

# Direct GPU-based Volume Deformation

Friedemann Rößler<sup>1</sup>, Torsten Wolff<sup>1</sup>, Thomas Ertl<sup>1</sup>

<sup>1</sup> Universität Stuttgart, Visualization and Interactive Systems Institute, Stuttgart, Germany

contact person: friedemann.roessler@vis.uni-stuttgart.de

## Abstract:

*Fast and realistic soft-tissue deformation is an important feature for medical simulation environments. In this work we present a direct deformation approach for volumetric datasets which is completely performed on the GPU. It takes on the one hand advantage of a GPU's computing power; on the other hand deformation is easily combined with hardware-supported rendering. Our deformation technique is based on the physically-inspired 3D ChainMail algorithm and directly works on the original volume dataset. We propose a deformation pipeline which handles all steps of manipulation, deformation and volume visualization.*

*Keywords: Volume Deformation, ChainMail Algorithm, Volume Visualization, GPGPU*

## 1 Purpose

Deformable models are often used in medical applications, e.g. virtual surgery, for the realistic simulation of the behavior of soft tissue. Physically-based approaches, like mass-spring systems or finite element methods (FEM), usually use an explicitly generated model which is deformed by expensive computations. For visualization the deformed model has to be additionally transformed into a renderable representation. Altogether, the deformation process is very complex and time-consuming. To overcome these drawbacks we developed a deformation technique that directly acts on the originally acquired volume data of a patient without the need of expensive preprocessing. It is based on the 3D ChainMail algorithm [1] which has the advantage of low computational costs. Our approach is similar to a method presented by Schulze et al. [2], but, in contrast to that, our algorithm is completely performed on the GPU. It includes the whole process of manipulation, deformation and volume rendering. This exploits on the one hand the fast parallel computing capabilities of modern graphics hardware. On the other hand, it avoids the expensive transfer of deformation information to the GPU for visualization.

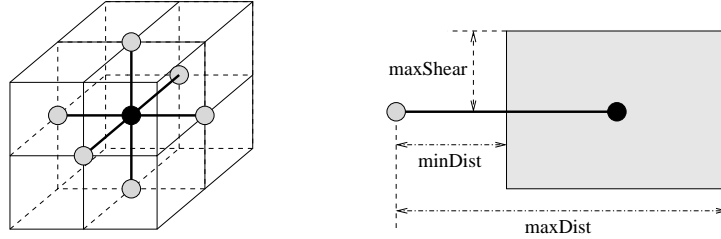
The interactive manipulation and deformation of volumetric objects is an active research field. Chen et al. [3] give a good overview about this topic. They present several geometrical and physically-based approaches and show their usage in medical and non-medical applications. There exists also some work on deformation computation on the GPU. Georgii et al. [4] presented a GPU-based mass-spring deformation system which they combined with standard surface rendering. Tejada and Ertl [5] proposed a similar approach but perform direct volume rendering on the deformed tetrahedral grid.

## 2 Methods

### 2.1. The 3D ChainMail Algorithm

The 3D ChainMail algorithm operates on deformation elements which are initially arranged on a three-dimensional regular grid. Each element is connected to its six nearest neighbors in x-, y-, and z-direction (see Fig. 1 left). The relative position of a grid element to its neighbors is governed by several constraints that give the minimally and maximally allowed distance and the maximally allowed shear (see Fig. 1 right). To simulate anisotropic deformation these limits can differ for each neighbor.

The algorithm works as follows: It starts with a single moved grid element. The neighbors of this element are checked if they still satisfy the ChainMail constraints. If not, the affected neighbors are minimally moved to fulfill the constraints again. Then the algorithm goes on with the newly moved elements and tests their neighbors in the same way. This procedure is repeated until the list of moved elements is empty. Gibson has shown that each ChainMail element has to be touched only once if the ChainMail constraints are constant throughout the volume [1]. However, the first-moved-first-processed propagation order does not create valid grid configurations for inhomogeneous deformation constraints. For this reason, Schill et. al. [6] presented an enhanced ChainMail algorithm which first processes the elements with the largest displacement. This propagates the deformation information faster through stiffer material, analogous to the broadening of a shockwave.



**Fig. 1:** The 3D ChainMail Grid: (left) the six neighbors (gray) of a grid element (black); (right) the area (grey rectangle) of valid positions of an element (black) relative to its left neighbor (grey).

## 2.2. Mapping Deformation and Visualization to the GPU

For GPU-based ChainMail deformation we store the 3D deformation grid in a 3D texture where each texel holds the displacement vector of the corresponding grid element. To be independent of the volume's real dimensions all computations are performed on a normalized cube with edge length one. By this, the grid elements' original positions are implicitly given by the texture coordinates of the associated texels. Further on, the resolution of the deformation grid can be chosen independently of the original volume dataset.

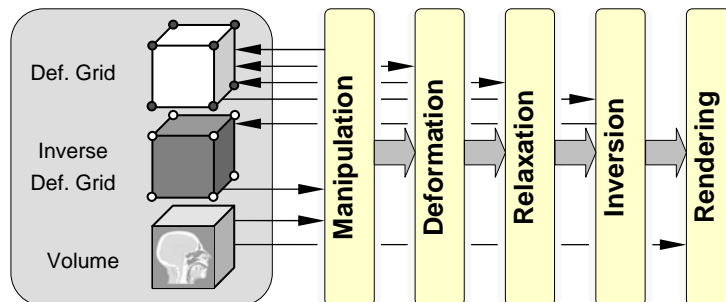
We use the OpenGL API and the high level shader language GLSL to perform all steps of volume deformation and visualization. The processing of a single step of the deformation algorithm is realized by directly rendering to the 3D deformation texture via a framebuffer object. To avoid inconsistencies during the parallel processing of the fragments it is not possible to simultaneously read and write from/to the same texture. Instead the deformation grid texture is duplicated and the two textures are used alternately for reading and writing (ping-pong rendering).

Our GPU-based deformation approach is based on the ChainMail Algorithm, but inverts the way of deformation propagation. Instead of actively displacing neighbor elements, an element checks if it violates a constraint to one of its neighbors and displaces itself if required. This is necessary, because the parallel pipeline of a GPU does not permit to write to any other fragment than the currently processed. Further, the processing order of the deformation elements is not fixed to exploit the fact that a huge number of grid elements are evaluated in parallel. Instead an element is processed each time when one of its neighbors was moved in the previous step. This is continued until no more element is displaced. This approach automatically copes with inhomogeneous deformation constraints.

There are several techniques for rendering of the deformed volume. One possibility is to perform a pre-rendering step which resamples the deformed volume on a regular grid, like it is done in [2]. Alternatively, the deformation grid can be directly used as proxy geometry for rendering [7]. We use a third method which keeps the proxy geometry undeformed and applies the deformation instead to the texture coordinates just before the lookup in the volume texture [8]. Since the deformation grid gives the forward directed displacement of a grid element in object space, it has first to be inverted to get the corresponding backward directed displacement in texture space. We do this inversion in a pre-rendering step and store the results in an inverse deformation grid. The advantage of this technique is that it can be used in combination with standard volume-rendering for regular grids without introducing resampling errors.

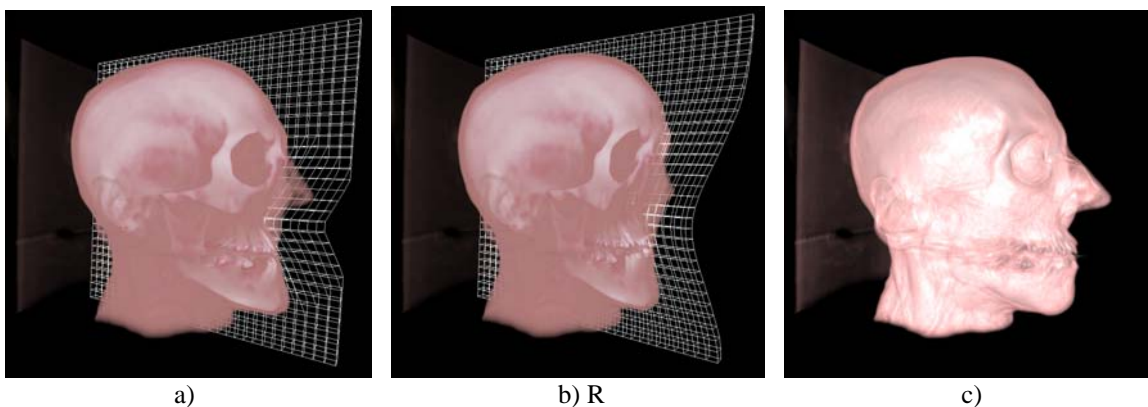
## 2.3. GPU-based Deformation Pipeline

We implemented a volume deformation pipeline which consists of five sequentially applied steps (see Figure 2), each performed by specific GLSL shaders. It basically acts on three data structures: the original volume dataset, the forward deformation grid, and the inverse deformation grid. In the following the five pipeline steps are detailed:



**Fig. 2:** The five steps of the deformation pipeline and the used data structures. The small arrows show for each pipeline step the data structures which are accessed for reading and/or writing.

1. **Manipulation:** In the manipulation step the current mouse move is mapped into normalized 3D model space and it is checked if a collision with the volume happens. For this, a single fragment is rendered and the applied shader is casting a ray from the start to the end position of the mouse move. At each sampling position the corresponding alpha value is read from the transfer function lookup table. If the alpha value is greater than zero a collision is found. Then the deformation process is initialized by moving the eight grid elements which are nearest to the collision position equal to the mouse move.
2. **Deformation:** The ChainMail deformation step is realized in multiple passes. In each pass it is checked for each grid element if any of its six neighbors was moved in the previous step. In this case, it is tested if one of the ChainMail constraints is violated and, if required, the current element is moved to the nearest valid position. The constraints are chosen with respect to a user-definable deformation transfer function. Since the deformation grid can have a lower resolution than the original volume dataset, the volume samples are looked up in a special tissue coefficient texture. This has the same resolution as the deformation grid and is generated from the original volume dataset by mipmap filtering. If no neighbor of an element is moved, the execution of the shader is directly discarded. This avoids, on the one hand, expensive computations for unaffected elements; on the other hand, it allows to check if the deformation grid has reached a geometrical equilibrium. For this, we use occlusion queries, which return the number of processed fragments during a rendering pass. The deformation step is further optimized by starting with the initially moved elements and incrementally enlarging the investigated grid region.
3. **Relaxation:** After the ChainMail deformation step, the deformation elements fulfill the geometric ChainMail constraints, but the linear displacement does not adequately simulate natural soft body behavior. For this reason an additional relaxation step has to be applied, which tries to move the grid elements to an energetic minimum. Currently, we have implemented a simple relaxation algorithm which was proposed in [9]. This iterative algorithm moves the grid elements in multiple rendering passes to the median of their six neighbor elements until an equilibrium is reached.
4. **Inversion:** For the computation of the inverse deformation grid we developed an iterative gradient-descent-like optimization algorithm, which searches for each element of the inverse grid the position which is mapped by the forward deformation to the element's fixed position. It starts with the grid element's original position and looks up at this position the forward displacement vector in the forward deformation grid. Then it computes the forward displaced position and compares this to the inverse grid element's position. If they differ, the inverse displacement position is adjusted and tested again. The adjustment should be done along the gradient of the error function, i.e. the length of the difference vector between the forward displaced position and the grid element's position. But to avoid expensive gradient computation we use the difference vector itself as an approximation of the gradient. Since ChainMail deformation and relaxation create a smooth grid configuration, the iterative search usually finds the global minimum of the error function. The GPU implementation of the inversion algorithm is straight forward. The iteration loop is placed inside the shader and the algorithm is performed in a single rendering pass. The forward displacement vectors that correspond to the intermediate inverse displacement positions are taken from the forward deformation texture by standard texture lookups with trilinear interpolation.
5. **Rendering:** For rendering of the deformed volume we use slice-based rendering with pre-integration. The modification to the standard algorithm is restricted to an additional trilinear lookup in the inverse deformation grid. The returned interpolated inverse displacement vector is given in normalized texture space and can directly be added to the current texture coordinate. For illumination the gradients can not be pre-computed, but have to be calculated on the fly.



**Fig. 3:** Three deformed human heads (256x256x225) with different deformation algorithms and grid sizes. a) 32x32, ChainMail only; b) 32x32, CM + Relaxation; c) 128x128x128, CM + Rel. + Illumination

### 3 Results and Discussion

We have tested our GPU-based deformation approach with a CT scan of a human head. Figure 3 shows three screenshots of the deformed head with different deformation grid resolutions and different stages of our algorithm applied. Figure 3a) and b) both present the deformation results for a grid resolution of  $32 \times 32 \times 32$  elements without (Fig 3a) and with (Fig 3b) relaxation. It can be clearly seen, that the additional relaxation creates a much smoother deformation result. In Figure 3c) a deformation grid of  $128 \times 128 \times 128$  elements is used and the volume is additionally illuminated. Here, the deformation below the nose affects a smaller region of the head.

Table 1 gives frame rates for deformation and rendering for different grid resolutions. The leftmost column shows the performance for rendering only. Standard volume rendering produces interactive frame rates, but additional illumination is four times slower. We restricted relaxation to five steps per frame and introduced instead intermediate relaxation if no interaction happens. A major advantage of our approach is that the resolution of the deformation grid is independent of the resolution of the original volume dataset. So the grid resolution can be chosen with respect to the desired locality of deformation and due to the required frame rate. Though, the memory consumption increases significantly with higher grid resolutions. A deformation grid with  $128 \times 128 \times 128$  elements and four 32-bit floating point values per element needs 32 MB, a  $256 \times 256 \times 256$  grid needs 256 MB. Since our algorithm needs the grid three times, for ping pong rendering and for inverse deformation, the memory of the applied GPU is exceeded for a grid resolution of  $256 \times 256 \times 256$  elements. A possible solution to overcome this limit would be a bricking scheme like proposed in [2]

	Rendering only	16x16x16	32x32x32	64x64x64	128x128x128
ChainMail	19.5	16.5	15.0	13.5	8.9
CM + Relaxation	19.5	15.5	14.5	11.5	7.0
CM + Re. + Illum.	5.0	4.5	4.4	4.2	3.3

Table 1: Frame rates in frames per second (fps) for deformation of a CT-Dataset of a human head with resolution  $256 \times 256 \times 225$ , viewport  $512 \times 512$ , GeForce 8800 GTX with 756 MB RAM.

The frame rates in Table 1 show that our ChainMail-based deformation approach fits well to the parallel architecture of Gus. Since it is directly combined with standard volume rendering it can be easily applied to medical simulation applications. However, the ChainMail technique has still some drawbacks that should be eliminated. E.g., the achievable frame rates highly depend on the applied deformation constraints. If the constraints simulate stiff material the deformation is propagated over a larger area of the grid than with soft tissue constraints. Hence, the ChainMail computation is more expensive in this case. Nevertheless, the presented deformation pipeline allows to modify single steps without affecting the others. So deformation and relaxation could be replaced by more complex deformation models.

### 5 References

- [1] Gibson, S.F.F.: 3D ChainMail: A Fast Algorithm for Deforming Volumetric Objects, Proc. Symp. Interactive 3D Graphics, pp. 149-154, 1997
- [2] Schulze, F., Bühler, K., Hadwiger, M.: Interactive Deformation and Visualization of Large Volume Datasets, Proceedings of International Conference on Computer Graphics Theory and Applications 2007; pp. 39-46, 2007
- [3] Chen, M., Correa, C., Islam, S., Jones, M. W. , Shen, P.-Y., Silver, D., Walton, S. J., Willis, P. J.: 3D Manipulating, Deforming and Animating Sampled Object Representations, Computer Graphics Forum, vol. 26, no. 6, pp. 824-852, 2007
- [4] Georgii, J., Ehtler, F., Westermann, R.: Interactive Simulation of Deformable Bodies on GPUs, Proceedings of Simulation and Visualization 2005, pp. 247-258, 2005
- [5] Tejada, E., Ertl, T.: Large Steps in GPU-based Deformable Bodies Simulation, Simulation Practice and Theory, vol. 13, no. 9, pp. 703-715, 2005
- [6] Schill, M.A., Gibson, S.F.F., Bender, H.-J., Männer, R., Biomechanical simulation of the vitreous humor in the eye using an Enhanced ChainMail Algorithm, Proceedings of Medical Image Computation and Computer Integrated Surgery 1998, pp. 679-687, 1998
- [7] Rezk-Salama, C., Scheuering, M., Soza, G., Greiner, G.: Fast Volumetric Deformation on General Purpose Hardware, Proceedings of ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware 2001, pp. 17-24, 2001
- [8] Brunet, T., Evan Nowak, E. Gleicher, M.: Integrating Dynamic Deformation into Interactive Volume Visualization, Proceedings of EG/IEEE VGTC Symposium on Visualization 2006, pp. 219-226, 2006
- [9] Gibson, S.F.F.: Using linked volumes to model object collisions, deformation, cutting, carving, and joining, , IEEE Transactions on Visualization and Computer Graphics , vol.5, no.4, pp.333-348, 1999