Hardware-Accelerated Volume And Isosurface Rendering Based On Cell-Projection

Stefan Röttger, Martin Kraus, Thomas Ertl

Visualization and Interactive Systems Group
Universität Stuttgart, Germany*

Abstract

We present two beneficial rendering extensions to the Projected Tetrahedra (PT) algorithm by Shirley and Tuchman. These extensions are compatible with any cell sorting technique, for example the BSP-XMPVO sorting algorithm for unstructured meshes.

Using 3D texture mapping our first extension solves the long-standing problem of hardware-accelerated but accurate rendering of tetrahedral volume cells with arbitrary transfer functions.

By employing 2D texture mapping our second extension realizes the hardware-accelerated rendering of multiple shaded isosurfaces within the PT algorithm without reconstructing the isosurfaces.

Additionally, two methods are presented to combine projected tetrahedral volumes with isosurfaces. The time complexity of all our algorithms is linear in the number of tetrahedra and does neither depend on the number of isosurfaces nor on the employed transfer functions.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation, I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling, I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism.

Keywords: Volume Rendering, Isosurfaces, Unstructured Meshes, Cell Projection, Graphics Hardware, Texture Mapping, Compositing.

1 Introduction

Ten years ago direct volume rendering of unstructured tetrahedral meshes was dramatically accelerated by the Projected Tetrahedra (PT) algorithm by Shirley and Tuchman [21], which is summarized in Section 2. Although there are numerous competing approaches to direct volume rendering of unstructured meshes, e.g. ray casting [22], slicing [35], or sweep-plane algorithms [27], several aspects of the PT algorithm are still subject of current research, e.g. the sorting of tetrahedral cells (see [5] and references therein). Our extensions of the PT algorithm are restricted to the rendering of projected tetrahedra.

The original PT algorithm approximates the opacity and color between vertices linearly resulting in Mach bands as reported by Max et al. in [14]. Stein et al. presented a solution for the correct interpolation of opacities utilizing 2D texture mapping in [24], which is also discussed in Section 2. However, this method is restricted to linear transfer functions for the opacity and still interpolates color components linearly ignoring the transfer functions for them inside the tetrahedra.

Our first improvement of the PT algorithm allows us to render both, opacity and color, accurately by exploiting 3D texture mapping. In particular this method allows us to employ arbitrary transfer functions. The method and its application to a volume density optical model is described in Section 3. In Section 4 we derive an approximate rendering method based on 2D texture mapping, which is supported by considerably more graphics systems and requires less texture memory.

A second extension allows us to include the rendering of isosurfaces in the PT algorithm using 2D texture mapping without extracting a polygonal representation of the isosurfaces. There are numerous algorithms to display isosurfaces efficiently. We will mention a selection in Section 5. However, none of these algorithms takes any particular advantage of the PT algorithm. Therefore, the costs of displaying an isosurface were not reduced by a combination with the PT algorithm in the past.

Our approach, however, reuses the visibility ordering and the decomposition of the tetrahedral cells, which are an essential part of every variant of the PT algorithm. There are many efficient algorithms for the visibility ordering (see [5]), which all appear to be compatible with our rendering extensions. By reusing the ordering and decomposition of tetrahedra our method is capable of rendering isosurfaces without constructing a polygonal representation. As it is conceptually similar to the first pass of the multi-pass algorithm for smoothly shaded isosurfaces by Westermann and Ertl [28], we present a variant of this first pass in Section 6. We employ this idea in the context of the PT algorithm and present a specialized single-pass algorithm for flat-shaded isosurfaces using 2D texture mapping in Section 7. Moreover, a two-pass algorithm for smoothly shaded isosurfaces is described in Section 8.

Extensions for colored and multiple isosurfaces are discussed in Section 9, while Section 10 presents two methods for mixing isosurfaces with projected volume cells, either approximately but smoothly using appropriate blending and texture mapping or more accurately by modifying the texture maps.

We emphasize that the worst-case time complexities of all our methods, i.e. volume rendering with arbitrary transfer functions, rendering of multiple and smoothly shaded isosurfaces, and mixing of isosurfaces with projected volume cells, are linear in the number of tetrahedra and neither depend on the transfer functions nor on the number of isosurfaces.

*Universit¨at Stuttgart, IfI, Abt. VIS, Breitwiesenstr. 20-22, 70565 Stuttgart, Germany; E-mail: {Stefan.Roettger | Martin.Kraus | Thomas.Ertl}@informatik.uni-stuttgart.de.

2 The PT Algorithm

The PT algorithm visualizes a scalar function $f(x,y,z)$ defined over a region of three-dimensional space by rendering partially transparent polygons, which can be processed very quickly by specialized graphics hardware.

The PT algorithm can be summarized as follows (see also [21]):

1. Decompose the volume into tetrahedral cells. Scalar values are defined at each vertex of the mesh. Inside each tetrahedral cell, $f(x,y,z)$ is assumed to be a linear combination of the vertex values.
2. Sort the cells according to their visibility.
3. Classify each tetrahedron according to its projected profile and decompose it into smaller triangles (see Figure 1).

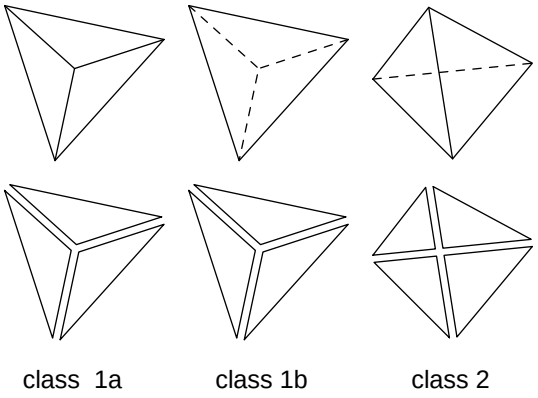

Figure 1: Classification of non-degenerate projected tetrahedra (top row) and the corresponding decompositions (bottom row) according to [21].

4. Find color and opacity values for the triangle vertices using ray integration.
5. Render the triangles.

In the reminder of this section and in Sections 3 and 4 we will only discuss methods to improve the last two points: ray integration and rendering of the decomposed triangles with emphasis on hardware-accelerated rendering.

The original PT algorithm interpolates color and opacity linearly between the triangle vertices. This, however, is an approximation which leads to rendering artifacts as demonstrated in [14, 24].

In order to avoid these artifacts Stein et al. suggested in [24] to use a 2D texture map with the texture coordinates being the averaged extinction coefficient τ and the thickness l of the projected cell, while the texture map contains an α-component which is set to $\alpha = 1 - \exp(-\tau l)$. In between the vertices of each triangle the texture coordinates and, therefore, τ and l are interpolated linearly; thus, this approach is restricted to a linearly varying extinction coefficient τ, i.e. a linear transfer function $\tau = \tau(f(x,y,z))$. Moreover, the color is still linearly interpolated between vertices. Williams et al. extended these ideas to piecewise linear transfer functions in [32].

3 PT with Accurate Opacity and Color

In this section a generalization of the method of Stein is presented which works for color and opacity, and places no restrictions on the transfer functions. We achieve these benefits by employing 3D texture mapping.

Let us start by investigating the situation depicted in Figure 2.

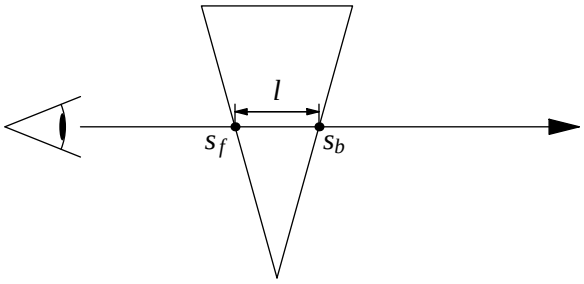

Figure 2: Intersecting a tetrahedral cell with a viewing ray. s_f and s_b are the scalar values on the entry (front) and exit (back) face respectively; l denotes the thickness of the cell for this ray.

As texture coordinates are interpolated linearly, we should only use variables, the values of which vary linearly with screen coordinates. We will restrict our considerations to orthographic projections. In this case l varies linearly for geometric reasons; s_f and s_b vary linearly because they are interpolated linearly between vertices as mentioned above. Therefore, s_f, s_b, and l should be the three texture coordinates. Fortunately, all other values, e.g. color, opacity, etc., can be calculated from l, s_f, and s_b. Thus, we can set up a 3D texture map which contains the color and opacity characterizing the intersection of a ray and a cell in dependency of l, s_f, and s_b.

For many applications the calculation of the texture map is a preprocessing step and, therefore, not time-critical. Usually it includes a numerical integration of a ray for each texel in the 3D texture map. We sketch the procedure for the volume density optical model proposed by Williams and Max [13, 31, 32] with a chromaticity vector $\kappa = \kappa(f(x,y,z))$ and a scalar optical density $\rho = \rho(f(x,y,z))$, which are the transfer functions of this model.

Assuming cells are processed back to front, the addition of the projection of a cell changes an existing pixel color I to a new pixel color I' by the formula (compare Equation (4) in [31])

$$I' = \underbrace{\int_0^l \exp\left(-\int_0^t \rho(s_l(u))\mathrm{d}u\right)\kappa(s_l(t))\rho(s_l(t))\mathrm{d}t}_{\mathrm{RGB_{t3D}}} + \underbrace{\exp\left(-\int_0^l \rho(s_l(t))\mathrm{d}t\right)}_{1-\alpha_{\mathrm{t3D}}} \times I \tag{1}$$

with the abbreviation

$$s_l(x) = s_f + \frac{x}{l}(s_b - s_f).$$

$\mathrm{RGB_{t3D}}$ denotes the color components (note that κ is a vector), and α_{t3D} the opacity of an entry in the 3D texture map. $\mathrm{RGB_{t3D}}$ and α_{t3D} depend on the texture coordinates l, s_f, s_b, and the transfer functions κ and ρ. Thus, the texture map has to be updated whenever the transfer functions are modified.

It is an intrinsic limitation of our method that κ and ρ have to depend on the same scalar field. However, we are not limited to this optical model; for example the model of Wilhelms and Van Gelder [13, 29, 32] could be implemented by simply replacing $\kappa(s_l(t))\rho(s_l(t))$ by a differential color vector $E(s_l(t))$ (or $g(s_l(t))$ in the notation of [13, 32]).

After the calculation of the texture map in a preprocessing step, all tetrahedra are projected from back to front. Before rendering the triangles of one projected tetrahedron, the three texture coordinates are set for each vertex of the triangles. Then they are blended appropriately into the frame buffer. The blending operation corresponds to

$$I' = \mathrm{RGB}_{\mathrm{t3D}} + (1 - \alpha_{\mathrm{t3D}}) \times I,$$

and is done very efficiently by today's graphics hardware.

We give a synthetic example of this rendering method in Figure 3. The scalar values at the vertices of the visualized tetrahedral mesh are determined by the distance of each vertex to the surface of a sphere. The transfer function of the opacity is 0 except for a small interval, which results in the two partially opaque rings in Figure 3.

In summary our method allows us to exploit hardware-supported 3D texture mapping in order to render projected tetrahedra without the need to do any time consuming calculations for each pixel. Our approach is not as accurate as ray-casting algorithms or the high accuracy (HIAC) volume rendering system described in [32] because of limited texture memory and non-linear transformations in the case of perspective projections. Especially limited texture memory will limit the size of the 3D texture map resulting in a less accurate resampling of the transfer functions. Within this limited accuracy, however, arbitrary transfer functions may be used without affecting the rendering times, whereas the performance of the HIAC system depends crucially on the chosen transfer functions. In particular, thin peaks are possible within our approach resulting in unshaded isosurfaces as demonstrated in the appendix.

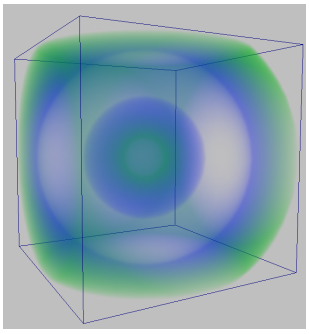

Figure 3: Visualization of a synthetic data set with non-linear transfer functions implemented with a 3D texture map of dimensions $64 \times 64 \times 64$ (1 MB).

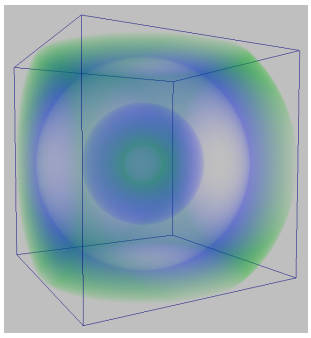

Figure 4: Same data set as in Figure 3 but rendered using a 2D texture map of dimensions 256×256 (256 KB). (See Section 4.)

4 A New Approximation for PT

As hardware-supported 3D texture mapping is only available on a few graphics workstations, and the 3D texture maps that are employed in Section 3 need rather much texture memory, we will describe a new approximation to the rendering of projected tetrahedra using 2D texture mapping, which interpolates the opacity linearly. However, this method allows us to use arbitrary transfer functions, while existing hardware-accelerated solutions are limited to linear transfer functions within each cell (e.g. [24]).

The basic idea is to approximate the dependencies of the integrals in Equation (1) on l by linear terms, and to implement these terms by a modulation of the vertex colors. The remaining integrals depend only on s_f and s_b, and can thus be tabulated in a 2D texture map.

The dependencies on l in Equation (1) become more explicit with the variable substitutions $t' = t/l$ and $u' = u/l$:

$$\begin{aligned} I' &= l \int_0^1 \exp\left(-l \int_0^{t'} \rho(s_1(u'))\mathrm{d}u'\right) \\ &\quad \times \kappa(s_1(t'))\rho(s_1(t'))\mathrm{d}t' \\ &\quad + \exp\left(-l \int_0^1 \rho(s_1(t'))\mathrm{d}t'\right) \times I. \end{aligned}$$

For $l = 0$ this equation reduces to $I' = I$. For given ρ, κ, s_f and s_b we evaluate the integrals for another value $l = \bar{l} = \text{const.}$ and extrapolate linearly in l. The optimal choice of $\bar{l}$ depends on the particular application but setting $\bar{l}$ equal to the average cell thickness has proven to be a good approximation. The 2D texture map is defined by

$$\begin{aligned} \mathrm{RGB}_{\mathrm{t2D}} &= \bar{l} \int_0^1 \exp\left(-\bar{l} \int_0^{t'} \rho(s_1(u'))\mathrm{d}u'\right) \\ &\quad \times \kappa(s_1(t'))\rho(s_1(t'))\mathrm{d}t', \\ \alpha_{\mathrm{t2D}} &= 1 - \exp\left(-\bar{l} \int_0^1 \rho(s_1(t'))\mathrm{d}t'\right) \end{aligned} \tag{2}$$

and is modulated by colors at the vertices with the RGBα components set equal to $(l/\bar{l}, l/\bar{l}, l/\bar{l}, l/\bar{l})$. In practice we are scaling these colors by the maximum opacity value in the texture map in order to avoid clamping for values $l > \bar{l}$. This scaling is compensated by multiplying the entries in the texture map with the reciprocal value. The combined effect of texturing and blending with appropriate blending coefficients (e.g. in OpenGL `GL_ONE` for the source blend factor and `GL_ONE_MINUS_SRC_ALPHA` for the destination blend factor) is

$$I' = \frac{l}{\bar{l}} \times \mathrm{RGB}_{\mathrm{t2D}} + \left(1 - \frac{l}{\bar{l}} \times \alpha_{\mathrm{t2D}}\right) \times I,$$

which is our new approximation of Equation (1).

On the one hand, this approximation results in artifacts because of the linear interpolation (see [24]), on the other hand, the use of 2D texture mapping enables us to utilize larger texture maps compared with the 3D texture maps employed in Section 3 resulting in an improved resampling of the transfer functions.

Figure 4 shows the synthetic example from Figure 3 using 2D instead of 3D texture mapping. The linear approximation results in slightly smaller opacities resulting in lighter colors, while the improved resampling results in sharper edges of the structures generated by the transfer functions. The middle image in Figure 13 represents an example of a 2D texture map generated by Equation (2).

The following sections discuss an independent extension of the PT algorithm capable of displaying smoothly shaded isosurfaces without vertex interpolations. Additionally, two methods are presented to combine projected tetrahedra with opaque isosurfaces.

5 Prior Work about Isosurfaces

For an in-depth introduction into current research about isosurfaces the reader is referred to [1]. Isosurfaces are an indispensable tool in volume visualization, although direct volume rendering includes much more information in one picture. However, isosurfaces are preferred for many applications as they are usually more comprehensible. Thus, direct volume rendering techniques are often extended with isosurfaces in order to combine the advantages of both techniques.

In their description of the PT algorithm [21] Shirley and Tuchman suggested to calculate isosurfaces based on a marching tetrahedra algorithm similar to the marching cubes algorithm [12, 34]. The combination of these algorithms makes it possible to render isosurfaces with any degree of transparency as noted in [21].

However, research on marching cells algorithms concentrated on reducing the number of cells tested for intersections with the isosurface [2, 3, 4, 11, 19, 20, 30] and on simplifying the polygonal mesh representing the isosurface [7, 10, 15, 16, 18].

Instead of reducing the number of polygons point-based algorithms for the extraction of isosurfaces [6, 9, 17, 23] do not produce any polygons. Westermann's multi-pass algorithm for shaded isosurfaces [28] also does not construct a polygonal representation of the isosurface. As our algorithm is based on the same idea, we present the common concept in Section 6 before discussing our algorithm in Section 7.

6 A Hardware-Accelerated Marching Cells Algorithm

This section discusses a variant of the first pass of Westermann's algorithm for shaded isosurfaces in unstructured grids [28]. The algorithm presented here sets all pixels of the silhouette of an intersection of an isosurface with a tetrahedral cell. Figure 5a shows the resulting silhouette, while Figures 5b and 5c show intermediate steps of the algorithm.

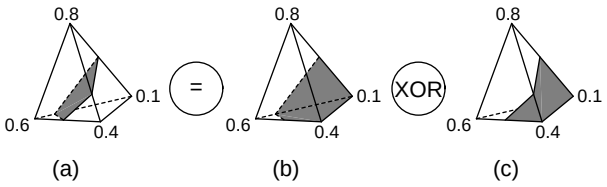

Figure 5: The polygon of an isosurface (isovalue 0.5) within a tetrahedral cell (a) can be obtained by an XOR combination of the two pictures (b) and (c). (b) shows the parts of the back faces of the cell with scalar value less than 0.5. (c) is the analogue to (b) for the front faces.

The first step is to render those parts of the back faces of the cell where the interpolated scalar value is less than the isovalue (see Figure 5b). Utilizing OpenGL this can be achieved by setting the α-components of the vertices' color to the scalar values and activating an appropriate α-test. Then the front faces are rendered in exactly the same way, i.e. again only those parts are rendered where the interpolated scalar value is less than the isovalue (see Figure 5c). By combining both pictures with an exclusive-OR (XOR) operation the correct set of pixels is obtained. Using OpenGL an XOR operation can be realized with the help of a 1-bit stencil buffer by inverting its contents whenever a pixel passes the α-test.

Note that the result is not sensitive to the order of the polygon rendering, i.e. the back and front faces could be rendered in any order. The result is also the same if the α-test is inverted for all faces, i.e. if those parts of the polygons are rendered where the interpolated scalar value is greater than the isovalue. Westermann's original algorithm differs in so far as the α-test is inverted for the back faces only and the pictures are combined with an AND-operation. However, this requires additional passes in order to generate both faces of the isosurface.

In summary this algorithm requires the rendering of all front and back faces in order to set the stencil buffer and to render either the front or the back faces once more for flat-shaded isosurfaces. Thus, for a tetrahedral cell five to seven triangles have to be rendered, while a polygonal representation of the isosurface in a tetrahedron needs only one or two triangles. Therefore, the advantage of interpolating the scalar data with the help of OpenGL hardware is more than compensated by the need to render additional polygons.

The situation is, however, fundamentally different in the context of the PT algorithm as will be discussed in the following section.

7 Flat-Shaded Isosurfaces

As mentioned in Section 2 the PT algorithm [21] triangulates the projection of tetrahedra as shown in Figure 1. However, instead of refering to a triangulation of the projected silhouette into triangles, we can as well think of a decomposition of the original tetrahedron into smaller tetrahedra, which are projected after the decomposition. The projections of these smaller tetrahedra are all of the same kind: Two faces are degenerate and the other two faces are projected onto the same (non-degenerate) triangle. This observation enables us to reduce the algorithm presented in Section 6 to a single-pass algorithm for these tetrahedra using 2D texture mapping.

As explained in Section 6 pixels are set if and only if the interpolated scalar value of either the back or the front face is less than the isovalue. As noted the back and front face are projected onto the same triangle. Therefore, it is sufficient to render this triangle using a checkerboard-like, two-dimensional texture map as shown in the right-hand column of Figure 6 with the two texture coordinates corresponding to the interpolated scalar value of the back and front face, respectively. (See the appendix for an alternative derivation of this 2D texture map.)

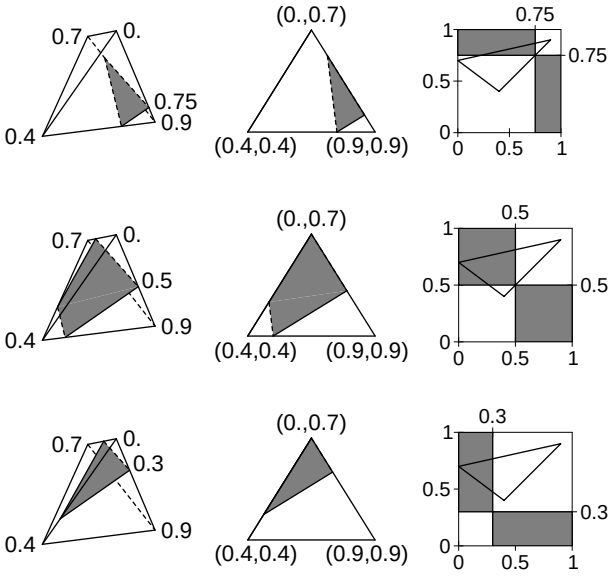

Figure 6: Projected tetrahedra (middle column) with flat-shaded isosurfaces for isovalues 0.75 (top row), 0.5 (middle row), and 0.3 (bottom row). The left-hand column shows the same tetrahedra slightly rotated with scalar data at the vertices. These values define the texture coordinates included in the pictures of the actual projections in the middle column. The right-hand column shows the corresponding texture maps including the triangles in the space of texture coordinates.

The first texture coordinate corresponds to the scalar value on the front face and the second texture coordinate to the scalar value on the back face. As the scalar data are interpolated linearly, the texture

coordinates should also be interpolated linearly. Perspective corrections of texture coordinates should, therefore, be disabled. Actual values of texture coordinates have to be specified at the vertices of the triangle and are determined by the scalar data defined at the vertices of the projected tetrahedron. (See the left-hand column in Figure 6 for the scalar data defined at the vertices of the tretrahedron and the middle column for the resulting pairs of texture coordinates at the vertices of the projected triangle.) If the scalar data are not in the appropriate range for texture coordinates, the values have to be scaled accordingly. However, this can be done in a preprocessing step.

The texture itself has to determine the α-component, i.e. the opacity, which has to be 1 for opaque isosurfaces whenever either the first or the second texture coordinate is less than the isovalue, and 0 otherwise (see the right-hand column of Figure 6). As this pass does not allow any kind of smooth shading, we employ flat shading, i.e. the RGB-components of the color of the triangle are constant.

Unfortunately, edges of isosurface patches within triangles (see the middle column of Figure 6 for some examples) will cause rendering artifacts as there is no mechanism which aligns them exactly to the corresponding edges in the projected tetrahedra in front or behind. We can avoid gaps by slightly modifying the texture map, effectively 'thickening' the isosurface. This eliminates artifacts for opaque isosurfaces; for partially transparent isosurfaces, however, this will visually enhance edges of the tetrahedral mesh by rendering pixels twice. In fact these edges help to comprehend the three-dimensional structure of flat-shaded isosurfaces. Nonetheless, removing these artifacts for partially transparent isosurfaces is an open problem and requires additional efforts in the future.

8 Smoothly Shaded Isosurfaces

Our algorithm for smoothly shaded isosurfaces is again a variant of the corresponding passes of Westermann's algorithm for shaded isosurfaces in unstructured grids [28]; however, there are several crucial differences. For each triangle the steps of our algorithm are:

1. Render the shaded back face triangle restricted to the isosurface silhouette as discussed in Section 7.

2. Repeat the preceding step for the front face triangle.

3. Form the weighted sum of the two pictures.

The weights differ for each pixel as they depend on the relative distances of the isosurface to the front and back face, respectively (see Figure 7). For reasons which will become clear in the next paragraph, let α denote the weight of a pixel of the front triangle. According to Figure 7 the weight α is

$$\alpha = \frac{s_{\text{iso}} - s_b}{s_f - s_b} \quad \text{for} \quad s_f < s_{\text{iso}} < s_b \quad \text{or} \quad s_f > s_{\text{iso}} > s_b$$

with the isovalue s_{iso}; s_f and s_b were defined in Section 3. The weight of a corresponding pixel on the back face triangle is $1-\alpha$. While weights for all pixels were calculated in software in [28], we are calculating the weighted sum completely in hardware.

We still use the 2D texture map of Section 7 for the back face triangle but employ a modified version of this texture map for the front face triangle. This new texture map (see Figure 8 for an example) is modulated with the weights α. As the original texture map contains only opacity values 0 and 1, this modulated map in fact stores the weights $\alpha = \frac{s_{\text{iso}} - s_b}{s_f - s_b}$ for the front face triangle. (Remember that s_f and s_b are the texture coordinates and that the texture map already depends on s_{iso}.) Thus, the weights α in fact specify opacities. Using this texture map when rendering the front face triangle and blending it appropriately onto the opaque back face triangle generates, therefore, the correct weighted sum of both triangles.

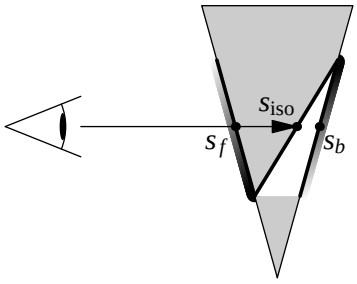

Figure 7: Rendering smoothly shaded isosurfaces by shading the back and front face triangle, and forming the weighted sum. Weights are symbolized by gray scales and are determined by the relative distances of the front and back faces to the isosurface given by $(s_{\text{iso}} - s_b)/(s_f - s_b)$ and $(s_{\text{iso}} - s_f)/(s_b - s_f)$ respectively.

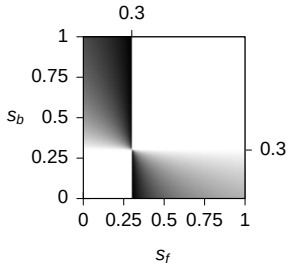

Figure 8: A 2D texture map used for a front face triangle; black corresponds to opacity 1 (opaque), white to opacity 0 (transparent). It is a modulation of the lower texture map in Figure 6 with the weights $\alpha = \frac{s_{\text{iso}} - s_b}{s_f - s_b}$ and $s_{\text{iso}} = 0.3$.

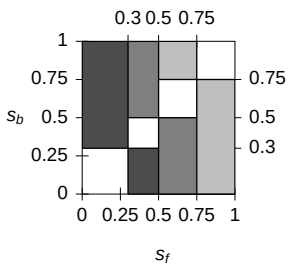

Figure 9: The correct combination of the texture maps from Figure 6 into a single texture map for multiple isosurfaces. (See Section 9.)

Thus, our algorithm for smoothly shaded isosurfaces can be reformulated in two passes for each tetrahedron:

1. Render the shaded back face triangle restricted to the isosurface silhouette. (See Section 7.)

2. Blend the shaded front face triangle modulated with a texture map containing the correct weights onto the back face triangle.

Special care has to be taken with vertices from the decomposition of projected tetrahedra, because they can result in artifacts similar to those induced by hanging nodes. Therefore, the color of a vertex inserted between two vertices of the mesh has to be equal to the color generated by the graphics hardware interpolating between these vertices.

The algorithm was used in Figure 10 to render several isosurfaces of different colors as explained in the following section.

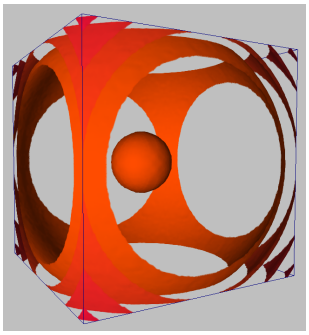

Figure 10: Several isosurfaces extracted from the data set shown in Figures 3 and 4.

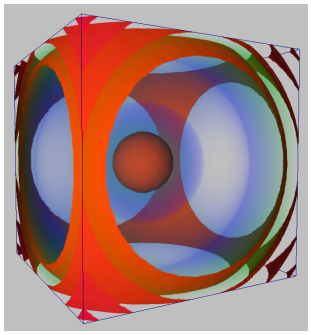

Figure 11: Smooth combination of Figures 4 and 10. (See Section 10.)

9 Colored and Multiple Isosurfaces

The techniques presented in Sections 7 and 8 can be extended to colored and multiple isosurfaces. Coloring can be achieved by setting the vertex colors to white and modulating them with colored RGBα texture maps. The two faces of an isosurface can be colored independently by choosing different colors for texels with $s_f > s_b$ and $s_f < s_b$ respectively.

An example of a texture map for multiple isosurfaces is given in Figure 9, which shows the combination of the (colored) texture maps from Figure 6. The 'visibility ordering' is easy to understand: For $s_f < s_b$ we view along the gradient of the scalar field, thus isosurfaces for smaller isovalues occlude those for greater isovalues, and vice versa for $s_f > s_b$.

Assuming that all cells are rendered, the number of isosurfaces n in the texture map does not affect the rendering time. For opaque isosurfaces our method shares this feature with Westermann's algorithm for multiple isosurfaces [26], while ray-casting approaches depend at least logarithmically on n. For partially transparent isosurfaces our method does still not depend on n while the dependency of ray-casting approaches changes to n.

10 Mixing Isosurfaces with Projected Volumes

It was claimed that rendering mixtures of opaque polygons and volumetric data is straightforward, e.g. in [8]. This claim, however, does not apply to any cell projecting approach including the PT algorithm, since special attention has to be paid to partially occluded cells. In [32] Williams et al. suggest to slice each cell at user-specified isovalues. The time complexity of this method, however, depends linearly on the number of isosurfaces. As we noted in Section 9 the time complexity of our algorithm does not depend on the number of isosurfaces; therefore, we propose two alternative methods of mixing isosurfaces and projected tetrahedra, which are more appropriate in this context.

The algorithm presented in Section 8 allows us to smoothly include projected tetrahedra by rendering them after the corresponding back face triangle and before the front face triangle. This order ensures that the projected volume is completely occluded where the front face triangle is opaque, i.e. where the isosurface is in front of the volume at the front face, and that the volume is not affected where the front face is transparent, i.e. where the isosurface is behind the volume at the back face. Figure 7 illustrates this correlation: The relative thickness of the occluded part of the tetrahedron (white) corresponds to the weight of the front face (left gray scale).

An example employing this method is given in Figure 11, which mixes the isosurfaces of Figure 10 with the projected tetrahedra of Figure 4. More realistic examples are presented in Figures 14, 16, and 17. Figure 13 comprises the three 2D texture maps required to render the NASA Bluntfin data set (Figure 14).

Although our approach avoids discontinuities, it is not completely accurate with respect to correct ray integration. Therefore, we developed a more rigorous method. For opaque isosurfaces the ray integration in Equation (2), respectively Equation (1) if 3D texture mapping is employed, has to be stopped as soon as one of the isovalues is reached, i.e. for $s_l(t) = s_{\text{iso}}$ (see Figure 7). By rendering the isosurfaces for each triangle first (either in one pass for flat-shaded isosurfaces or two passes for smoothly shaded isosurfaces), followed by the projected volume with the modified 2D or 3D texture map, we are able to generate an accurate picture.

An example of a 2D texture map generated this way is shown in Figure 15 which corresponds to the middle texture in Figure 13. The isosurfaces manifest themselves in transparent vertical stripes which correspond to a scalar value s_f on the front face of a tetrahedron slightly greater than one of the isovalues. In Figure 12 this technique is used to visualize the opacity of the corresponding 3D texture map of Section 3.

Both methods presented in this section can be generalized to partially transparent isosurfaces.

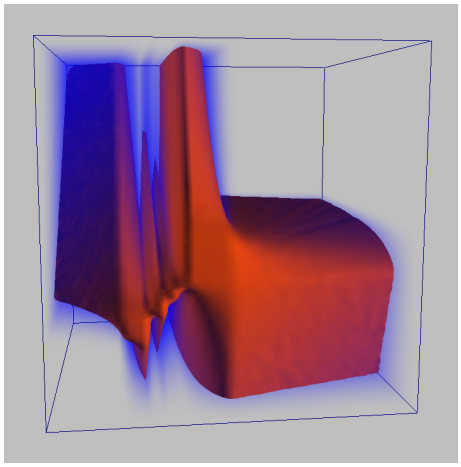

Figure 12: Visualization of the opacity of the 3D texture map that corresponds to the 2D texture map in Figure 15. The additional dimension parameterizes the length of the viewing ray within a tetrahedral cell. The isosurface represents opacity values of 0.25.

11 Results

With hardware-accelerated texture mapping the direct volume rendering methods presented in Sections 3 and 4 are essentially as fast as existing implementations of the PT algorithm. We emphasize that the rendering times for our methods is not affected by the particular transfer functions employed.

Our extensions of the PT algorithm are hard to compare with "non-PT" algorithms for direct volume rendering, e.g. approaches based on slicing, because the most time critical step of the PT algorithm is the sorting of the tetrahedra, which is not affected by the extensions presented in this paper.

The algorithms for the rendering of isosurfaces described in Sections 7 and 8 depend on the correct sorting and decomposition of the tetrahedral cells, while most of the algorithms mentioned in Section 5 do not require any sorting or decomposition of tetrahedra.

Moreover, we did not attempt to reduce the number of cells tested for intersections with the isosurface. Thus, most of the algorithms mentioned in Section 5 will usually be faster than our current implementation if used to render only a single isosurface. However, as our worst-case rendering time does not depend on the number of isosurfaces, our method will outrun most of the other algorithms if the number of isosurfaces is large enough (see also Table 1).

Moreover, our rendering algorithms greatly benefit from a combination with projected volume cells as described in Section 10 because the sorting and decomposition of tetrahedra can be reused in this scenario. Thus, the inclusion of isosurfaces in a visualization application based on the PT algorithm is almost for free. As the rendering in our methods includes extraction and triangulation of the isosurface, the rendering time (without sorting and decomposition of tetrahedra) should be compared to the sum of the extraction, triangulation, and rendering times of other algorithms. Additional efforts required by other algorithms for partially transparent isosurfaces and mixing with volume cells should also be considered in a fair comparison.

no. isosurfaces	no. cells	flat-shaded	smoothly shaded
1	14,729	0.09 sec.	0.22 sec.
2	25,361	0.20 sec.	0.41 sec.
10	25,361	0.20 sec.	0.41 sec.

Table 1: Rendering times (including 'extraction' and 'triangulation') for isosurfaces from the NASA bluntfin data set. The number of cells refers to the number of tetrahedra intersected by at least one isosurface. Timings for the sorting and decomposition of tetrahedra are not included as these steps are already done by the original PT algorithm without our extensions.

The rendering times in Table 1 were obtained on an Octane MXE with a MIPS R10K 250 MHz CPU. The isosurfaces were extracted from the NASA bluntfin data set, which was converted into 187,395 tetrahedra. An image with three isosurfaces is depicted in Figure 14. Clearly the rendering times for flat-shaded isosurface depend on the number of intersected tetrahedra (no double-counting) instead of the number of isosurfaces. Smoothly shaded isosurfaces require about twice as much time because the back and front faces have to be rendered separately.

For a single, smoothly shaded isosurface our rendering time is close to the 0.2 seconds reported by Westermann in [28]. The rendering performance is comparable to the results of Wittenbrink in [33].

12 Conclusion

We presented a hardware-accelerated but accurate way of rendering projected tetrahedra using 3D texture mapping. We expect this method to become more attrative as 3D texture mapping hardware becomes more widespread. As an alternative we derived a less accurate method using 2D texture mapping. Both methods allow us to employ arbitrary transfer functions, which is important for several volume visualization techniques, e.g. the extraction of unshaded isosurfaces with appropriate transfer functions.

A second extension to the PT alogrithm allows us to render multiple isosurfaces which may be partially transparent, colored, and/or smoothly shaded. The generation of these isosurfaces does not require the explicit calculation of new vertices and needs only a fraction of the computation time of the PT algorithm.

We have also described two methods to combine isosurfaces with projected volume cells. In this case our approach to rendering isosurfaces rather than extracting them is much faster than traditional techniques.

Furthermore, we expect the possibility of interactively rendering dozens or hundreds of isosurfaces in combination with projected volume cells to be useful for visualizing data sets which are less effectively visualized by direct volume rendering, e.g. the display of multiple time surfaces in flow visualization as presented by Westermann [26].

Finally, we would like to point out that our methods are especially suited for low-level graphics APIs; therefore, they would greatly benefit from an implementation of the PT algorithm in hardware. In particular the calculation of the vertices of the decomposed tetrahedra including the interpolation of gradients and scalar values at the new vertices could be tremendously accelerated. Supplemented with an API that also includes utility functions for the generation of suitable texture maps such a hardware implementation of the PT algorithm would allow programmers to quickly develop very fast and powerful volume visualization applications for structured and unstructured meshes.

13 Acknowledgments

The authors would like to thank R. Westermann for many fruitful discussions and B. Tomandl for providing the CT scan of the bonsai, which is visualized in Figure 17.

Appendix

This appendix demonstrates the extraction of unshaded isosurfaces with the technique presented in Section 3 by choosing an approriate transfer function ρ. As a side effect the 2D texture maps of Section 7 reveal themselves as special cases of the 3D texture map of Section 3.

In order to extract the isosurface for an isovalue s_{iso} we have to set $\rho(s) = 0$ for $s \neq s_{\text{iso}}$ and "$\rho(s_{\text{iso}}) = \infty$". Formally, we set $\rho(s) = C\delta(s - s_{\text{iso}})$ with a large constant C and Dirac's delta function $\delta(x)$ (see [25]); multiple isosurfaces correspond to a sum of delta functions. As $\kappa(s_{\text{iso}})$ is constant, we are only interested in the value of α as defined in Equation (1):

$$\begin{aligned}
1-\alpha &= \exp\left(-\int_0^l \rho(s_l(t))\mathrm{d}t\right) \\
&= \exp\left(-\int_0^l C\delta(s_f + \frac{t}{l}(s_b - s_f) - s_{\text{iso}})\mathrm{d}t\right) \\
&= \exp\left(-\int_0^l C\left|\frac{l}{s_b - s_f}\right|\delta\left(t - l\frac{s_{\text{iso}} - s_f}{s_b - s_f}\right)\mathrm{d}t\right) \\
&= \exp\left(-C'H\left(\frac{s_{\text{iso}} - s_f}{s_b - s_f}\right)H\left(\frac{s_{\text{iso}} - s_b}{s_f - s_b}\right)\right)
\end{aligned}$$

with $C' = C\left|\frac{l}{s_b - s_f}\right|$ and the Heaviside step function $H(x)$ (see [25]). Thus, for $C \to \infty$ we obtain

$$\alpha = H\left(\frac{s_{\text{iso}} - s_f}{s_b - s_f}\right)H\left(\frac{s_{\text{iso}} - s_b}{s_f - s_b}\right),$$

which is independent of l. The dependency on s_f and s_b is already visualized in the texture maps shown in Figure 6. Obviously, the 2D texture maps used in Section 7 are in fact special cases of the 3D texture map of Section 3. However, the derivation presented in Sections 6 and 7 appears to be more intuitive and comprehensible.

References

[1] C. L. Bajaj (ed.). *Data Visualization Techniques.* John Wiley & Sons, 1999.

[2] C. L. Bajaj, V. Pascucci, and D. R. Schikore. Fast Isocontouring For Improved Interactivity. In *1996 Symposium on Volume Visualization*, pages 39-46, 1996.

[3] P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno. Speeding Up Isosurface Extraction Using Interval Trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158-170, 1997.

[4] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno. Optimal Isosurface Extraction From Irregular Volume Data. In *1996 Symposium on Volume Visualization*, pages 31-38, 1996.

[5] J. Comba, J. T. Klosowski, N. Max, J. S. B. Mitchell, C. T. Silva, and P. L. Williams. Fast Polyhedral Cell Sorting For Interactive Rendering Of Unstructured Grids. In *Computer Graphics Forum (Proc. EUROGRAPHICS '99)*, 18(3):369-376, 1999.

[6] J. W. Durkin and J. F. Hughes. Nonpolygonal Isosurface Rendering For Large Volume Datasets. In *IEEE Visualization '94*, pages 293-300, 1994.

[7] K. Engel, R. Westermann, and T. Ertl. Isosurface Extraction Techniques For Web-Based Volume Visualization. In *IEEE Visualization '99*, pages 139-146, 1999.

[8] K. Kreeger and A. Kaufman. Mixing Translucent Polygons With Volumes. In *IEEE Visualization '99*, pages 191-198, 1999.

[9] Chyi-Cheng Lin and Yu-Tai Ching. An Efficient Volume-Rendering Algorithm With An Analytic Approach. *The Visual Computer*, 12(10):515-526, 1996.

[10] Y. Livnat and C. Hansen. View Dependent Isosurface Extraction. In *IEEE Visualization '98*, pages 175-180, 1998.

[11] Y. Livnat, H.-W. Shen, and C. R. Johnson. A Near Optimal Isosurface Extraction Algorithm Using Span Space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73-84, 1996.

[12] W. E. Lorensen and H. E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *ACM Computer Graphics (Proc. SIGGRAPH '87)*, 21(4):163-169, 1987.

[13] N. Max. Optical Models For Direct Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99-108, 1995.

[14] N. Max, B. Becker, and R. Crawfis. Flow Volumes For Interactive Vector Field Visualization. In *IEEE Visualization '93*, pages 19-24, 1993.

[15] C. Montani, R. Scateni, and R. Scopigno. Discretized Marching Cubes. In *IEEE Visualization '94*, pages 281-287, 1994.

[16] Kwang-Man Oh and Kyu Ho Park. A Type-Merging Algorithm For Extracting An Isosurface From Volumetric Data. *The Visual Computer*, 12(8):406-419, 1996.

[17] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan. Interactive Ray Tracing For Isosurface Rendering. In *IEEE Visualization '98*, pages 233-238, 1998.

[18] R. Shekhar, E. Fayyad, R. Yagel, and J. F. Cornhill. Octree-Based Decimation Of Marching Cubes Surfaces. In *IEEE Visualization '96*, pages 335-342, 1996.

[19] H.-W. Shen and C. R. Johnson. Sweeping Simplices: A Fast Iso-Surface Extraction Algorithm For Unstructured Grids. In *IEEE Visualization '95*, pages 143-150, 1995.

[20] H.-W. Shen, C. D. Hansen, Y. Livnat, and C. R. Johnson. Isosurfacing In Span Space With Utmost Efficiency (ISSUE). In *IEEE Visualization '96*, pages 287-294, 1996.

[21] P. Shirley and A. Tuchman. A Polygonal Approximation To Direct Scalar Volume Rendering. *ACM Computer Graphics (San Diego Workshop on Volume Visualization)*, 24(5):63-70, 1990.

[22] C. Silva and J. Mitchell. The Lazy Sweep Ray Casting Algorithm for Rendering Irregular Grids. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):142-156, 1997.

[23] M. Šrámek. Fast Surface Rendering From Raster Data By Voxel Traversal Using Chessboard Distance. In *IEEE Visualization '94*, pages 188-195, 1994.

[24] C. M. Stein, B. G. Becker, and N. L. Max. Sorting And Hardware Assisted Rendering For Volume Visualization. In *1994 Symposium on Volume Visualization*, pages 83-89, 1994.

[25] E. W. Weisstein. *Eric Weisstein's World Of Mathematics.* `http://mathworld.wolfram.com/`.

[26] R. Westermann, C. R. Johnson, and T. Ertl. A Level-Set Method For Flow Visualization. In *IEEE Visualization '00*, 2000.

[27] R. Westermann and T. Ertl. The VSBUFFER: Visibility Ordering Unstructured Volume Primitives By Polygon Drawing. In *IEEE Visualization '97*, pages 35-43, 1999.

[28] R. Westermann and T. Ertl. Efficiently Using Graphics Hardware In Volume Rendering Applications. *ACM Computer Graphics (Proc. SIGGRAPH '98)*, pages 169-177, 1998.

[29] J. Wilhelms and A. van Gelder. A Coherent Projection Approach For Direct Volume Rendering. *ACM Computer Graphics (Proc. SIGGRAPH '91)*, 25(4):275-284, 1991.

[30] J. Wilhelms and A. van Gelder. Octrees For Faster Isosurface Generation. *ACM Transactions on Graphics*, 11(3):201-227, 1992.

[31] P. L. Williams and N. Max. A Volume Density Optical Model. In *ACM Computer Graphics (1992 Workshop on Volume Visualization)*, pages 61-68, 1992.

[32] P. L. Williams, N. L. Max, and C. M. Stein. A High Accuracy Volume Renderer For Unstructured Data. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):37-54, 1998.

[33] C. M. Wittenbrink. CellFast: Interactive Unstructured Volume Rendering. *IEEE Visualization '99 Late Breaking Hot Topics*, pages 21-24, 1999.

[34] B. Wyvill, G. Wyvill, and C. McPheeters. Data Structures For Soft Objects. *The Visual Computer*, 2:227-234, 1986.

[35] R. Yagel, D. Reed, A. Law, P. Shih, and N. Shareef. Hardware Assisted Volume Rendering Of Unstructured Grids By Incremental Slicing. In *1996 Symposium on Volume Visualization*, pages 55-63, 1996.

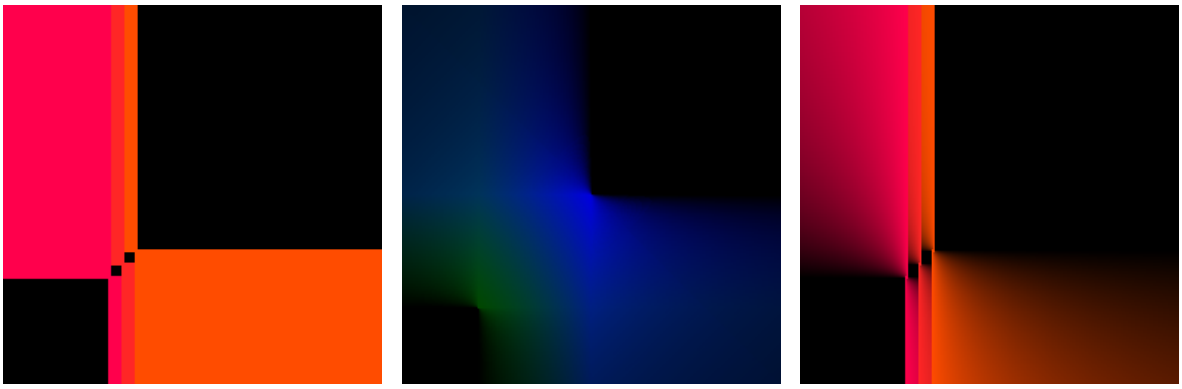

Figure 13: These 2D texture maps of dimensions 256×256 were used to render the Bluntfin data set depicted below. The left and right textures were employed to render the back and front face triangles, whereas the projected volume was generated by the middle texture. The texels on the diagonal of this texture represent the transfer functions. Black pixels in these images correspond to completely transparent texels.

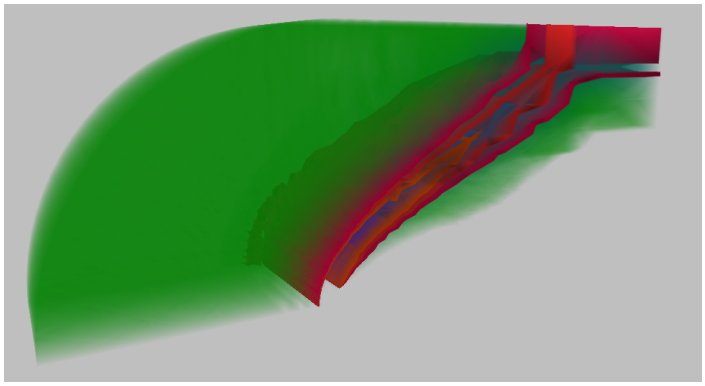

Figure 14: Visualization of the Bluntfin data set with three isosurfaces mixed with projected tetrahedra.

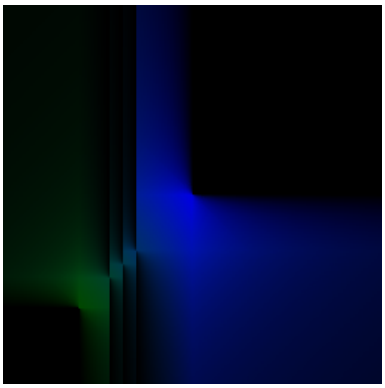

Figure 15: 2D texture map corresponding to the middle texture of Figure 13 but with the integration stopped at the isovalues.

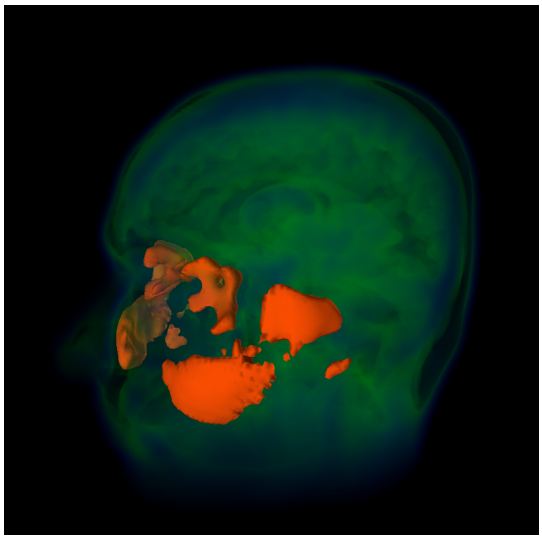

Figure 16: A visualization of an MRI head scan. The isosurface depicts soft tissue located in the cheeks and behind the eye balls.

Figure 17: A CT scan of a bonsai: Leaves are visualized by direct volume rendering, while the trunk and the branches are shown by the brown isosurface.