Interactive Cutting Operations for Generating Anatomical Illustrations from Volumetric Data Sets

Jörg Mensmann Timo Ropinski Klaus Hinrichs
Department of Computer Science
University of Münster
Einsteinstraße 62
48149 Münster, Germany
{mensmann, ropinski, khh}@math.uni-muenster.de

ABSTRACT

In anatomical illustrations deformation is often used to increase expressivity, to improve spatial comprehension and to enable an unobstructed view onto otherwise occluded structures. Based on our analysis and classification of deformations frequently found in anatomical textbooks we introduce a technique for interactively creating such deformations of volumetric data acquired with medical scanners. Our approach exploits the 3D ChainMail algorithm in combination with a GPU-based ray-casting renderer in order to perform deformations. Thus complex, interactive deformations become possible without a costly preprocessing or the necessity to reduce the data set resolution. For cutting operations we provide a template-based interaction technique which supports precise control of the cutting parameters. For commonly used deformation operations we provide adaptable interaction templates, whereas arbitrary deformations can be specified by using a point-and-drag interface.

Keywords: Volume rendering, volume deformation, volume cutting, illustration.

1 INTRODUCTION

In recent years medical imaging technologies such as computed tomography (CT) or positron emission tomography (PET) have revolutionized many areas of medical diagnosis and other fields of medical practice. However, medical students still learn from anatomical textbooks and atlases, in which the style of illustrations has not changed fundamentally for over a century.

While many research approaches aim at 3D volume visualization, the aspect of creating illustrative images from volume data by using computer-aided concepts has found less consideration. Besides using a specific overall drawing style, e. g., by the definition of a certain color palette, anatomical illustrations often apply special techniques to disclose complex spatial relationships. In this context one of the main problems is to deal with occlusions of important parts of a data set. A concept frequently used by anatomical illustrators, but not often considered in visualization, is deformation: soft tissue is cut and flipped open to allow an occlusion-free view onto underlying structures, organs are twisted to visualize their shape, or blood vessels are pulled using pointed hooks for accentuation.

In this paper we propose an interactive system which supports real-time cutting and deformation of a full resolution volume object, and thus enables the user to achieve effects similar to those found in anatomical illustrations. Since the volume data visualized corresponds to real physical objects, the deformations should have a physical foundation, possibly using voxel intensities to specify material properties such as elasticity. However, because the main focus is on the interactive generation of cuts and deformations but not on simulation, physical realism has to be balanced with needs of interactivity and artistic freedom. An example of an illustration generated by using our system is shown in Figure 1.

The remainder of this paper is structured as follows. Section 2 examines work related to deformation and illustrative rendering of volumetric data. In Section 3 we present a classification of typical deformations found in anatomical atlases. The methods used for interactive deformation and rendering are described in Section 4. Section 5 presents interaction techniques for applying deformations to real-world data. Results are discussed in Section 6, and Section 7 concludes the paper.

2 RELATED WORK

Before introducing our approach, we discuss the handling of occlusions in illustrative visualization, algorithms for volume deformation and rendering, and related interaction techniques.

2.1 Occlusion in Illustrative Visualization

Different techniques are used in medical or technical illustrations to enable insights into occluded regions. *Cutaway views* [Die03] represent the most basic form

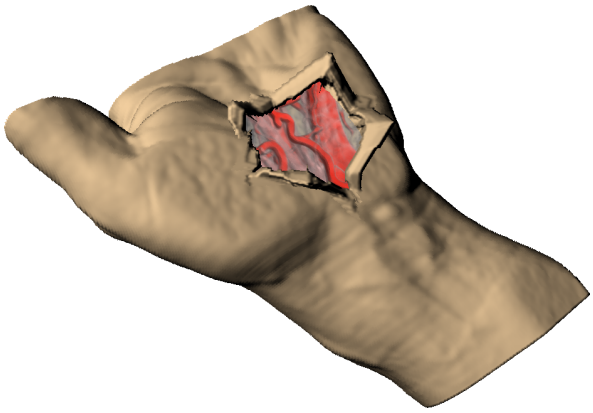


Figure 1: Cutting and flipping open the hand using our deformation and interaction techniques.

of simplifying a structure for illustration by discarding irrelevant or obscuring parts of an object. In a medical context this corresponds to the removal of (parts of) organs or tissue. In general use, simple cutting planes are very common while more complex cutaway shapes are found less frequently. *Ghosting* [Bru05] may be interpreted as an extension to cutaway views, where parts of an object are not completely removed but rather drawn with a different degree of transparency, which retains more context information. Most often found in technical illustrations, *exploded views* [Bru06] separate parts of an object by moving them to different positions. This enables the viewer to comprehend spatial relationships and, as in assembly instructions, get a mental image of how parts fit together. With *deformations* not an object's position but rather its shape is modified to allow an unobscured view onto the deformed object itself or other objects. Deformations are frequently used in medical illustrations.

2.2 Deformation Models

There exists a variety of deformation algorithms but most of them work on meshes rather than directly on volume data. Most are not suitable for real-time applications when using volume data sets of realistic sizes. An overview of deformation models in computer graphics is given by Gibson and Mirtich [Gib97b], and more recently by Chen et al. [Che05] and Nealen et al. [Nea05].

A well-established geometry-based model is *Free-Form Deformation* by Sederberg and Parry [Sed86], where the object to be deformed is placed in a lattice of grid points and the user manipulates these control points. Correa et al. [Cor06] use predefined, purely geometric deformation operators for illustrative visualization, without consideration of physical properties. *Mass-spring systems* represent a popular simplified physical model and have been widely used for modeling deformable objects. The *Finite Element Method* (FEM) approximates a continuous function which satisfies some equilibrium expression. This leads to very accurate results and therefore FEMs are in widespread use for modeling deformations, including soft tissue deformation. But this accuracy has a strong negative impact on performance when processing large objects.

All these methods are problematic with regard to the physical model, computational requirements, flexibility, or accuracy, making them unsuitable for real-time deformation of medical volume data for our use case. The 3D ChainMail algorithm as introduced by Gibson [Gib97a, Sch98] uses a different approach, solving most of these problems. The algorithm is based on a *linked volume* representation in which each element of a volume data set is directly linked to its six neighbor elements, and simple constraints are defined for these links. To model elastic material correctly, additionally a relaxation algorithm has to be applied to the results of the ChainMail algorithm, which iteratively tries to minimize an object's energy configuration. While more physically-*inspired* than physically-based, the algorithm has the advantage of being very fast even for large data sets, because it works locally. Successful usage in several surgery simulation systems [Gib97c, Sch98] shows that it can produce reasonable results—even though they might not be physically correct.

2.3 Rendering of Deformed Volumes

While hardware-accelerated volume rendering using texture slices [Cul94] is still prevalent in visualization applications, GPU-based ray-casting [Krü03] is much more flexible and becomes feasible due to significant improvements in GPU shader performance. This method first renders a proxy geometry in order to generate textures containing the *entry and exit points* (EEP). Pixel colors in the EEP textures encode start and end positions of rays which are then casted through the volume using a fragment shader.

The straight-forward approach for rendering deformed volumes is to directly apply the deformation algorithm to the data set and to render the resulting deformed volume. This revoxelization or *forward-mapping* requires resampling to ensure voxels are positioned in a rectilinear grid which is required for most rendering methods. Schulze [Sch07] presents such a system based on the ChainMail algorithm, which supports efficient deformation for small parts of large data sets only. Besides the high computational costs for resampling, additional memory needed for the deformed volume and resampling artifacts are problematic for general use.

An alternative approach is applying the deformation implicitly during rendering with *backward-mapping*. Kurzion and Yagel [Kur97] introduced the concept of *ray deflectors* which deform view rays during the rendering process rather than the volume data itself. Their purely geometric algorithm has the disadvantage that specifying deformations is not intuitive and, as complex deformations have to be modeled by several ray deflec-

tors, performance may become an issue. Rezk-Salama et al. [RS01] present a similar approach where the volume object is adaptively subdivided into a set of sub-cubes, and the deformation is then specified by transforming texture coordinates.

Revoxelization is flexible and should give good quality results when choosing a sophisticated resampling method, but performance is expected to be inadequate in the general case. In addition, it is uncertain whether optimization can help in practice without restricting the model to only small deformations. Speed is an issue for the ray deflector solution. Furthermore complexity of implementation and usage, together with a lack of a physical model, makes this method unsuitable.

2.4 Interaction

Virtual tools resembling scalpels, scissors, and forceps and their associated interaction metaphors have the advantage of being understood easily. But relying on "real" tools also limits interaction in the virtual system to what is possible in the physical world, preventing the system from exploiting the full potential of a computer-based solution. Therefore the system by McGuffin et al. [McG03] uses easily understandable interaction tools which are based on physical instruments and have been extended to allow a more light-weight interaction style. Correa et al. [Cor07] present a deformation method which requires only two-dimensional user input, but restricts the possible deformations.

3 MEDICAL ILLUSTRATION

In this section we analyze and describe the usage of deformation in anatomical atlases.

When classifying common types of deformations found in medical illustrations we exclude (piece-wise) rigid deformations, such as exploded views, which are easier to model and found more often in technical than in medical illustrations. Three main types of non-rigid deformations can be distinguished: pulling, turning, and cutting open. When an object such as a muscle or a vein occludes another object lying further behind, instead of cutting it away completely it can be *pulled* a small distance to provide an unobstructed view. An entire object can also be *turned* to provide an unobstructed view or to show it from the back. To show the internal structure of an object, it can be *cut and flipped open* in one or more directions. Finally, different deformation types can be combined to form a more complex deformation.

This classification was developed by examining deformations found in three common anatomical atlases, the classical atlases by Netter [Net97] and Sobotta [Sob05], and the newer Prometheus atlas [Sch05] which was drawn using 2D graphics applications. All three are regularly used in anatomy courses. It is notable that their illustration plates do not make use of transparency or ghosting techniques, whereas cutaways are used ubiquitously. Table 1 shows the results of analyzing the illustrations which make use of deformations; listed are the numbers of occurrences for the different types of deformations. However, it should be noted that many illustration plates show basically the same scene with only minor variations. Therefore the numbers can only give a rough hint about the actual use of deformation techniques. Nevertheless it can be concluded that the simple pulling deformations dominate. Cutting/flipping is found less often, perhaps because simple cutaways which remove parts of an object without any deformation are very common. For the anatomical layman turning is somewhat difficult to distinguish from pulling, since the two deformation types mainly differ in the illustrator's intention, not the visual result.

Source	Plates	Pull	Turn	Cut/Flip
Netter	29	19	6	6
Sobotta	57	33	7	23
Prometheus	37	22	16	15
Total	123	74	29	44

Table 1: Types of deformations found in some common anatomical atlases. Given are the total number of illustration plates containing deformations and the fraction of specific deformation types. Some plates contain more than one type of deformation.

4 OUR APPROACH TO VOLUME DEFORMATION

Based on the classification described in the previous section we have designed a system for interactively creating such deformations from medical volume data.

The following requirements for the system were derived from analyzing classical illustration techniques and methods for deformation and visualization. The goal is physically *comprehensible* but not necessarily physically *correct* behavior. It is the user's task to ensure a realistic result. This gives a certain degree of freedom that a strict physical model would possibly prevent. Our main contribution is the introduction of intuitive interaction concepts supporting the user by reducing complexity of six DoF interactions while still allowing interactive frame rates. Thus, we had to ensure that deformation and visualization achieve interactive frame rates without time-consuming preprocessing, while the full data set resolution is maintained. Full resolution is important since sometimes fine structures of an object need to be deformed, and down-sampling could make this impossible. Finally, since medical illustrations usually do not utilize transparency and ghosting, support for rendering of semi-transparent objects is optional.

4.1 Proxy-Geometry Deformation

An important aspect of deformation is the visual representation. In order to achieve interactive frame rates,

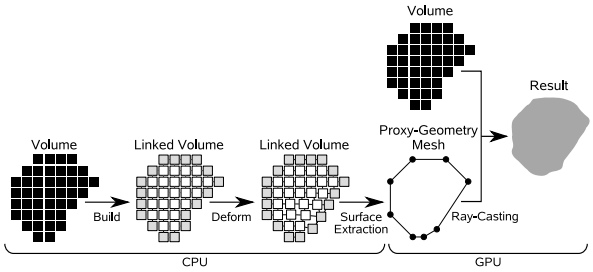


Figure 2: Overview of our deformation and rendering approach. Based on the data set a linked volume is generated and deformed before a surface mesh is extracted and used as proxy geometry for ray-casting.

we combine GPU-based ray-casting [Krü03] with the ChainMail algorithm. A conceptual overview of our approach is depicted in Figure 2.

With the exception of cutting operations, the ChainMail algorithm always keeps a static list of elements which form the object's surface. Hence, no additional computation is necessary to extract the surface elements after each deformation step. GPU-based ray-casting usually makes use of a box as proxy-geometry for creating the entry and exit points for each ray. Since all contributing voxels are located in the volume that is contained within the object's surface, the box proxy-geometry can be replaced by the object's surface without changing the rendering algorithm. When doing so, deforming the proxy-geometry has the effect of a space transformation, as it modifies the entry and exit points which control the position and direction of rays casted during rendering.

4.2 Functional Units in Proxy-Geometry Deformation

In the following we describe how we utilize the ChainMail algorithm, how we extract the object's surface from the deformed data structure, how we generate entry and exit points for the rendering, and how we ensure correct shading for deformed volumes.

ChainMail Deformation The linked volume data structure can be created directly from a volume given as a three-dimensional array. Neighborhood information is given implicitly by the rectilinear array. When the user has selected one or more elements for movement, the algorithm tests for constraint violations, while working on each element at most once per deformation step. For non-homogeneous material, a mapping from the intensity saved for each voxel to certain material properties can be made.

While the soft tissue found in medical volume data sets often shows elastic properties, adding elastic relaxation does not automatically provide a realistic simulation, as human tissue may show a more complex behavior (see e. g. Fung [Fun05]). For illustrative applications elasticity is not needed in general. Just like

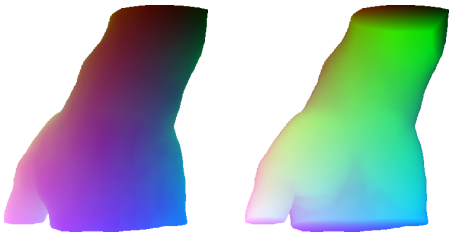


Figure 3: Entry and exit points textures for a proxy-geometry generated from the hand data set.

artists prefer modeling clay to a rubber-like material, elastic relaxation may interfere when artistic freedom is more important than physical realism.

Making cuts into the volume is realized by simply removing the link between two adjacent elements and adding them to the list of surface elements. This modifies the logical object surface, but no other modifications or special actions are necessary.

Surface Extraction For extracting a surface mesh from the linked volume, the marching cubes algorithm [Lor87] seems appropriate, although its computational requirements are quite high, as the basic marching cubes algorithm cannot make use of the available information about elements located on the object surface. However, without considerably increasing the number of generated triangles the marching cubes algorithm cannot ensure that the generated proxy-geometry completely encloses all object voxels, and thus for our case marching cubes is unsuitable.

Taking into account all information about surface elements and the grid-like structure of the data a simpler approach can be chosen. Although deformations may change the overall object shape, the ChainMail constraints limit relative movement of adjacent elements and ensure that the deformed linked volume may still be treated as a rectilinear grid. Our surface extraction algorithm successively looks at the ChainMail structure from all six principal directions and searches for cycles formed by four connected surface elements, for each adding a quad to the output geometry while removing redundant quads. Although the computation time of the surface extraction algorithm is proportional to the number of surface elements, which is significantly lower than the total number of volume elements, it could still be difficult to reach interactive performance for large volumes. Fortunately, it is not necessary to run the surface extraction after every deformation step. The relation between a volume element in the ChainMail structure and a vertex in the extracted surface persists as long as no topology changes take place. Therefore only the positions of vertices corresponding to displaced volume elements have to be updated. After a cut has been made the surface extraction must be repeated, taking into account the modified list of surface elements.

Entry and Exit Points Generation The entry and exit points are generated from the proxy-geometry created in the previous surface extraction step, as shown in Figure 3. Each vertex in the proxy-geometry surface relates to a specific voxel of the volume. To generate entry points, the geometry is rendered with the associated original voxel positions encoded as vertex colors. These colors are linearly interpolated over the entire surface by the graphics hardware. When a volume element is moved by deformation, the associated vertex is moved as well. But its color still encodes the original position of the volume element, while for each fragment in the entry points image of an undeformed proxy-geometry the fragment position and the position encoded in the fragment color are equal.

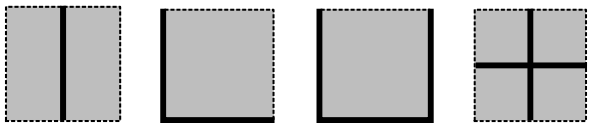
The depth buffer can be used to retrieve all fragments with maximum z-value to get exit points and with minimum z-value to get entry points. To be able to use transparency in the volume rendering, a more complex *depth-peeling* approach [Eve02] with multiple rendering passes would have to be incorporated, but this feature is not required for our use case. After generating entry and exit points the standard ray-casting process can be used without any modifications. For an undeformed volume the visual result is exactly the same as with a simple cube proxy-geometry.

Shading Ray-casting allows to apply shading to the rendering by deriving normal vectors from volume gradients. These normals are then used within a local illumination model for each sample along a view ray. While giving good results for an undeformed volume, this technique is not applicable after deformation. The problem is that gradients would have to be recalculated after each deformation, but this is not possible since the deformation is applied only implicitly during rendering. Gradient calculation requires analyzing neighboring voxels, but after deformation it is hard to localize a voxel's neighbors. When using the old gradients, all deformed surfaces would be shaded as if they were still located in their original position which results in shading artifacts.

Consequently, a different shading approach must be chosen; more specifically an improved *normal estimation* is necessary. Yagel et al. [Yag92] have analyzed several image-space algorithms for calculating normals based on a depth image. An approach using averaged forward differences reduces sensitivity to noise and discretization artifacts:

$$\tilde{n}_{i,j} = \left(\frac{1}{N} \sum_{k=1}^{N} P_{i+k,j} - P_{i,j} \right) \times \left(\frac{1}{N} \sum_{k=1}^{N} P_{i,j+k} - P_{i,j} \right),$$

with $\tilde{n}_{i,j}, P_{i,j} \in \mathbb{R}^3$, and where $P_{i,j}$ is the position of the fragment at screen coordinates (i, j). As an optimization, it is decided on a per-fragment basis whether to use gradients or normal estimation. When a ChainMail element is moved by deformation, this is encoded in the alpha value of the corresponding vertex in the proxy-geometry, influencing the alpha value of the resulting EEP fragments and therefore selecting normal calculation in the ray-casting shader. Figure 9 shows the successive images created and used during the shading process. Notice the incorrect normals for the deformed part of the object in Step 2 in contrast to the correct estimated normals in Step 4.



(a) single **(b)** L-form **(c)** U-form **(d)** cross

Figure 4: Some of the predefined cutting templates.

5 INTERACTION

In our system, user interaction mainly consists of two operations: specifying and applying a cut, as well as deforming the newly cut object. These operations are usually applied sequentially and can therefore be handled independently during the interaction process.

The common problem is that a three-dimensional object is transformed, while the user only sees a two-dimensional projection onto the screen and is using a two-dimensional input device, i. e., the mouse. In systems for CAD or 3D modeling, this lack of depth information is often solved by splitting the screen into four views, one orthogonal to each principal axis, plus a three-dimensional view. But this is unsuitable and unintuitive for a visualization application where only a single main view is in use.

Cutting As mentioned above, only a small number of different cuts are actually used in anatomical atlases. McGuffin [McG03] notes that there is "at least anecdotal evidence" that anatomists prefer to remove tissue semantically layer by layer, rather than making arbitrary planar cuts. This claim is supported by our findings described in Section 3. Therefore the user interface can be simplified by just providing a small set of predefined cuts, specified by *cutting templates*, which may be resized and positioned freely within the scene. These templates share some resemblance with a cookie cutter, with the main difference that they do not start cutting as soon as they touch an object, but only when the user explicitly initiates the cut. Some of the cutting templates defined in our system are shown in Figure 4.

An easy way for realizing placement of cutting templates would be dragging an associated 3D widget for translation and rotation, as shown in Figure 5 (left). While this permits unrestricted placement, such a user interface may get cumbersome as it requires several changes of perspective to verify the correct position. Since most often the cutting templates need to be placed on an object surface, a geometric constraint can simplify this task. When activated, clicking on the 2D rendering

of the volume places a cut onto the surface, centered at the position of the picked voxel. Its orientation is calculated from the voxel's gradient, giving a surface normal. After the initial placement the cutting template may be dragged and moved to a different position, while always remaining on the object's surface. The implementation of the cut placement requires only two 2D images as input: a rendering where each pixel encodes the position of the first non-transparent voxel hit by the ray, and a similar rendering encoding the voxel's normalized gradient, i. e., the normal. Both images can be created simultaneously in a single rendering pass using OpenGL's multiple render targets extension, see Figure 9 Step 2 and 3 for an example.

In order to superimpose a cut on a surface, the cutting template not only has to be placed and oriented correctly, but also has to be fitted to the shape of the surface. In the following we only consider two-dimensional cutting templates consisting of cutting lines which are fitted to the surface and extruded later on to obtain the desired three-dimensional cutting template. Usually a cut is determined by several cutting lines on the surface of the object which meet in a common center point, similar to spokes of a wheel which meet in its center. First the (planar) cutting template is positioned on the object's surface in such a way that the center point coincides with the location selected by the user, and then it is oriented according to the surface normal at this position. In general only the common center point of the cutting lines will lie on the surface, but not the entire cutting line. Since no neighborhood information can be retrieved directly from a surface voxel, we use an image-based technique to map the cutting lines to the surface efficiently, as shown in Figure 6. For each cutting line its start and end point is projected to screen space and the next pixel on the line connecting these two points is sampled. Based on this sampling the position and the normal of the current pixel are retrieved and taken into account when reprojecting the end point. This reprojection is necessary in order to allow an alignment of the lines along the surface. The reprojection process is repeated for further pixels until the length of the 3D polygonal chain formed by the corresponding points in voxel space exceeds the length of the cutting line. The cut is determined by the polygonal chain constructed during this process.

The line segments forming the polygonal chain are extruded in the direction opposite to the surface normal to create the desired 3D cutting template, while the user controls the amount of extrusion and thereby the cutting depth. Due to noise in normal calculation for adjacent surface voxels, multiple normal vectors are averaged to give the direction for this extrusion. This results in the final 3D cut which can then be applied to the data.

Deformation The next step after a cut has been applied is the specification of the desired deformation. A point-and-drag interface is intuitive and suitable for simple

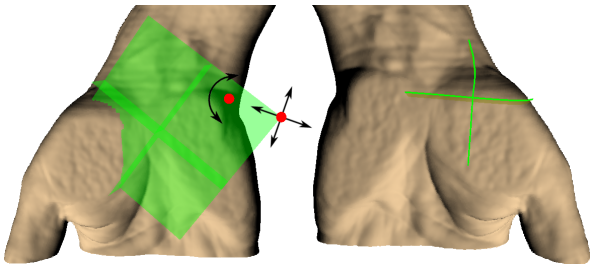


Figure 5: Placing a cutting template using 3D widgets (left) and our surface-based placement technique (right).

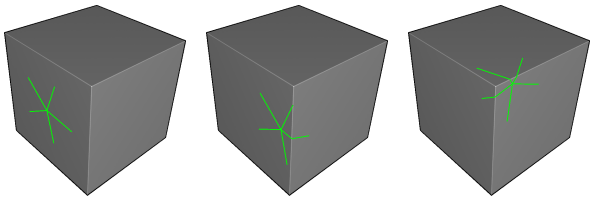


Figure 6: Surface-based cutting template placement.

deformations such as pulling. For more complex cases the 2D interface proves unsuitable, as the deformation has to be performed in multiple steps. After each step the camera orientation has to be changed in order to check the deformation achieved so far. This becomes even more difficult when multiple points have to be dragged simultaneously. *Deformation templates* can help to solve this problem. Similar to cutting templates they can be moved in the volume specifying a certain type of deformation with several editable parameters. For example, a deformation template for a flip-open deformation would be used after a corresponding cutting template has been applied, with parameters such as the number of incisions, size, and amount of aperture. The deformation is determined by control points on the object's surface which are initially placed close to the center point of the template, shown in the leftmost image in Figure 10. When the user specifies the amount of deformation, the points head in different directions, each moving on a circle segment, and thus pulling the ChainMail structure and applying the deformation. This is demonstrated by the image sequence in Figure 10.

6 RESULTS

In Figure 1 a surface-based cutting template with four cutting lines was applied to a hand data set before the skin was deformed using a deformation template. Phong shading is used for rendering the skin, while the internal structure with blood vessels is rendered in a second pass using cel-shading and a different transfer function. The images from the two passes are merged by taking into account the calculated depth values for each pixel. A simpler *pulling* deformation of the same data set is depicted in Figure 7. Here the cutting template shown in Figure 4(a) was manually placed on the object; the cut was then dilated by pulling on both sides. For Figure 8

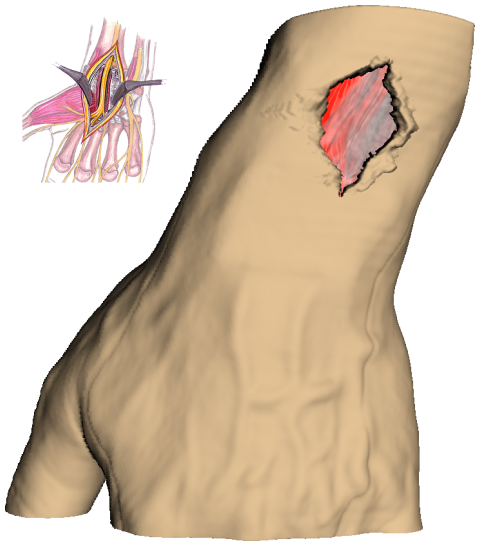


Figure 7: Applying a single cut with manual deformation using the point-and-drag interface, compared to a similar hand-drawn illustration (by Howell MediGraphics, `http://www.medigraphics.com`).

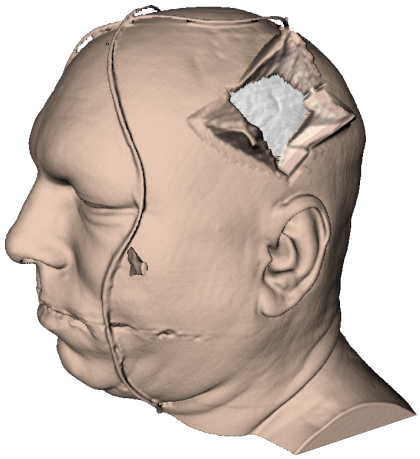


Figure 8: The head of the Visible Human (courtesy of the United States National Library of Medicine) is flipped open using a three-way cutting template.

a surface-based cutting template with three cutting lines was applied to the head of the Visible Human data set, and then manually deformed.

The hand data set consists of $256 \times 128 \times 256$ voxels, from which a ChainMail structure with 2,710,516 elements was generated. About 4.3% of those elements lie on the surface, therefore the surface-extraction algorithm generates a proxy-geometry with 693,200 vertices. The head data set with 256^3 voxels resulted in 6,040,852 ChainMail elements, 3.9% on the surface resulting in 1,529,000 vertices. Providing meaningful frame rates when performing a deformation is difficult, but it is safe to say that the time needed for deformation depends linearly on the number of ChainMail elements involved. For all scenes and deformations shown in this paper the system stayed responsive and supported smooth interaction. Frame rates for rendering have been mainly unaffected by the deformation, the hand data set reaching 38 FPS, the head data set 23 FPS, for a rendering with 512×512 pixels. All tests were conducted on a machine with an Intel Core 2 Duo E6300 CPU, 2 GB of RAM, and an NVIDIA GeForce 7900 GTX graphics board.

7 CONCLUSION

In this paper, techniques for generating interactive computer-based medical illustrations have been introduced, with the focus on interactively applying deformation to anatomical objects.

The deformation technique developed in this paper utilizes the 3D ChainMail algorithm to deform a volume object and renders the result using ray-rasting. It has been shown that the 3D ChainMail algorithm can give real-time results for reasonably sized data sets. The simple yet flexible data structure has proven useful for applications besides deformation, such as cutting and surface extraction. Some deformations can make use of the physically-inspired behavior of the linked volume, especially pulling, because they are similar to the ChainMail deformations, where a given element is grabbed and pulled. With adequate data sets, material properties based on real data can be used to define deformation constraints for the 3D ChainMail algorithm, exploiting its support for non-homogeneous material. The advantages of a more realistic physical model (e. g. FEM or mass-spring systems) could also be evaluated, but a good balance between realism and interactivity is necessary. However, it seems that current methods cannot fulfill these requirements.

Although the focus was on techniques for interactively performing cuts as well as deformations, a rendering approach has been presented which applies a deformation to the proxy-geometry which is used by GPU-based ray-casting. This technique allows to include the system into an existing ray-caster, without notable impact on the overall rendering performance. The implemented simple surface extraction algorithm is insufficient for surfaces of objects consisting of only few voxels, e. g., nerves or skin, nor can the 3D ChainMail algorithm accurately deform such objects. While the presented rendering technique allows to interactively change all deformation parameters, the image quality could be improved. An improvement would be achieved by resampling the ChainMail data structure into a new volume data set, which could be rendered using standard volume rendering techniques. Although this approach would lack interactivity, it would be suitable for rendering a single final image after all deformation parameters have been specified interactively. Finally, the ChainMail algorithm could be improved to not produce the visible deformation artifacts due to the rectilinear composition of the used data structure.

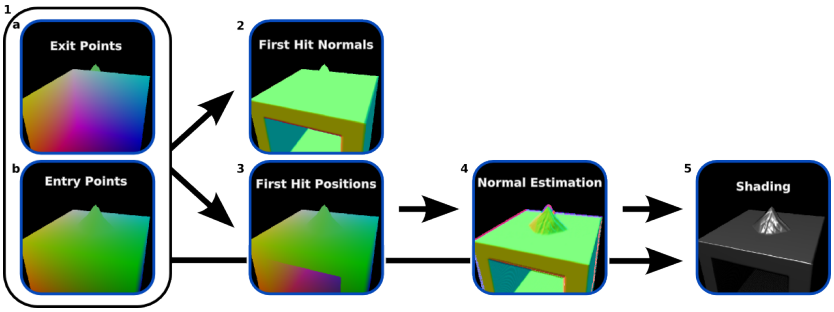


Figure 9: Shading process with normal estimation.

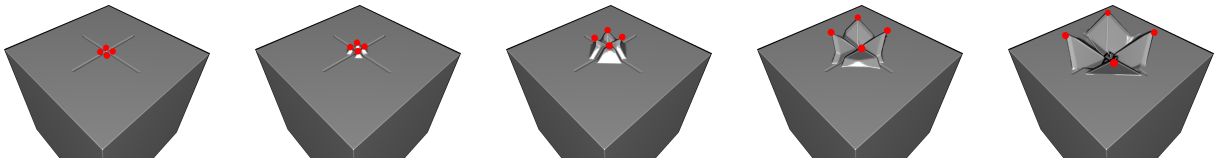


Figure 10: Applying a deformation template on a cube consisting of 128^3 voxels.

ACKNOWLEDGEMENTS

This work was partly supported by grants from Deutsche Forschungsgemeinschaft (DFG), SFB 656 MoBil Münster, Germany (project Z1). The presented concepts have been integrated into the VOREEN volume rendering engine (`http://www.voreen.org`).

REFERENCES

[Bru05] Bruckner, S. and Gröller, M. E. VolumeShop: An interactive system for direct volume illustration. In *Proceedings of IEEE Visualization 2005*, pp. 671–678. 2005.

[Bru06] Bruckner, S. and Gröller, M. E. Exploded views for volume data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1077–1084, 2006.

[Che05] Chen, M., Correa, C., Islam, S., Jones, M. W., Shen, P.-Y., Silver, D., Walton, S. J., and Willis, P. J. Deforming and animating discretely sampled object representations. In *Eurographics 2005, State of the Art Reports*, pp. 71–94. 2005.

[Cor06] Correa, C., Silver, D., and Chen, M. Feature aligned volume manipulation for illustration and visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1069–1076, 2006.

[Cor07] Correa, C., Silver, D., and Chen, M. Volume deformation via scattered data interpolation. In *Proceedings of Eurographics/IEEE VGTC Workshop on Volume Graphics 2007*, pp. 91–98. 2007.

[Cul94] Cullip, T. J. and Neumann, U. Accelerating volume reconstruction with 3D texture hardware. Technical report, University of North Carolina at Chapel Hill, 1994.

[Die03] Diepstraten, J., Weiskopf, D., and Ertl, T. Interactive cutaway illustrations. *Computer Graphics Forum*, 22(3):523–532, 2003.

[Eve02] Everitt, C. Interactive order-independent transparency. Technical report, NVIDIA Corporation, 2002.

[Fun05] Fung, Y.-C. *Biomechanics: Mechanical Properties of Living Tissues*. Springer-Verlag, 2nd edition, 2005.

[Gib97a] Gibson, S. 3D Chain Mail: A fast algorithm for deforming volumetric objects. In *Proceedings of 1997 Symposium on Interactive 3D Graphics*, pp. 149–154. 1997.

[Gib97b] Gibson, S. and Mirtich, B. A survey of deformable modeling in computer graphics. Technical report, Mitsubishi Electric Research Laboratory, 1997.

[Gib97c] Gibson, S., Samosky, J., Mor, A., et al. Simulating arthroscopic knee surgery using volumetric object representations, real-time volume rendering and haptic feedback. In *CVRMed-MRCAS '97*, pp. 369–378. Springer-Verlag, 1997.

[Krü03] Krüger, J. and Westermann, R. Acceleration techniques for GPU-based volume rendering. In *Proceedings of IEEE Visualization 2003*, pp. 287–292. 2003.

[Kur97] Kurzion, Y. and Yagel, R. Interactive space deformation with hardware-assisted rendering. *IEEE Computer Graphics and Applications*, 17(5):66–77, 1997.

[Lor87] Lorensen, W. E. and Cline, H. E. Marching Cubes: A high resolution 3D surface construction algorithm. In *Proceedings of SIGGRAPH '87*, pp. 163–169. 1987.

[McG03] McGuffin, M. J., Tancau, L., and Balakrishnan, R. Using deformations for browsing volumetric data. In *Proceedings of IEEE Visualization 2003*, pp. 401–408. 2003.

[Nea05] Nealen, A., Müller, M., Keiser, R., Boxerman, E., and Carlson, M. Physically based deformable models in computer graphics. In *Eurographics 2005, State of the Art Reports*, pp. 113–140. 2005.

[Net97] Netter, F. H. *Atlas der Anatomie des Menschen*. Thieme, 1997.

[RS01] Rezk-Salama, C., Scheuering, M., Soza, G., and Greiner, G. Fast volumetric deformation on general purpose hardware. In *HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, pp. 17–24. 2001.

[Sch98] Schill, M., Gibson, S., Bender, H.-J., and Männer, R. Biomechanical simulation of the vitreous humor in the eye using an enhanced ChainMail algorithm. In *MICCAI '98*, pp. 679–687. Springer-Verlag, 1998.

[Sch05] Schünke, M., Schulte, E., Schumacher, U., Voll, M., and Wesker, K. *Hals und Innere Organe*. Prometheus Lernatlas der Anatomie. Thieme, 2005.

[Sch07] Schulze, F., Bühler, K., and Hadwiger, M. Interactive deformation and visualization of large volume datasets. In *2nd International Conference on Computer Graphics Theory and Applications (GRAPP) 2007*, pp. 39–46. 2007.

[Sed86] Sederberg, T. W. and Parry, S. R. Free-form deformation of solid geometric models. In *Proceedings of SIGGRAPH '86*, pp. 151–160. 1986.

[Sob05] Sobotta, J. *Kopf, Hals, obere Extremität*, volume 1 of *Atlas der Anatomie des Menschen*. Urban & Fischer, 21st edition, 2005.

[Yag92] Yagel, R., Cohen, D., and Kaufman, A. Normal estimation in 3D discrete space. *The Visual Computer*, 8(5/6):278–291, 1992.