Interactive Isosurfaces with Quadratic C^1 Splines on Truncated Octahedral Partitions

Alexander Marinc[1], Thomas Kalbe[2], Markus Rhein[3] and Michael Goesele[2]

[1]Fraunhofer IGD Darmstadt [2]GRIS TU Darmstadt [3]Universität Mannheim

ABSTRACT

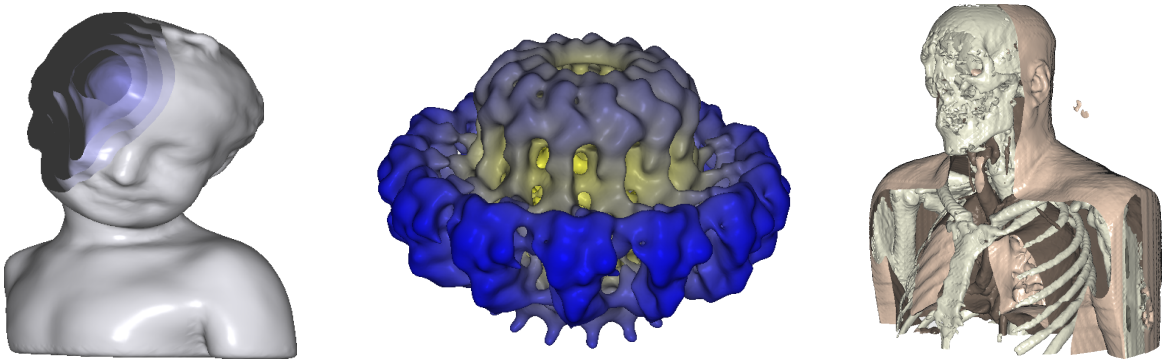
The reconstruction of a continuous function from discrete data is a basic task in many applications such as the visualization of 3D volumetric data sets. We use a local approximation method for quadratic C^1 splines on uniform tetrahedral partitions to achieve a globally smooth function. The spline is based on a truncated octahedral partition of the volumetric domain, where each truncated octahedron is further split into a fixed number of disjunct tetrahedra. The Bernstein-Bézier coefficients of the piecewise polynomials are directly determined by appropriate combinations of the data values in a local neighborhood. As previously shown, the splines provide an approximation order two for smooth functions as well as their derivatives. We present the first visualizations using these splines and show that they are well-suited for GPU-based, interactive high-quality visualization of isosurfaces from discrete data.

Keywords: piecewise quadratic polynomials, volume data, GPU ray casting, isosurfaces

1. INTRODUCTION

One of the main challenges for the visualization of isosurfaces from discrete data sets is to find a continuous function which is defined on the whole domain and which approximates or interpolates the discrete values given at the grid points. Several techniques to create appropriate visualizations have been developed, with different advantages and disadvantages with respect to the quality of the reconstructed surfaces, visualization performance, or memory requirements. The most popular surface reconstruction method from volume data is the Marching Cubes algorithm.[1] Although GPU implementations of Marching Cubes exist,[2] memory requirements for the triangle meshes are high which makes it difficult to reconstruct isosurfaces from large volume data sets. Furthermore, the resulting meshes and accordingly the surface approximations suffer from several aliasing artifacts, such as stair-casing.

Another well-known method is the usage of tensor product splines. With tensor product splines of degree two or three, the construction of smooth surfaces is possible but requires the computation of polynomial pieces with total degree six or nine, respectively. For high-quality shading with interactive frame rates the desired polynomial degree should, however, be as low as possible, while still yielding an overall smooth surface model.



(a) (b) (c)

Figure 1: Renderings performed with our new quadratic C^1 spline. (a) Isosurfaces reconstructed from a signed distance function on a 256^3 grid (about 50 000 up to 300 000 visible tetrahedra). (b) Isosurface of biological data (100^3 voxels, 1.2 mio. visible tetrahedra). (c) Isosurfaces of medical data (128^3 voxels, 800 000, and 2.2 mio. visible tetrahedra).

Visualization and Data Analysis 2011, edited by Pak Chung Wong, Jinah Park, Ming C. Hao, Chaomei Chen, Katy Börner, David L. Kao, Jonathan C. Roberts, Proc. of SPIE-IS&T Electronic Imaging, SPIE Vol. 7868, 786808 · © 2011 SPIE-IS&T · CCC code: 0277-786X/11/$18 · doi: 10.1117/12.876105

Alternatively, trivariate splines, i.e., splines defined on tetrahedral partitions of the domain, have been proposed for isosurface reconstruction and visualization. Roessl et al.[3] used quadratic polynomials of total degree two in Bernstein-Bézier form (B-form) to derive a continuous function approximating the volume data. The coefficients of the piecewise polynomials are hereby directly available from the volume data by appropriate averages of local data values. The low degree of the polynomial pieces allows for an efficient ray-surface intersection for high-quality visualization by ray casting. One of the drawbacks of this approach is that the spline is C^1 smooth only in certain points. Cubic C^1 splines on tetrahedral partitions have been proposed by Sorokina and Zeilfelder,[4] and a GPU visualization algorithm for isosurface ray casting based on cubic C^1 and quadratic super splines has been given.[5,6] Both splines are defined on the so-called *type-6* tetrahedral partition where each data cube is split into 24 congruent tetrahedra. Another closely related approach is given by Kloetzli et al.,[7] where cubic C^0 splines are constructed on arbitrary tetrahedral partitions by a Moving Least Squares approximation of the volume data. Note that in this method, the coefficients cannot be computed directly from the volume data by simple averaging. Thus, memory demands for storing the precomputed polynomials is high, rendering a real-time approach for reconstruction and visualization difficult.

We use ray casting on the GPU based on a new trivariate spline which for the first time solves the problem of finding a local approximation method by quadratic polynomials and is globally C^1 continuous.[8] This spline is based on a more complex tetrahedral partition using a truncated octahedral partition of the volumetric domain, see Section 3, and provides a better numerical approximation of the original data compared to the cubic splines on type-6 tetrahedral partitions.

We give a short overview of trivariate splines in B-form and associated tetrahedral partitions in the next section. Then, we introduce the new spline in Section 3, i.e., describe the underlying tetrahedral partition and give formulas defining the coefficients of the polynomial pieces. In Section 4 we briefly describe the visualization algorithm which is adapted from our earlier work[5,6] to the new partition. Finally, we compare the visual quality obtained from our method with standard trilinear interpolation as well as cubic C^1 splines on type-6 tetrahedral partitions, and analyze its performance.

2. TRIVARIATE BERNSTEIN-BÉZIER-SPLINES

We use smooth trivariate splines of degree two in piecewise Bernstein-Bézier form to obtain a continuous approximation of the discrete data. The B-form bears several advantages, including stable evaluation of the spline and its derivatives. For smoothness between neighboring polynomials only local rules based on geometrical conditions have to be considered.

For a non-degenerated tetrahedron T with vertices $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$, each polynomial is given by

$$s_{|T} \equiv \sum_{i+j+k+l=2} b_{ijkl} B^2_{ijkl}, \tag{1}$$

where $b_{ijkl} \in \mathbb{R}$ are the ten Bernstein-Bézier coefficients, and B^2_{ijkl} are the Bernstein polynomials of degree two, defined by

$$B^2_{ijkl} \equiv \frac{2!}{i!\,j!\,k!\,l!} \phi_0^i \phi_1^j \phi_2^k \phi_3^l. \tag{2}$$

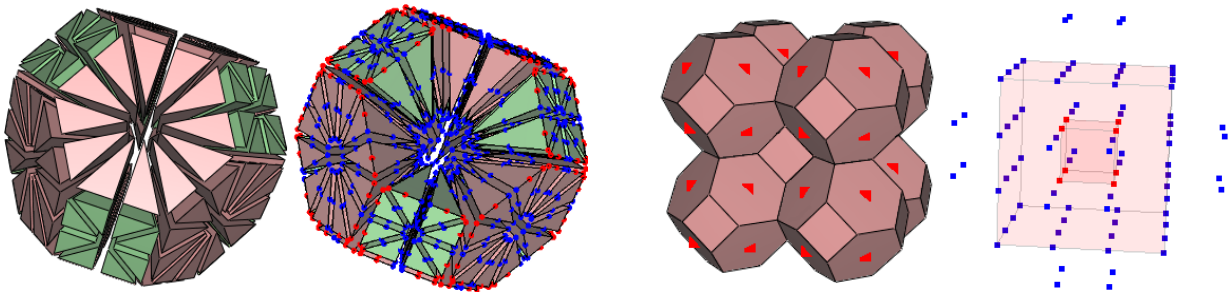
Here, ϕ_μ with $\mu = 0,1,2,3$, are the barycentric coordinate functions with respect to T. These are linear polynomials uniquely defined by the property

$$\phi_\mu(\mathbf{v}_\nu) = \delta_{\mu,\nu}, \quad \mu,\nu = 0,1,2,3,$$

where $\delta_{\mu,\nu}$ is Kronecker's symbol. The relation between a point $\mathbf{x}$ in $\mathbb{R}^3$ and associated barycentric coordinates is thus given by $\begin{pmatrix}\mathbf{x} & 1\end{pmatrix}^\mathsf{T} = A_T \cdot \begin{pmatrix}\phi_0 & \phi_1 & \phi_2 & \phi_3\end{pmatrix}^\mathsf{T}$, where $A_T \in \mathbb{R}^{4\times 4}$ is defined as

$$A_T = \begin{pmatrix} \mathbf{v}_0 & \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 \\ 1 & 1 & 1 & 1 \end{pmatrix}. \tag{3}$$

Each polynomial is uniquely determined by the ten coefficients b_{ijkl}, $i+j+k+l=2$, which are associated with the domain points $1/2(i\,\mathbf{v}_0 + j\,\mathbf{v}_1 + k\,\mathbf{v}_2 + l\,\mathbf{v}_3)$ on T. Values and derivatives of the polynomials can be efficiently evaluated with the algorithm of de Casteljau. Using *blossoming*,[9] a generalization of the de Casteljau algorithm where the arguments may vary on each level, we can reduce the trivariate polynomial to a univariate quadratic polynomial along an arbitrary ray intersecting the tetrahedron T. Now it is easy to find the root of this univariate polynomial by solving simple quadratic equations, which is a strong benefit for the later visualization, see Section 4.2.



(a) (b)

Figure 2: (a) *Left*: Each TO is split into 144 tetrahedra. *Right*: The domain points on each tetrahedron are shown in red and blue. The front-most tetrahedra are removed for clarity of the illustration. (b) *Left*: the arrangement of the TOs within the volume. *Right*: the 88 data values determining the coefficients of one single TO.

2.1 Smoothness Between Neighboring Polynomials

Another important advantage of the B-form is the simple description of the smoothness conditions between neighboring polynomials. Consider two neighboring, non-degenerated tetrahedra $T = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$ and $\widetilde{T} = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \widetilde{\mathbf{v}}_3]$, that share the common triangle $T \cap \widetilde{T} = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2]$. Let b_{ijkl} and $\widetilde{b}_{ijkl}$, $i+j+k+l=2$, be the polynomial coefficients in B-form of T and $\widetilde{T}$. The polynomials are continuously connected over $[\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2]$ if $b_{ijk0} = \widetilde{b}_{ijk0}$ holds true for $i+j+k=2$. Furthermore, the patches are also C^1 continuous over $[\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2]$ if additionally the condition

$$b_{ijk1} = \widetilde{b}_{i+1,j,k,0}\phi_0(\widetilde{\mathbf{v}}_3) + \widetilde{b}_{i,j+1,k,0}\phi_1(\widetilde{\mathbf{v}}_3) + \widetilde{b}_{i,j,k+1,0}\phi_2(\widetilde{\mathbf{v}}_3) + \widetilde{b}_{i,j,k,1}\phi_3(\widetilde{\mathbf{v}}_3) \tag{4}$$

holds true for $i+j+k=1$. C^1 smoothness between two neighboring polynomials can therefore be described by three simple conditions. If these conditions are fulfilled for all neighboring polynomials in the volume, then the spline is globally C^1 smooth on the whole underlying domain. The challenge is to determine the coefficients from the data values while fulfilling all necessary conditions for a globally smooth spline and simultaneously allowing a good approximation of the data.

3. SPLINES ON TRUNCATED OCTAHEDRAL PARTITIONS

A complete description of the partition and the computation of the appropriate coefficients is given in Rhein and Kalbe. [8] Here we summarize the main ideas and the necessary basics needed to understand our visualization pipeline.

3.1 The Tetrahedral Partition

In the following we characterize a truncated octahedron (TO) and show how it is further subdivided into 144 tetrahedra. Each TO consists of six square and eight hexagonal faces that are connected at 24 vertices, see Fig. 2a. Assuming that the center of a TO with height h lies in the origin of a three-dimensional coordinate system, the vertices are all permutations of the triple $(0, \pm h/2, h)$. All tetrahedra of a TO share one point in the barycenter of the TO which we denote as $\mathbf{v}_0$ in every tetrahedron $T = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$. Furthermore, there is another vertex of each tetrahedron lying in the barycenter of a face of the TO, say $\mathbf{v}_1$, one vertex at the midpoint of an edge of the TO, say $\mathbf{v}_2$, and one vertex, say $\mathbf{v}_3$, coincides with a vertex of the TO itself. Therefore, each tetrahedron has three faces inside the TO and one face on the boundary of the TO. This results in two differently shaped tetrahedra, depending on the shape of the face of the TO the boundary face is lying on. In the following we denote all tetrahedra by T^S if the boundary face is within a squared face of a TO and T^H otherwise. The faces of a tetrahedron are denoted by F^S and F^H accordingly, whether they are a face of a tetrahedron T^S or not. Considering a TO partition of the volumetric domain and a uniform tetrahedral partition of the TOs as described above, the C^1 smoothness of a spline in B-form can be described completely by three conditions, see Section 2.1, for all interior triangular faces. Since the construction of the tetrahedral partition results in eight possible geometrically distinct connections between neighboring polynomials, the C^1 continuity is described by 24 formulas. We show a typical condition for a triangular face $F_2^H = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_3]$ in Fig. 3: Due to the symmetry of the subdivision scheme, all occurring weights are simple fractions or even zero, which results in a further simplification of the formula involving only four or three coefficients. The full set of equations can be found in Rhein and Kalbe. [8]

$$b_{1010} = \tfrac{1}{2}b_{1100} - \tilde{b}_{1010} + \tfrac{3}{2}b_{1001}$$
$$b_{0110} = \tfrac{1}{2}b_{0200} - \tilde{b}_{0110} + \tfrac{3}{2}b_{0101}$$
$$b_{0011} = \tfrac{1}{2}b_{0101} - \tilde{b}_{0011} + \tfrac{3}{2}b_{0002}$$

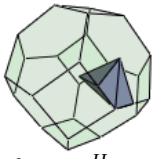


Figure 3: The C^1 conditions for the faces F_2^H.

3.2 The Truncated Octahedral Partition and the Data Mesh

Before we give an overview on how to compute the coefficients of a TO, it is necessary to understand how the TOs are connected in a space-filling partition and how they are positioned in a regular volumetric grid. In Fig. 2b, left, we give an example of a small 4^3 data grid including the embedded TO. As shown, each grid point lies at the barycenter of a hexagonal face. We directly associate these eight grid points with the appropriate TO. Each data point is hereby associated with two TOs. Therefore, to cover the whole domain we need one fourth as many TOs as there are data points in the volumetric grid. The spline approximation scheme[8] is described through a linear operator which maps the set of discrete data values into the space of quadratic C^1 splines regarding the uniform tetrahedral partition described in the previous section. This operator is given explicitly by concrete formulas for the computation of the spline coefficients in B-form as simple linear combinations of some local data values. On each TO, the spline is uniquely determined by 96 coefficients: the 24 coefficients associated with the vertices of the TO, together with the 60 coefficients associated with the domain points on the edges of the TO. The remaining coefficients follow from the C^1 conditions. For computing the 96 determining coefficients we need the 88 neighboring values as shown in Fig. 2b, right. Alternatively, we can compute the coefficients on each tetrahedron directly from the volume data, where we need 28 neighboring values.

4. GPU KERNELS FOR INTERACTIVE ISOSURFACE VISUALIZATION

In this section, we describe our pipeline for interactive isosurface visualization using quadratic C^1 splines on truncated octahedral partitions. We first describe our preprocessing step adapted from Kalbe et al.,[6] followed by an overview of the GPU ray casting approach for visualization.

4.1 GPU Preprocessing

Since the spline coefficients on each TO are given by averages of local portions of the data, we can process each TO independently. This is the basis for a massively parallel computation of the spline coefficients. As shown in the next subsection, tetrahedra are processed in the graphics pipeline for visualization. In order to improve rendering performance, we thus determine in a preprocess the tetrahedra contributing to the final surface. To do this, we can simply exploit the *convex hull property* of the B-form: if all b_{ijkl} of a tetrahedron T are either below or above the isolevel ρ_{iso}, we can

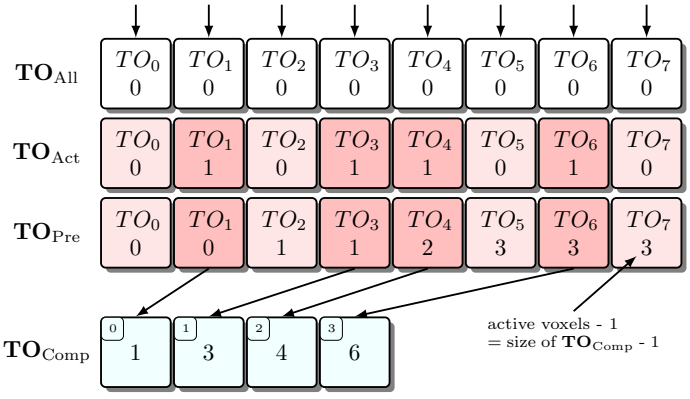
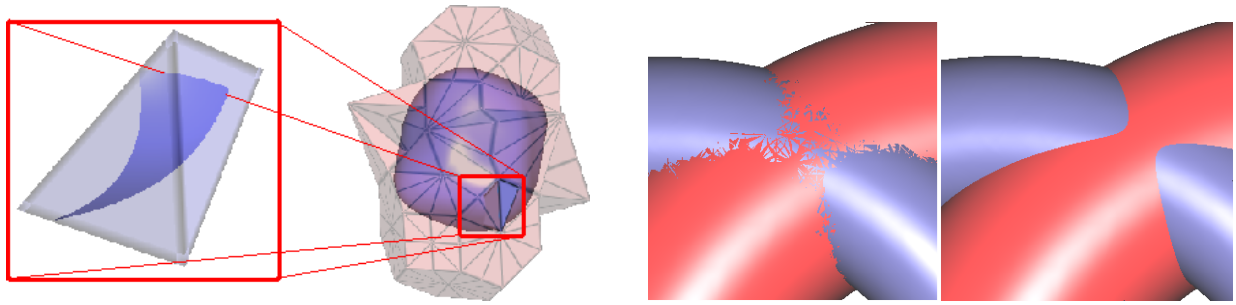


Figure 4: Illustration of parallel prefix sums used in our approach (see Sect. 4.1).



(a) (b)

Figure 5: (a) An enlarged surface patch within one tetrahedron of the smooth quadratic spline. (b) *Left*: Overlapping surfaces lead to errors when the fragments' z-values are not adjusted. *Right*: The same surfaces with adjusted z-values.

discard T from further processing. Since, in most cases, only a portion of the tetrahedra contribute to the surface, we get a significant speedup for visualization. We denote the set of contributing tetrahedra as the *active partition*. We further use 144 distinct lists to encode the active partition, since then we can use *instancing* for fast rendering of each tetrahedron type in the TO partition, see Section 4.2. To obtain the active partition for each type of tetrahedron, we use *parallel prefix sums*[10,11] from the CUDA data parallel primitives library (CUDPP), see Fig. 4 for an illustration. All tetrahedra are classified by a CUDA kernel as either *active* or *not active* by using the convex hull property of the B-form. The results are written into the arrays $\mathbf{TO}_{\text{All}}$. Next, we compute the exclusive prefix sums of $\mathbf{TO}_{\text{All}}$ and find the result in the array $\mathbf{TO}_{\text{Pre}}$. The array $\mathbf{TO}_{\text{Pre}}$ is finally compressed into an array $\mathbf{TO}_{\text{Comp}}$, which size equals the number of active tetrahedra and where each entry corresponds to a unique index defining the position of the tetrahedron in the volume. Note that the preprocess has to be done for each change of isolevel ρ_{iso}.

4.2 Visualization by GPU Ray Casting

Visualization of the active tetrahedra is performed in the OpenGL graphics pipeline in a hybrid cell projection / ray casting approach. We render the active tetrahedra and perform ray-surface intersections during fragment processing. Since all tetrahedra of one particular type have the same shape, but different positions within the grid, we use geometry instancing to render all tetrahedra of one type at once. Each of the 144 different shapes is represented as a triangle strip with six vertices. The instance ID is used to index into the previously computed lists $\mathbf{TO}_{\text{Comp}}$. The vertex shader reads the value of $\mathbf{TO}_{\text{Comp}}$ from the appropriate texture at the position defined by the current instance ID and translates the index into the 3D world space coordinates of the current tetrahedron.

For later calculation of the ray-surface intersections in the fragment shader, as well as clipping of intersections to the tetrahedron geometry, the vertex shader further prepares two barycentric coordinates $\phi_i, \widetilde{\phi}_i$, $i = 0,1,2,3$, for each tetrahedron vertex $\mathbf{v}_\mu$, $\mu = 0,1,2,3$. The $\phi_i = \phi_i(\mathbf{v}_\mu)$ are given by $\delta_{i,\mu}$. The $\widetilde{\phi}_i$ correspond to the unit length extension of the viewing ray and are calculated by using the inverse of the matrices A_T from Eq. 3. The 144 different matrices A_T^{-1} are precomputed once and stored on the GPU in constant memory. The barycentrics are then interpolated across the front-facing triangles of T in the rasterizer.

The fragment shader first calculates the b_{ijkl} of the current patch from the volume texture by using the appropriate weightings of the local neighborhood, see Section 3. Note that the b_{ijkl} could also be computed in the vertex shader, but we observed that for large volumes, the number of tetrahedron vertices approaches the number of fragments and computing the b_{ijkl} in the fragment shader gives us a performance gain. Next, we perform blossoming to obtain a univariate representation of the surface patch along the viewing ray: A de Casteljau step on the first level is performed with ϕ to obtain the four coefficients $b^{[1]}_{ijkl}(\phi)$, $i+j+k+l=1$, as well as for $\widetilde{\phi}$ resulting in $b^{[1]}_{ijkl}(\widetilde{\phi})$, $i+j+k+l=1$.

We proceed with three more steps on the second level of the de Casteljau algorithm. Note that each de Casteljau step on the second level corresponds to a dot product of two vectors in $\mathbb{R}^4$. We thus obtain the three coefficients $b_{20} = b^{[1]}_{ijkl}(\phi)\cdot\phi$, $b_{11} = b^{[1]}_{ijkl}(\phi)\cdot\widetilde{\phi}$, and $b_{02} = b^{[1]}_{ijkl}(\widetilde{\phi})\cdot\widetilde{\phi}$, which are the coefficients of a functional quadratic Bézier curve restricted along the viewing ray. The intersection is found by plugging the ray into that curve and solving for the roots $t_{0,1}$ of the resulting quadratic polynomial in one variable. In order to restrict the intersections to the tetrahedron geometry, we calculate the

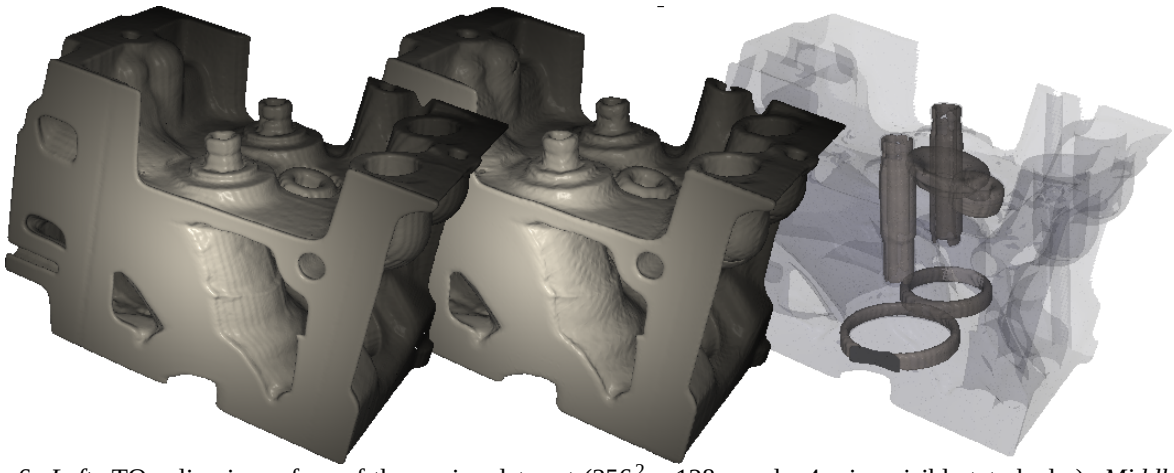


Figure 6: *Left*: TO spline isosurface of the engine data set ($256^2 \times 128$ voxels, 4 mio. visible tetrahedra). *Middle*: TO spline isosurface of a subsampled version of the original data set ($128^2 \times 64$ voxels, 1.4 mio. visible tetrahedra). *Right*: Isosurfaces with approximated transparency.

barycentric coordinates $\phi(t_0)$ and $\phi(t_1)$ by linear interpolation of ϕ and $\widetilde{\phi}$ using t_0 and t_1, respectively, as parameters. We take the smallest t with $\phi_i(t) \geq 0$, $i = 0,1,2,3$, if such a t exists, and discard the fragment otherwise.

We get the directional derivative w.r.t. the ray at the intersection point, $b^{[1]}_{ijkl}(\phi(t))$, $i+j+k+l=1$, by a linear interpolation of $b^{[1]}_{ijkl}(\phi)$ and $b^{[1]}_{ijkl}(\widetilde{\phi})$ with t. The gradient at the intersection point is finally obtained by three additional scalar products in $\mathbb{R}^4$, using the first three rows of A_T^{-1} and the directional derivative $b^{[1]}_{ijkl}(\phi(t))$.

An example of the resulting spline surface within its bounding truncated octahedra, as well as one enlarged polynomial patch and its bounding tetrahedron, is given in Fig. 5a. Currently, the fragment's z value corresponds to a front facing triangle of T, and not the actual surface point. This is usually not a problem if only a single surface is visualized. For proper clipping of several surfaces, we normalize the distance from the eye-plane to the intersection into the z-buffer and achieve the result shown in Fig. 5b, right. We further applied an approximation of order independent transparency by using weighted sums for each pixel.[12] An example is shown in Fig. 6, right.

5. RESULTS

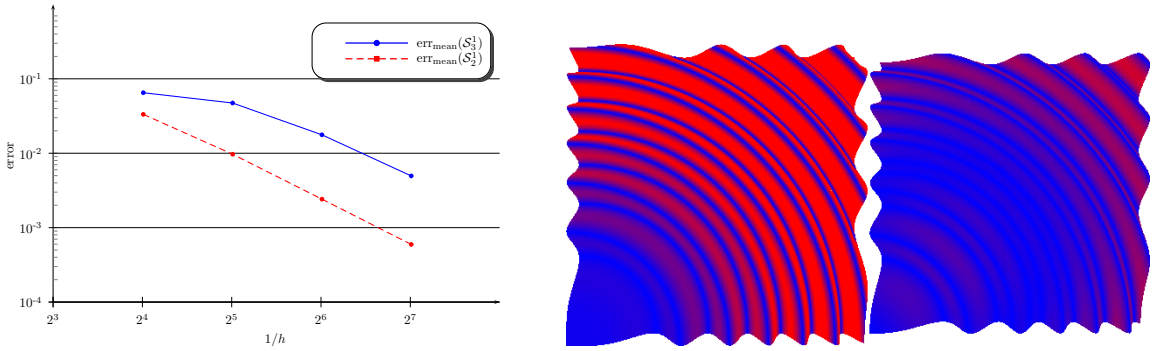
In this section, we discuss some of the results of our rendering scheme for the new quadratic C^1 spline on TO partitions ($\mathcal{S}^1_2$). We further compare $\mathcal{S}^1_2$ with trilinear interpolation and the closely related cubic C^1 splines on type-6 partitions ($\mathcal{S}^1_3$).

5.1 Numerical Approximation

As shown in[4,8] the new $\mathcal{S}^1_2$ spline, as well as the $\mathcal{S}^1_3$ spline, both approximate smooth functions with order two, i.e., the error goes down by a factor of four if the grid spacing is halved, but the constants are slightly better for the new $\mathcal{S}^1_2$ spline. We further visually compare the approximation error of the new spline with the $\mathcal{S}^{\frac{1}{3}}$ spline in Figure 7b, showing the reconstruction of the highly oscillating Marschner-Lobb function,[13] sampled from a sparse 64^3 grid. Here, the colors indicate the error varying from blue (low error) to red (higher error), showing that the new quadratic spline reconstructs the function with lower error. The errors for the Marschner-Lobb function are also summarized in the graph in Fig. 7a.

5.2 Visualization Performance

We analyzed the quadratic spline with a series of tests. First, we note that for typical isosurfaces, about 1% to 5% of all tetrahedra within the volumetric domain Ω are active, see also Fig. 1a-c and Fig. 6 for examples. Each TO covers four data points of Ω, which, on average, results in 36 tetrahedra per data point. In contrast to this, in the type-6 partition each data cube is covered by 24 tetrahedra. In fact, we have about one half more tetrahedra for the same isolevel as for the $\mathcal{S}^{\frac{1}{3}}$ splines on type-6 partitions, see Table 1. Further, for on-the-fly computation of coefficients in the visualization, we need 28 ($\mathcal{S}^{\frac{1}{2}}$), and 23 texture accesses ($\mathcal{S}^1_3$), respectively.



(a) (b)

Figure 7: (a) Comparison of the reconstruction errors for the Marschner-Lobb (ML) function with increasing sample rates on a log-log scale. Blue curve: mean error of the cubic C^1 spline on type-6 partitions ($\mathcal{S}_3^1$). Dashed red curve: mean error of the new quadratic C^1 spline ($\mathcal{S}_2^1$). (b) Reconstructed ML function sampled on a 64^3 grid. *Left*: $\mathcal{S}_3^1$ spline. *Right*: $\mathcal{S}_2^1$ spline. Red denotes higher error, blue low error.

grid size	cubic type-6 spline ($\mathcal{S}_3^1$) #tets	reconst.	FPS	quadratic TO spline ($\mathcal{S}_2^1$) #tets	reconst.	FPS
64^3	13 398	16 [ms]	125	19 224	19 [ms]	59
128^3	711 000	107 [ms]	25	1 061 000	131 [ms]	26
128^3	2 316 000	123 [ms]	8	3 388 000	180 [ms]	9
256^3	4 153 000	95 [ms]	4	5 716 000	192 [ms]	5
256^3	5 371 000	88 [ms]	3	8 066 000	210 [ms]	3

Table 1: Reconstruction times (computation of the active partition) and FPS for selected isosurfaces with increasing number of active tetrahedra. *Left*: cubic C^1 splines on type-6 partitions. *Right*: quadratic C^1 splines on TO partitions. All timings were done with a NVIDIA GTX 480 GPU with 480 shader cores and on a 1000×1000 view port.

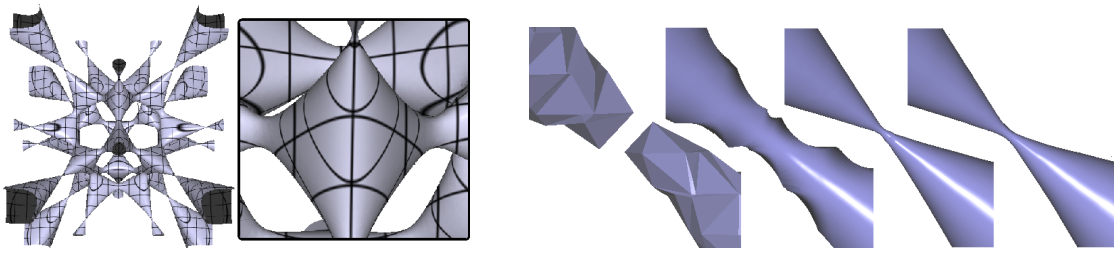
Table 1 also shows the reconstruction times for the computation of the active partition, see Section 4.1, as well as the frames per second on a 1000×1000 view port. Note that reconstruction times are in the range of a few milliseconds. For both splines, the frames per second are directly connected to the number of tetrahedra in the active partition. Note that the higher number of tetrahedra of the $\mathcal{S}_2^1$ splines is in most cases compensated for by the significantly less complex shaders, because $\mathcal{S}_3^1$ requires to evaluate and solve cubic equations for ray-surface intersections instead of quadratics.

5.3 Visualization Results

We demonstrate the new quadratic spline with visualizations of real-world data from engineering (Fig. 6), biological (Fig. 1b), and medical applications (Fig. 1c). Isosurfaces reconstructed from synthetic data are shown in Fig. 1a, where we used a signed distance function generated from a triangle mesh. Fig. 8a shows a reconstruction of *Barth's sextic* function sampled on a uniform grid. In Fig. 8b we give a comparison of isosurfaces obtained by Marching Cubes, simple ray casting with trilinear interpolation, cubic C^1 splines, and quadratic C^1 splines, respectively. The zoom up into one of the spikes of Barth's sextic from Fig. 8a demonstrates the improved quality of smooth splines, in particular for the new quadratic spline.

6. CONCLUSIONS

We have shown that smooth splines on uniform tetrahedral partitions can be used for interactive and high-quality visualizations of isosurfaces from measured data. In particular, we have shown the first visualizations of a new spline, which is a local method to obtain a global C^1 approximation of volume data by quadratic polynomials. We compared the performance of our method with the closely related quasi-interpolating cubic C^1 splines on type-6 tetrahedral partitions. Our quadratic C^1 spline leads to significantly lower shader complexity which in most cases compensates for the higher tetrahedron count of the TO partitions. Frame rates are therefore competitive, and in many cases even better than with the cubic C^1 spline. We further visually compared the new spline to standard methods for isosurface visualization, such as Marching Cubes and trilinear interpolation, showing the improved quality of a smooth reconstruction.



(a) (b)

Figure 8: (a) Reconstruction of Barth's sextic sampled from a 128^3 grid. (b) Zoom ups into one of the spikes of Barth's sextic. *From left to right*: Marching Cubes, trilinear ray casting, S_3^1 spline, the new S_2^1 spline.

The splines we used are direct methods, where the polynomial coefficients are directly available from the volume data by simple averages of the data values in a local neighborhood. The polynomials can thus be computed on-the-fly directly on the GPU without the need to inflate the data for storing pre-computed coefficients. We have shown the limits of a cell projection approach for ray casting a large number of tetrahedra in B-form. First tests with a direct and *geometry free* approach for volume ray casting of type-6 splines, where we do not project single tetrahedra, have shown that speedups of more than one order of magnitude can be achieved using the same hardware.

Acknowledgments. This research was supported in part by the DFG project GZ ZE 443/7-1, AOBJ: 526229 and the DFG Emmy Noether fellowship GO 1752/3-1. Data sets in Fig. 1b,c are courtesy of C. Bajaj, University of Texas. Data set in Fig. 6 is courtesy of General Electric.

REFERENCES

[1] Lorensen, W. E. and Cline, H. E., "Marching cubes: A high resolution 3D surface construction algorithm," *SIGGRAPH Comput. Graph.* **21**(4), 163–169 (1987).

[2] NVIDIA Corporation, "CUDA software development kit: Version 2.1." `http://www.nvidia.com/object/cuda_get.html` (2009).

[3] Rössl, C., Zeilfelder, F., Nürnberger, G., and Seidel, H.-P., "Visualization of volume data with quadratic super splines," in [*Proceedings VIS*], 393–400 (2003).

[4] Sorokina, T. and Zeilfelder, F., "Local quasi-interpolation by cubic C^1 splines on type-6 tetrahedral partitions," *IMJ Numerical Analysis* **27**, 74–101 (2007).

[5] Kalbe, T. and Zeilfelder, F., "Hardware-accelerated, high-quality rendering based on trivariate splines approximating volume data," *Computer Graphics Forum* **27**(2), 331–340 (2008).

[6] Kalbe, T., Koch, T., and Goesele, M., "High-quality rendering of varying isosurfaces with cubic trivariate C^1-continuous splines," *Proc. of 5th International Symposium on Visual Computing* , 596–607 (2009).

[7] Kloetzli, J., Olano, M., and Rheingans, P., "Interactive volume isosurface rendering using bt volumes," in [*Proc. Symp. on Interactive 3D graphics and games*], 45–52 (2008).

[8] Rhein, M. and Kalbe, T., "Quasi-interpolation by quadratic C^1-splines on truncated octahedral partitions," *Computer Aided Geometric Design* **26**(8), 825–841 (2009).

[9] Seidel, H.-P., "An introduction to polar forms," *IEEE Computer Graphics & Applications* **13**(1), 38–46 (1993).

[10] Blelloch, G. E., "Prefix sums and their applications," tech. rep., Carnegie Mellon University (1990).

[11] Harris, M., Sengupta, S., and J.D.Owens, "Parallel prefix sum (scan) with CUDA," in [*GPU Gems III*], Nguyen, H., ed., 851–876, Addison-Wesley (2008).

[12] Meshkin, H., "Sort-independent alpha blending," tech. rep., Perpetual Entertainment (2007).

[13] Marschner, S. R. and Lobb, R. J., "An evaluation of reconstruction filters for volume rendering," in [*Proceedings VIS*], 100–107 (1994).