is traversed before the accumulated opacity of the viewing ray exceeds the threshold for early-ray-termination.

For distance/ray the case is different. Since the average segment length is reduced, the maximal leap distance in space-leaping is limited. The consequence is that additional sample points at the boundaries between two consecutive segments are necessary. This increase can directly be seen in figure 6c. This effect of the average length of ray segments on the efficiency of space-leaping can be observed more clearly if we compare figure 6a with figure 6c. When the deformation amplitude is between 0-25, the average length of ray segments decreases dramatically and the distance/ray increases correspondingly, as shown in figure 6c. As the average length of ray segments approaches its converging value, the distance/ray increases linearly with the deformation since the length of the ray increases linearly as well.

It is difficult to evaluate the change of the acceleration rate as a function of the deformation magnitude, since there are two computationally different types of operations in the optimized ray casting: real sampling operations of the volume and operations to access only the distance for space leaping. The latter is computationally cheaper than real sampling operations which involves opacity interpolation, shading and compositing. Even so, we list the coarsely evaluated acceleration rates under different deformation magnitudes in table 1, where every operation to retrieve a distance value for space-leaping is also counted as a normal “sampling” operation. As shown in the table, when there is no deformation, the acceleration rate for the opaquely mapped volume, which has a lot of empty space, is 14.16. This is much higher than the acceleration rate for the semi-transparent mapped volume, which is 4.43. For semi-transparent objects the number of sample points decreases only gradually, while for opaque scenes there is a significant decrease of a factor of two to three.

Table 1. Ratios of overall sample number between non-optimized and optimized ray casting

setting of engine’s opacity	deformation magnitude			
	0	15	30	45
Semi-transparent	4.43	3.29	3.13	3.23
opaque	13.70	7.11	6.23	6.20
opaque with lots of empty space	14.16	5.10	4.67	4.52

This shows that the deformation has its main effect on space-leaping due to the limitation of the space-leaping distance by the segment length.

6. OUTLOOK

We have presented a method for volume rendering of deformed objects with support of algorithmic optimizations by combining FFD with inverse-ray-deformation. Through the inverse deformation of the viewing ray, the desired deformation effect can be achieved without reconstructing the intermediate deformed volume. Rendering of deformed volumes is therefore of the same complexity as the rendering undeformed volumes, although it depends additionally on the magnitude of the deformation. Further, the algorithmic optimizations, like distance-coding based space-leaping and early-ray-termination are integrated in this method without any difficulty.

Experiments show that the magnitude of deformation have a very limited effect on the efficiency of early-ray-termination, while the efficiency of space-leaping is decreased due to shorter segments of polyline which approximates the ray path in the deformed space. With large deformations we still achieve a speedup compared to the non-optimized version of 3.13, 6.20, and 4.52 for semi-transparently, opaquely, and opaque-emptily mapped volume data respectively.

Although the space deformation can be realized with different methods, such as functional deformation and ray-deflectors, we use a B-spline FFD grid to define the deformation, because FFD is a very powerful deformation tool and is widely accepted in computer graphics community. The FFD control points are interactively changed to design the desired deformation. Such a process has no physical fidelity. We plan to incorporate physical reality into the deformation procedure. This will be implemented by combining the FFD with the other physical deformation models, e.g. Mass-Spring systems or Finite-Element-Method. In fact, the FFD is suitable to be combined with such physical deformation models [26][27]. With current technologies the deformation using a Mass-Spring system or other simplified FEM variations can be implemented in real time [8]. We plan therefore to combine the Mass-Spring system and FFD grid into a generator of space deformation, thereby developing a complete volume deformation system which exploits the high efficiency of the inverse-ray-deformation based ray casting approach proposed in this paper to deliver volume deformation with interactive speed.

REFERENCE

[1]A. Barr, Global and Local Deformations of Solid Primitives, ACM Computer Graphics, vol. 18, pp.21-30, 1984.

[2]T. Sederberg and S. Parry, Free-Form Deformation of solid Geometric Models, ACM Computer Graphics, vol. 20, pp.151-160, 1986.

[3]J. Griessmair and W. Purgathofer, Deformation of Solids with Trivariate B-splines. Eurographics’89, pp.134-148, 1989.

[4]S. Coquillart, Extended Free Form Deformation: a Sculpturing Tool for 3D Geometric Design. ACM Computer Graphics, vol. 24, pp.187-193, 1990.

[5]H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, L. Seiler, The VolomePro Real-Time Ray-Casting System, Proceedings SIGGRAPH 99, pp.251-260, 1999.

[6]D. Chen and D. Zeltzer, Pump It Up: Computer Animation of a Biomechanically Based Model of Muscle Using the Finite Element Method, Proceedings SIGGRAPH'92, pp.89-98, 1992.

[7]S. Cotin, H. Delingette and N. Ayache, Real-Time Elastic Deformations of Soft Tissues for Surgery Simulation, IEEE Transaction on Visualization and Computer Graphics, Vol. 5, pp.62-73. 1999.

[8]S. Gibson, B. Mirtich, A Survey of Deformable Modeling in Computer Graphics, http://www.merl.com/reports/TR97-19/index.html.

[9]Y. Chen, Q. Zhu and A. Kaufman, Physically-based Animation of Volumetric Objects Proceeding of IEEE Computer Animation '98, pp.154-160, 1998.

[10]Y. Kurzion and R. Yagel, Space Deformation Using Ray Deflectors, Proceedings of the 6th Eurographics Workshop on Rendering, Dublin, Ireland, pp.21-32, 1995.

[11]D. Terzopoulos and K. Waters, Physically-based Facial Modeling, Analysis, and Animation, Journal of Visualization and Computer Animation, vol. 1, pp.73-80, 1990.

[12]L. Nedel and D. Thalmann. Real Time Muscle Deformations Using Mass-Spring Systems, Computer Graphics International, pp.156-165, 1998.

[13]J. Hughes, Scheduled Fourier Volume Morphing, ACM Computer Graphics Vol. 26, pp.43-46, 1992.

[14]T. Schiemann and K. H. Höhne, Definition of Volume Transformations for Volume Interaction, in: Duncan J, Gindi G [eds.], IPMI'97, Springer, Heidelberg, 1997.

[15]Nikhil Gagvani, Deepak R. Kenchammana-Hosekote, D. Silver, Volume Animation Using the Skeleton Tree. Proceedings IEEE Symposium on Volume Visualization, pp.47-53, 1998.

[16]V. Chandru, N. Mahesh, M. Manivannan, S. Manolhar, Voxel-based Sculpting and Keyframe Animation System, Computer Animation'00, 2000.

[17]Algorithms and theory of computation handbook, ed. by Mikhail J. Atallah. - Boca Raton, Fla. : CRC Press, 1999.

[18]R. A. Drebin, L. Carpenter, P. Hanrahan, Volume Rendering, ACM Computer Graphics Vol. 22, pp.65-74,1988.

[19]M. Levoy, Efficient Ray Tracing of Volume Data, ACM Transactions on Graphics, vol.9,pp.245-261, 1990.

[20]P. Lacroute, Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation, Ph.D. dissertation, Technical Report CSL-TR-95-678, Stanford University, 1995.

[21]P. Lacroute, M. Levoy, Fast Volume Rendering Using a Shear-Warp Factorization of the View Transformation, Proceedings SIGGRAPH'94, pp.451-458, 1994.

[22]R. Yagel, Towards Real Time Volume Rendering, Proceedings of GRAPHICON'96, Vol. 1, pp.230-241, 1996.

[23]Gunter Knittel, The UltraVis System, Proceedings Volume Visualization and Graphics Symposium 2000, pp.71-79, 2000.

[24]K. J. Zuiderveld, A. H. Koning, M. A. Viergever, Acceleration of Ray-Casting Using 3D Distance Transforms, Proceedings of Visualization in Biomedical Computing, pp.324-335, 1992.

[25]R. Yagel and Z. Shi, Accelerating Volume Animation by Space-Leaping, Proceedings of Visualization'93, pp.62-69, 1993.

[26]J. Chadwick, D. Haumann and R. Parent, Layered Construction for Deformable Animated Characters, Proceedings SIGGRAPH'89, pp.243-252, 1989.

[27]G. Hirota, R. Maheshwari, M. C. Lin, Fast Volume-Preserving Free Form Deformation Using Multi-Level Optimization, Proceedings of ACM Symposium on Solid Modeling and Applications, 1999.

About the authors

Haixin Chen is a PhD student of the institute of Computer Science V, University of Mannheim, Germany.

Post Address:

ICM, Universität Mannheim,
B6, 23-29
D-68131 Mannheim, Germany

E-mail: chen@mp-sun1.informatik.uni-mannheim.de
Tel.: +49 621 181 2554
Fax.: +49 621 181 2634

Dr. Jürgen Hesser is a deputy professor at the Institute for Computational Medicine, University of Mannheim/Heidelberg, Germany.

Post Address:

ICM, Universität Mannheim,
B6, 23-29
D-68131 Mannheim, Germany

E-mail: jhesser@rumms.uni-mannheim.de
Tel.: +49 621 181 2635

Prof. Dr. Reinhard Männer, the chair of Computer Science V, University of Mannheim, Germany.

Post Address:

Lehrstuhl für Informatik V,
Universität Mannheim,
B6, 23-29
D-68131 Mannheim, Germany

E-mail: maenner@ti.uni-mannheim.de
Tel.: +49 621 181 2640