Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

MORITZ EHLKE, HEIKO RAMM, HANS LAMECKER, STEFAN ZACHOW

Efficient projection and deformation of volumetric intensity models for accurate simulation of X-ray images

ZIB-Report 12-40 (November 2012)

Herausgegeben vom
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Takustraße 7
D-14195 Berlin-Dahlem

Telefon: 030-84185-0
Telefax: 030-84185-125

e-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

Efficient projection and deformation of volumetric intensity models for accurate simulation of X-ray images

M. Ehlke and H. Ramm and H. Lamecker and S. Zachow

Zuse Institute Berlin, Medical Planning Group, Berlin, Germany

Abstract

We present an efficient GPU-based method to generate virtual X-ray images from tetrahedral meshes which are associated with attenuation values. In addition, a novel approach is proposed that performs the model deformation on the GPU. The tetrahedral grids are derived from volumetric statistical shape and intensity models (SSIMs) and describe anatomical structures. Our research targets at reconstructing 3D anatomical shapes by comparing virtual X-ray images generated using our novel approach with clinical data while varying the shape and density of the SSIM in an optimization process. We assume that a deformed SSIM adequately represents an anatomy of interest when the similarity between the virtual and the clinical X-ray image is maximized. The OpenGL implementation presented here generates accurate (virtual) X-ray images at interactive rates, thus qualifying it for its use in the reconstruction process.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Hardware architecture—Parallel processing I.3.3 [Computer Graphics]: Picture/Image Generation—Viewing algorithms

1. Introduction

The reconstruction of a patient's 3D anatomy based on a single or a few X-ray images for diagnosis and advanced therapy planning has received increasing interest in recent years [LWH06, Zhe10, BKdB*11]. Compared to 3D image acquisition methods like computed tomography (CT), X-ray imaging is widely available and rather inexpensive.

The common concept of most existing reconstruction methods is to project many variations of a 3D shape onto an image plane with a known X-ray setup. Those projections are then compared to a patient's X-ray within an optimization framework. The projected model instance that best depicts the 2D shape in the X-ray image(s) is assumed to be the best approximation of the underlying 3D anatomy. However, the reduction of dimensionality occurring during the X-ray acquisition renders this process an ill-posed problem. Even for a human observer it can be hard to resolve ambiguities in X-ray images resulting from overlapping structures [SQNS12].

The goal is to generate realistically looking radiograph images that mimic the appearance of clinical X-ray images as good as possible to allow for a direct image-based comparison. At the same time, a suitable method should be able to produce large quantities of projections that are required during the optimization process.

This work is structured as follows: First, existing methods for 3D reconstruction from X-ray images are discussed that employ 2D projection images. We introduce the theory behind X-ray attenuation and how it can be modeled on 3D deformable tetrahedral grids in Section 3. Our GPU-based projection algorithm is presented in Section 4 and an OpenGL implementation including a method for fast deformation is outlined in Section 5. The implementation is evaluated and discussed in Sections 6 and 7. We finally conclude our work in Section 8.

2. Related Work

Existing approaches for the reconstruction of 3D anatomical models from X-ray data typically employ deformable surface models in combination with contour-based distance measures [DLvB*10, Zhe10, BKdB*11]. Contours can be computed very efficiently. However, contour matching turned out to be problematic since similar contours have to be determined in both the projected model and the clinical X-

ray images. An improvement would be to directly compare image intensities rather than contour information. One solution to incorporate intensity information is the projection of CT-like atlases. Steininger et al. [SFK08] for instance perform the projection of a deformable hexahedral grid of the proximal femur by means of a ray casting approach.

Lamecker et al. [LWH06] propose a method for 3D shape reconstruction from X-ray images by comparing a thickness projection of a deformable surface model to X-rays using image-based distance measures like mutual information. They conclude that this naive approach of modelling X-ray attenuation is not sufficient to model the heterogeneous density inside the bone. A similar approach for visualization purposes was presented by Vidal et al. [VGF*10]. They describe regions of different bone density by multiple surface models.

A more adaptive sampling of the interior densities of the anatomical structure can be achieved by the use of unstructured grids. Here, the size of the volume elements defines the quality of the sampling. There are a variety of GPU-based algorithms available for the visualization of unstructured grid data [GW06, MMF10, WKME03]. Those methods typically employ absorption-emission based lighting models that are not directly applicable to the simulation of X-ray attenuation.

Yao [Yao02] proposed an idea to provide a dense sampling of density values on low resolution tetrahedral grids by using higher-order polynomial functions. The deformable model is represented as a point distribution model (PDM) that additionally encodes density values learned from CT data. Yao utilized a CPU-based projection algorithm that respects the non-linear density distribution in each tetrahedron while simulating X-ray attenuation. Sadowsky et al. [SC06] seize the idea of Yao and simulate X-ray images using a Projected Tetrahedra approach that is partly implemented on the GPU. The deformation is performed on the CPU, consequently the geometry and intensity information has to be copied to the GPU memory after the model is altered.

Contribution: In this paper we present an extension to the work of Yao and Sadowksy et al. that generates virtual X-ray images from deformable volumetric density models. Density information is described as a higher-order polynomial function on each volumetric cell of an unstructured grid, allowing for accurate density representation even on coarse grids. Unlike previous methods, our approach is implemented entirely on the GPU, thus avoiding time consuming copy operations between GPU- and system-memory.

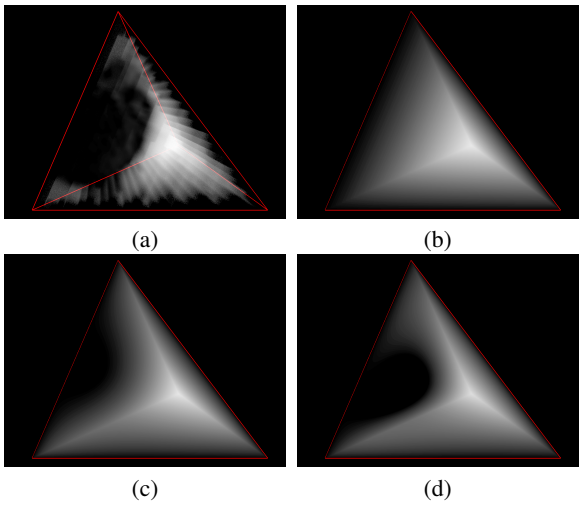


Figure 1: *Actual density distribution within a single tetrahedron acquired from a CT volume (a) and approximations using Bernstein polynomial density functions of degree $d = 1$ (b), $d = 2$ (c) and $d = 3$ (d). A higher polynomial degree better approximates the non-linear gray value distribution of the CT data.*

3. Background

3.1. Higher-order X-ray attenuation functions

The overall attenuation encountered by a monochromatic X-ray beam $p(x) = w_{in} + (w_{out} - w_{in}) \cdot x$ passing through a material is described by the Beer-Lambert law:

$$I_{out} = I_{in} \cdot e^{-\int_{p(x)} \alpha(w) \mathrm{dw}} \tag{1}$$

where I_{in}/I_{out} are the input/output intensities of the beam, w_{in} and w_{out} are the entrance and exit points, and $\mu = \alpha(w)$ denotes the linear attenuation coefficient of some tissue encountered by the ray at point w. The function $\mu = \alpha(w)$ is referred to as the *density distribution* of an anatomical structure.

We follow the proposal of Yao [Yao02] and model anatomical structures as tetrahedral grids with Bernstein polynomials describing the density distribution within each tetrahedron. The Bernstein polynomial of degree d is parameterized per tetrahedron t using Bernstein coefficients $c_t = \{c_{i,j,k,l}\}$ with $i + j + k + l = d$. Given a point $b = (b_x, b_y, b_z, b_w)^T$ in local barycentric coordinate space of t, the corresponding linear attenuation coefficient calculates as

$$\alpha_t(b) = \sum_{i+j+k+l=d} [c_{i,j,k,l} B^d_{i,j,k,l}(b)] \tag{2}$$

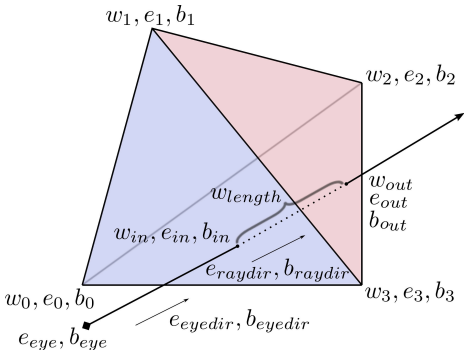


Figure 2: *Ray interaction with a single tetrahedron. The ray originates at the source e_{eye}, b_{eye}, intersects the tetrahedron at w_{in}, e_{in}, b_{in}, traverses the tetrahedron by length w_{length} and exits at the point $w_{out}, e_{out}, b_{out}$. The letter w stands for world coordinates, e for normalized eye coordinates, and b for barycentric coordinates.*

where

$$B^d_{i,j,k,l}(b) = \frac{d!}{i!j!k!l!}(b_x)^i(b_y)^j(b_z)^k(b_w)^l \qquad (3)$$

is the Bernstein basis function of degree d. The number of Bernstein coefficients assigned per tetrahedron therefore depends on the degree of the polynomial that is used to describe the density distribution: $|c_t| = \binom{d+3}{3}$, with $n = 1, 4, 10, 20$ for degrees $d = 0, 1, 2, 3$ respectively. By applying higher polynomial degrees, non-linear density distributions can be expressed in a tetrahedron (cf. Figure 1).

To accumulate the attenuation encountered by a ray $p(x)$ passing through a tetrahedron (cf. Figure 2), the Bernstein polynomial density distribution is integrated along the ray:

$$\int_{p(x)} \alpha_t(b)\mathrm{d}b = |w_{out} - w_{in}| \cdot \sum_{i+j+k+l=d} [c_{i,j,k,l} \int_{b_{in}}^{b_{out}} B^d_{i,j,k,l}(b)\mathrm{d}b] \qquad (4)$$

See [Yao02] for a detailed closed form solution of Equation 4.

3.2. Statistical shape and intensity models (SSIMs)

A SSIM, as employed in this work, is a statistical model of shape and intensity (or density) variation generated from a set of training tetrahedral grids by means of a principal component analysis (PCA). The SSIM represents the mean shape and density plus their specific variations described by the PCA eigenvectors. We stick with the notation of Yao [Yao02], who expressed an SSIM as $Y = \bar{Y} + Pr$. Here, $Y = [Y_s, Y_\mu]$ denotes an SSIM instance with tetrahedral vertex positions Y_s and coefficients Y_μ to Bernstein polynomial density distributions of a fixed degree. A model deformation is described as a linear combination of deformation parameters r and the eigenvector matrix P, added to the average vertex positions and Bernstein coefficients $\bar{Y} = [\bar{Y}_s, \bar{Y}_\mu]$.

We chose this type of deformable volumetric model, because the deformation can be controlled by only a few parameters. Additionally, tetrahedra are the simplest polyhedral cell type and allow a straight-forward interpolation in their interior.

4. A GPU-based algorithm for fast projection of unstructured tetrahedral grids

According to Equations 1 and 4, the total attenuation encountered by an X-ray in a single tetrahedron can be calculated directly, if the ray traversal distance in world coordinates $w_{length} = ||w_{out} - w_{in}||$ and its entrance and exit points in barycentric coordinates b_{in} and b_{out} are known. Moreover, the Beer-Lambert law states that the total attenuation calculates by the accumulated contributions of all tetrahedra along the ray, irrespective of the order they are traversed ("visibility order"). Our algorithm for simulating X-ray attenuation in tetrahedral grids exploits these properties by independently processing each tetrahedron of the grid.

We stream the tetrahedra and corresponding Bernstein polynomial coefficients through the vertex, geometry and fragment stages of the graphics pipeline. The entrance parameters of the rays on the front-facets of the tetrahedra, such as b_{in}, are interpolated linearly between tetrahedral vertices. Our approach adapts the idea of cell-based ray casting [WKME03, GW06] to determine the ray exit parameters b_{out} and their traversal depths w_{length} in the fragment shader stage using direct ray-facet intersection tests in barycentric coordinates. Both the entrance and exit parameters are then applied to integrate the Bernstein density functions (Equation 4) in closed form using per-fragment operations. As proposed in [SC06], we combine the contributions of single tetrahedra by summing up their contributions in a post-processing step.

4.1. Calculating the rays' entrance parameters

In a first processing step, *per-vertex* operations transform the tetrahedral vertices into normalized device coordinates and eye coordinates (e_i). The vertex positions are then handed to the per-geometry processing stage together with the Bernstein coefficients of the respective tetrahedron.

The *per-geometry* operations construct the matrix $M = [e_0, e_1, e_2, e_3]$ and its inverse M^{-1} using the normalized eye coordinates computed in the per-vertex stage. M^{-1} is a linear transformation matrix that projects eye coordinates into the local barycentric coordinate space of the respective tetrahedron. We apply M^{-1} to compute b_{eydir}, the eye ray

direction vector in barycentric coordinates at the tetrahedral vertices: $b_{eyedir} = b_i - \frac{M^{-1} \cdot e_{eye}}{|M^{-1} \cdot e_{eye}|}$. The vector b_{eyedir} points from the source to the entrance point in barycentric coordinates of a ray in the tetrahedron. An additional processing step then determines the length of its eye space correspondent, $e_{eyedir} = e_i - e_{eye}$, which is equal the length of the eye coordinates e_i at the tetrahedral vertices. Note that the source (the eye position) is located at $(0,0,0,1)^T$ in eye space.

Consecutive operations in the geometry shader stage triangulate a tetrahedron into its four facets. The parameters b_i, b_{eyedir} and $||e_{eyedir}||$ are assigned as parameters of the triangle facet's vertices. They are linearly interpolated between the front-faces of the tetrahedron, thus making the perspective-correct entrance parameters of all rays intersecting the tetrahedron available in the fragment shader stage. We utilize the culling functionality of graphics hardware to discard the rasterization of tetrahedral back-facets. The Bernstein coefficients are pushed down the GPU pipeline as non-varying parameters.

4.2. Calculating the rays' exit parameters

Given the front-face parameters b_{in}, b_{eyedir} and $||e_{eyedir}||$ of a ray, the goal is to find the correct exit parameter b_{out} and the traversal length w_{length}. Here, we propose a method which utilizes intersection tests in barycentric coordinates. In the following $b_{i,j}$ denotes the jth component of a barycentric coordinate b_i. Keeping in mind that local barycentric coordinates of a tetrahedron are non-negative and sum up to 1, we observe the following:

1. At least one component of b_{in} and of b_{out} is 0, since b_{in} and b_{out} are located on the facet of a tetrahedron: $\exists i.b_{in,i} = 0, \exists j.b_{out,j} = 0$.
2. In the "regular" case, a ray enters and exits a tetrahedron on two distinct facets. This implies, that the zero component of b_{in} has a different index than the zero component of the corresponding b_{out}: $\exists i.(b_{out,i} = 0 \wedge b_{in,i} > 0)$.

As an exception from (2.), the ray might only hit a tetrahedral vertex. In this case the traversal depth w_{length} is 0 and the ray is not attenuated.

We model the ray between the barycentric entrance and exit points of a tetrahedron as $b_{raydir} = b_{out} - b_{in}$. Since the vectors b_{raydir} and b_{eyedir} lie on the same ray of sight, b_{raydir} can be calculated by the linear transformation $b_{raydir} = b_{eyedir} \cdot s$ with the positive factor s. The barycentric exit coordinates of an ray then compute as

$$b_{out} = b_{in} + (b_{eyedir} \cdot s) \tag{5}$$

According to observation (1.), we know that one component of b_{out} is 0. Consequently, there exist four possible candidates for s, given that the parameters b_{in} and b_{eyedir} are known:

$$s_i = -\frac{b_{in,i}}{b_{eyedir,i}} \tag{6}$$

The s_i describe the solutions for intersecting the ray with all four tetrahedral facets.

Our algorithm for finding the correct s_i ignores all results with $b_{eyedir,i} \geq 0$ or $b_{in,i} = 0$. Among the remaining candidates, the smallest positive s_i is the correct solution for s. If no candidate is found, then $s = 0$ and therefore $b_{in} = b_{out}$.

Once the transformation factor s is determined, we use it to compute the actual exit point of the ray on the tetrahedron b_{out} according to Equation 5. Given b_{out} it would now be possible to determine e_{out} by applying the transformation matrix M and determine $w_{length} = ||e_{out} - e_{in}||$. However, we propose a more efficient method based on the fact that s also scales the ray direction vector in normalized eye coordinates to the traversal length of the ray:

$$\begin{aligned} w_{length} &= ||e_{out} - e_{in}|| \\ &= ||M \cdot (b_{out} - b_{in})|| \\ &= ||M \cdot (b_{raydir})|| \\ &= ||M \cdot s \cdot (b_{eyedir})|| \\ &= s \cdot ||e_{eyedir}|| \end{aligned} \tag{7}$$

In our algorithm, we apply s to calculate b_{out} according to Equation 5 and w_{length} according to Equation 7. Afterwards, all parameters are available to solve the Bernstein rendering integral in the fragment shader stage. The attenuation encountered by the ray is then returned as the fragment color and can be summed up (e.g. blended) with the contributions of other tetrahedra in the grid.

5. An OpenGL implementation

This section describes an OpenGL-based implementation of our algorithm. We will first propose an implementation of the algorithm to render static tetrahedral grids with higher-order attenuation functions that was presented in the previous section. This approach is then extended by a GPU-implementation for the deformation of SSIMs. We implemented the rendering pipeline based on OpenGL version 4.0, featuring the OpenGL Shading Language (GLSL) as of version 4.00.

5.1. Projection of tetrahedral grids on the GPU

In the following we will describe the render process for a single tetrahedron. To generate the final image, i.e. ac-

cumulating the attenuation of all tetrahedra in a 2D texture, we simply set the OpenGL blending functionality to `glBlendFunc(GL_ONE, GL_ONE)`, `glBlendEquation(GL_FUNC_ADD)` and bind a framebuffer object (FBO). Taking the exponential of the summed-up contributions in the 2D textures is then performed in the post-processing step using an addional rendering pass.

One `GL_POINT` primitive is issued per tetrahedron. The respective tetrahedral vertex positions in world coordinates and the Bernstein coefficients are thereby assigned as point vertex attributes. The vertex shader is executed once for every tetrahedron (`GL_POINT` primitive), and therefore has access to the four vertex coordinates of the respective tetrahedron. It performs the transformation into (normalized) eye and device coordinate space. The transformed vertices and the Bernstein coefficients are then streamed further down the rendering pipeline.

In the geometry shader stage, M^{-1} is computed using the GLSL `inverse()` call on $M = [e_0, e_1, e_1, e_3]$. The matrix M^{-1} is stored as a `dmat4` data type of 64bit precision. (Note: If `dmat4` is not available, `mat4` can be used alternatively. This might lead to visual artefacts due to numerical issues in the matrix inversion.) In the next step the tetrahedron is decomposed into its four triangle facets and the $||e_i||$, b_i and b_{eyedir} are linearly interpolated in between the geometry and fragment shader stages. OpenGL backface culling is enabled explicitly, such that only the front-faces of the tetrahedron are rasterized.

The fragment program performs the ray-tetrahedron intersection tests in one vector division and the correct solution s is found by issuing four `mix()` tests on the s_i. It then evaluates the Bernstein integral (Equation 4). In order to efficiently determine the integral solution, the Bernstein basis functions are precomputed using hard-coded multinomial factors as proposed by [SC06].

Due to numerical precision issues, the b_{eyedir} components sometimes contain values close to 0 at the edges or vertices of a tetrahedron. The implementation avoids artefacts, by discarding fragments which fulfill $||b_{raydir}|| > \sqrt{2}$.

5.2. Combined deformation and projection of SSIMs on the GPU

The whole SSIM is stored in the graphics hardware memory in order to perform the deformation and projection entirely on the GPU. Our implementation utilizes OpenGL texture buffer objects (TBOs) to store the mean vertex coordinates $\bar{Y}_s$ and the mean Bernstein coefficients $\bar{Y}_\mu$. Their components can be accessed using unique vertex identifiers and unique tetrahedron identifiers respectively. The eigenvector components of the vertex positions and of the Bernstein coefficients are each split in one texture array (`GL_TEXTURE_2D_ARRAY`). Every eigenvector maps to exactly one 2D texture where the respective eigenvector components are stored consecutively. Additionally, two vertex arrays hold the static tetrahedral indices and the corresponding vertex indices. The width and height of the 2D eigenvector textures and the number of eigenvectors are handed to the shader stages as uniform parameters. The deformation parameters are pushed to the pipeline using a TBO as well.

In the vertex shader stage, the mean values are extracted from the `samplerBuffer` (TBO) using the current tetrahedral and vertex ids. The normalized texture coordinates for accessing the eigenvector textures are then precomputed, applying the uniform texture width and height. Afterwards, the vertex shader performs the deformation of both vertex positions and Bernstein coefficients. The "deformed" tetrahedron is then projected in consecutive shader stages according to the projection method introduced in the preceding section.

To avoid multiple deformation of vertices that are shared between tetrahedra a preliminary rendering pass is issued that only performs the geometric deformation. Here, the OpenGL transform feedback buffer is utilized to avoid data exchange between CPU and GPU. One primitive is rendered per vertex, and the deformation of the vertex positions is computed in the vertex shader stage. The implementation discards the fragment rasterization using the OpenGL `glEnable(GL_RASTERIZER_DISCARD_NV)` functionality. Rather than putting the result on the screen, the deformed vertices are stored directly on the GPU. They are then bound to the shader programs as vertex buffer objects in a consecutive rendering pass. A different vertex shader program deforms the attenuation coefficients and proceeds with the projection accordingly. The model is therefore deformed and projected entirely on the GPU in three rendering passes.

6. Experiments and Results

Both the implemented projection algorithm and the combined deformation and projection methods were evaluated independently in terms of their speed. We additionally compared the rendering quality of the projection approaches to ground truth rendering generated from CT data. For the evaluation, a standard desktop PC with a 3GHz Intel Core2Duo CPU E8400 and 12 GB DDR2 main memory was used. The PCI Express x16 Gen2 slot of the machine was equipped with an NVIDIA GeForce GTX 560 Ti GPU (NVIDIA 295.20 graphics drivers).

6.1. Rendering speed

To evaluate the rendering speed of the projection method only, we used tetrahedral grids based on the Stanford "dragon" dataset with varying grid resolutions (from 17k tetrahedra to 4M tetrahedra) and the four density function degrees. The data was generated by sampling the dragon as Bernstein polynomial density distributions onto equally sized spherical

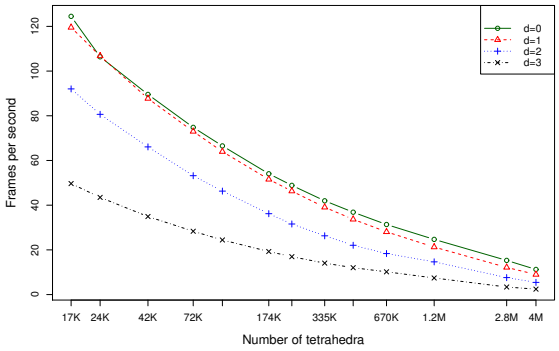


Figure 3: *Rendering speed performance in average frames per second (30 measurements) on the dragon dataset for all four density distribution degrees.*

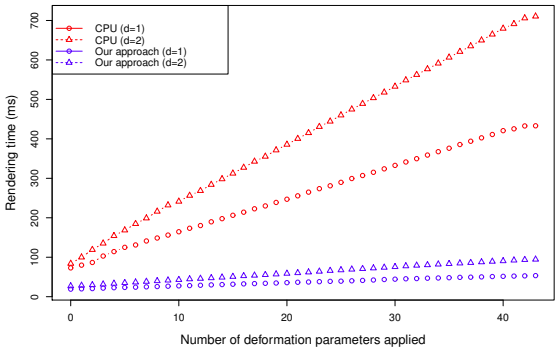


Figure 4: *Rendering time (in milliseconds, averaged over 30 experiments) of the combined deformation and projection of the femur SSIM for a density function degree $d = 1$ and $d = 2$.*

tetrahedral grids. We then recorded the projection speed while rotating the camera around the data in 12 degree increments. Therefore, in total 30 projections were generated per grid resolution and degree for a full 360° trajectory. Using an artificial dataset we could ensure that the viewport of size 1000^2 showed a constant pixel coverage of 75%.

The results depicted in Figure 3 show that the projection speed approximately decreases inversely linear to the number of tetrahedra rendered. Higher polynomial degrees reduce the projection speed compared to rendering density distributions with $d = 0$ or $d = 1$.

To evaluate the combined deformation and projection speed of SSIMs, we employed a statistical model based on an SSIM of the femur (thighbone) developed by Bryan et al. [BSMH*10] featuring a resolution of 616k tetrahedra and 45 eigenvectors. For the original SSIM only one density value was given for each tetrahedron. To compare different degrees we filled Bernstein coefficients for degrees 1 to 3 with dummy data. Although this results in inaccurate density distributions for higher polynomial degrees, the deformation and projection speed is not affected by this conversion.

The combined deformation and projection time of our implementations was recorded while projecting the SSIM onto an 1000^2 viewport in anterior view. To deform the model, 0 to 44 eigenvectors were considered and 30 combined deformations and projections issued for every number of deformation parameters. We chose the parameters randomly equally distributed within the range of the minimum and maximum weight of the respective training data. To provide a rough comparison to a CPU approach, a single-threaded deformation on the CPU was measured. Note that this did not include the time for projecting the deformed model.

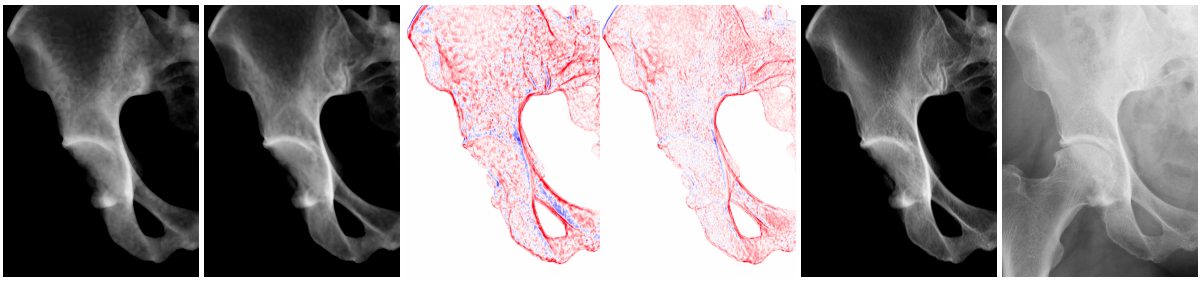
The results of this evaluation are given in Figure 4. The rendering time of the combined deformation and projection grows nearly linearly with the number of regarded eigenvectors (e.g. the number of applied deformation parameters). Our evaluation showed that the GPU-based deformation and projection of models with density degree 3 (not depicted) reaches a rendering time of 500 and 2000 milliseconds when applying 10 and 40 deformation parameters respectively. Density distributions with $d = 0$ were processed as fast as $d = 1$ distributions.

6.2. Rendering quality

To judge on the rendering quality, we compared our projection approach to ground truth images generated from clinical CT data of a pelvis (resolution $512 \times 512 \times 531$). We segmented the pelvis and extracted tetrahedral grids of four resolutions (20k, 58k, 252k and 731k tetrahedra) with density functions sampled from the original CT data. A GPU implementation of the ray casting algorithm proposed by [KW03] that is respecting the Beer-Lambert law was used to generate the ground-truth projections.

During experiments, the viewport was set to a resolution of 1500^2 with the projected pelvis covering 31% of the viewport pixels in anterior view. We measured the quality of the projections generated with our results in terms of RMS distance and mean absolute error to the ground truth. Only those pixels were regarded in the distance measures that were either covered by the projected pelvis in the ground truth or in the virtual X-ray image compared to it.

Table 1 summarizes the results and Figure 5 provides example projections and difference images. Note how the image quality increases by utilizing higher density distribution degrees while keeping the grid resolution fixed.



(a) (b) (c) (d) (e) (f)

Figure 5: *Comparison of X-rays to ground truth data from CT. SSIM instance featuring (a) 56k tetrahedra and degree $d = 0$, (b) $d = 2$. The corresponding difference images [(c)(d)] to ground truth (e) with red indicating positive error, blue negative error. An clinical X-ray image showing the same pelvis with similar pose is depicted in (f)*

	d=0	d=1	d=2	d=3
20k rms	0.066	0.056	0.054	0.051
abs	0.046	0.039	0.039	0.035
58k rms	0.046	0.038	0.034	0.030
abs	0.034	0.028	0.025	0.022
252k rms	0.035	0.029	0.023	0.021
abs	0.026	0.022	0.016	0.014
731k rms	0.029	0.023	0.019	0.018
abs	0.021	0.017	0.012	0.011

Table 1: *Root mean square and mean absolute error between X-rays from model instances and ground truth projections from the segmented pelvis CT (depicted in Figure 5). The maximal pixel intensity in the ground truth image is 0.974.*

7. Discussion

In general, it can be observed that the projection speed approximately decreases inversely linear to the number of tetrahedra rendered. One would expect this behavior, assuming that the computations performed per tetrahedron remains constant when the grid resolution is increased.

The simulation of X-ray attenuation in tetrahedral grids with Bernstein density functions of degree 1 is only slightly slower than using a degree of 0. This holds true for the combined deformation and projection on the GPU as well. We attribute this to the hardware-accelerated vector operations on the graphics hardware, which allows an efficient processing of the four Bernstein coefficients stored in a four component vector.

With higher-degree density distributions, the performance is decreasing. This matches well with the investigation of Sadowsky et al. [SC06]. They note that the decrease is caused by the Bernstein density function terms integrated in the fragment shader, which show a “trend to exponential growth”. However, our implementation still reaches interactive rates even for $d = 2$ and more than 1 million tetrahedra.

The rendering time of the combined deformation and projection grows nearly linearly with the number of regarded eigenvectors (e.g. the number of applied deformation parameters). For a polynomial degree of up to 2, the combined deformation and projection outperforms our CPU implementation by a factor of three to seven and scales better with respect to the number of eigenvectors. It is important to note that the CPU measurement does only include the time for deforming the model. In practice, additional resources are required to push the deformed model to the GPU and to project it accordingly.

In our evaluation, the combined deformation and projection of models with density degree 3 on the GPU is significantly slower than with lower degrees. We identified the texture fetch operations to extract the eigenvector components as a bottleneck. When projecting using a degree of 1, including the maximum number of deformation parameters, we achieve interactive rendering times of about 50ms. Since the major variation of the SSIM is captured by the first eigenvectors, only a subset of the deformation parameters have to be considered for the application in a 3D reconstruction framework. In this case a higher degree might be used that significantly increases projection quality without loss of performance compared to all parameters with a lower degree.

The quality evaluation indicates that our method generates virtual X-ray images similar to our ground truth. When higher-order density distributions are projected, anatomical features such as cortical structures are depicted distinctly even on lower grid resolutions. For density functions of degree 0, similar results can only be obtained by generating the X-ray images from grids of much higher resolution.

The difference images to ground truth data show discrepancies at the boundaries of the projected pelvic bone (cf. Figures 5(c) and (d)). We attribute this to the process of grid generation. The strong geometric simplification for the lower resolution tetrahedral grids in areas of high curvature leads to boundary triangles located “inside” the pelvis. Therefore, the boundary tetrahedra fail at recovering the thin,

high-density cortical shell of the bone during the assignment of density values. This has to be considered for model generation in future studies.

Compared to [SC06], our approach is a priori perspective correct. We tailored the algorithm for the efficient integration of Bernstein polynomial functions in barycentric space. It does not require a pre-computation of face normals or an explicit conversion from barycentric to eye space in order to compute all integration parameters.

8. Conclusion

In this work, we proposed an algorithm for fast and accurate simulation of X-ray images from deformable volumetric models, represented by tetrahedral grids. The method is specifically designed for GPU-accelerated execution and utilizes the specific properties of X-ray attenuation to reach a high degree of parallelization. Our projection method in combination with the model deformation process achieves interactive frame rates even for large tetrahedral grids.

By employing higher-order attenuation functions, it is possible to keep the resolution of the tetrahedral grid low while obtaining a quality that is comparable to significantly higher resolution grids with constant intensity encoding. Our method also supports the use of different polynomial degrees on the same grid, requiring only one additional rendering pass for each new degree. This allows tetrahedral models to be tailored for specific anatomical structures and applications, both in terms of grid resolution and density distribution.

We presented a shader program implementation of the projection algorithm that avoids any explicit branching or looping. It processes the tetrahedra independent from each other and therefore reaches a high degree of parallelism, while making full use of the hardware-accelerated vector operations. Due to the feed-forwardnature of the proposed pipeline, it might furthermore be coupled with arbitrary deformation techniques executed on the CPU or the GPU.

The proposed extension to embed the deformation of vertex positions and higher-order density distributions on the GPU is shifting computational resources from the CPU to the GPU. This has several advantages compared to a CPU-based deformation: First, the CPU can concurrently perform other computations, e.g. related to the reconstruction process. Second, for a density function's degree up to two, the combined deformation and projection on the GPU shows a performance increase by a factor of three to seven compared to the CPU-based deformation. Finally, the GPU-based deformation scales better with respect to the number of deformation parameters considered. Further research is required to find a more efficient GPU-memory storage and access pattern for very large SSIMs with a high polynomial degree.

Future studies should include an in-depth evaluation of the application of our projection method to the problem of reconstructing a 3D anatomical model from clinical X-ray data. We expect that in combination with existing methods for GPU-based image-registration [FVW*11], our projection method increases both, speed and accuracy of the 3D reconstruction process.

Acknowledgements

This work was partially supported by the EU-FP7 Project MXL (ICT-2009.5.2) and the DFG Research Center Matheon.

References

[BKdB*11] BAKA N., KAPTEIN B. L., DE BRUIJNE M., VAN WALSUM T., GIPHART J. E., NIESSEN W. J., LELIEVELDT B. P. F.: 2D-3D shape reconstruction of the distal femur from stereo X-ray imaging using statistical shape models. *Medical image analysis 15*, 6 (Dec. 2011), 840–50.

[BSMH*10] BRYAN R., SURYA MOHAN P., HOPKINS A., GALLOWAY F., TAYLOR M., NAIR P.: Statistical modelling of the whole human femur incorporating geometric and material properties. *Medical engineering & physics 32*, 1 (2010), 57–65.

[DLvB*10] DWORZAK J., LAMECKER H., VON BERG J., KLINDER T., LORENZ C., KAINMÜLLER D., SEIM H., HEGE H.-C., ZACHOW S.: 3D reconstruction of the human rib cage from 2D projection images using a statistical shape model. *International journal of computer assisted radiology and surgery 5*, 2 (Mar. 2010), 111–24.

[FVW*11] FLUCK O., VETTER C., WEIN W., KAMEN A., PREIM B., WESTERMANN R.: A survey of medical image registration on graphics hardware. *Computer methods and programs in biomedicine 104*, 3 (Dec. 2011), e45–57.

[GW06] GEORGII J., WESTERMANN R.: A generic and scalable pipeline for GPU tetrahedral grid rendering. *IEEE transactions on visualization and computer graphics 12*, 5 (2006), 1345–52.

[KW03] KRUGER J., WESTERMANN R.: Acceleration techniques for gpu-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003* (2003), IEEE.

[LWH06] LAMECKER H., WENCKEBACH T. H., HEGE H.-C.: Atlas-based 3D-shape reconstruction from x-ray images. In *Proc. Int. Conf. of Pattern Recognition (ICPR2006)* (2006), vol. Volume I, IEEE Computer Society, pp. 371–374.

[MMF10] MAXIMO A., MARROQUIM R., FARIAS R.: Hardware-Assisted Projected Tetrahedra. *Computer Graphics Forum 29*, 3 (Aug. 2010), 903–912.

[SC06] SADOWSKY O., COHEN J.: Projected tetrahedra revisited: A barycentric formulation applied to digital radiograph reconstruction using higher-order attenuation functions. *IEEE Transactions on visualization and computer graphics 12*, 4 (2006), 461–73.

[SFK08] STEININGER P., FRITSCHER K., KOFLER G.: Comparison of Different Metrics for Appearance-model-based 2D/3D-registration with X-ray Images. In *Bildverarbeitung für die Medizin (BVM)* (Berlin, Heidelberg, 2008), Informatik aktuell, Springer Berlin Heidelberg, pp. 122–126.

[SQNS12] SERRURIER A., QUIJANO S., NIZARD R., SKALLI W.: Robust femur condyle disambiguation on biplanar X-rays. *Medical engineering & physics* (Feb. 2012).

[VGF*10] VIDAL F. P., GARNIER M., FREUD N., LETANG J. M., JOHN N. W.: Simulation of X-ray Attenuation on the GPU. *Theory and Practice of Computer Graphics* (2010), 25–32.

[WKME03] WEILER M., KRAUS M., MERZ M., ERTL T.: Hardware-based view-independent cell projection. *IEEE Transactions on Visualization and Computer Graphics 9*, 2 (2003), 163–175.

[Yao02] YAO J.: *A statistical bone density atlas and deformable medical image registration*. PhD thesis, The Johns Hopkins University, 2002.

[Zhe10] ZHENG G.: Statistical shape model-based reconstruction of a scaled, patient-specific surface model of the pelvis from a single standard AP x-ray radiograph. *Medical physics 37*, 4 (Apr. 2010), 1424–39.