Convexification of Unstructured Grids

Stefan Roettger
Computer Graphics Group
University of Erlangen
roettger@cs.fau.de

Stefan Guthe
WSI/GRIS
University of Tübingen
sguthe@gris.uni-tuebingen.de

Andreas Schieber, Thomas Ertl
VIS Group
University of Stuttgart
ertl@informatik.uni-stuttgart.de

ABSTRACT
Unstructured tetrahedral grids are a common data representation of three-dimensional scalar fields. For convex unstructured meshes efficient rendering methods are known. For concave or cyclic meshes, however, a significant overhead is required to sort the grid cells in back to front order. In this paper we apply methods known from computational geometry to transform concave into convex grids. While this issue has been studied in theory it has not yet been applied to the specific area of unstructured volume rendering. This is mainly due to the complexity of the required geometrical operations. We demonstrate that the convexification of concave grids can be achieved by a combination of simple operations on triangle meshes. For convexified meshes the experimental results show that the performance penalty is only about 70% in comparison to approximately 300% for the fastest known concave sorting algorithm. In order to achieve high-quality visualizations we also adapt the pre-integrated lighting technique to cell projection.

KEY WORDS
Volume rendering, unstructured grids, cell projection, visibility sorting, convexification, pre-integration.

1 Motivation and Related Work

Considering the functional range of graphics libraries like OpenGL it is obvious that these libraries have a rich tool set for manipulation and rendering of triangle meshes. On the other hand, tetrahedral meshes, which are the three-dimensional counterpart of triangle meshes, have very little support. However, if a volumetric primitive were available right out of the box, direct volume visualization of unstructured grids would be straight forward. To achieve this goal, King et al. [7] have proposed a dedicated graphics hardware architecture, but unfortunately the architecture has not been realized yet. Efficient sweep plane algorithms for unstructured data are well established [14, 3], but currently the best suited method for dealing with volumetric rendering primitives is the projected tetrahedra (PT) algorithm of Shirley and Tuchman [13].

Recently, a competing hardware-accelerated approach has been presented by Weiler et al. [18]. They proposed a hardware-accelerated ray caster. While this is a promising approach, it currently has several drawbacks. First the available texture memory limits the maximum number of cells that can be ray cast. Secondly, ray casting is a screen space method as opposed to the PT algorithm, which is an object-space method. Finally, the performance of a hardware-accelerated ray caster may increase significantly with future graphics cards but right now its performance is in fact still lower than those of actual optimized implementations of the PT algorithm.

The PT algorithm requires the cells of the grid to be sorted in a back to front fashion. This procedure is known as visibility or depth sorting, which Wittenbrink [22] points out, is the main limiting factor of the PT algorithm. In the following we show how to remedy this main deficiency of the PT algorithm. By introducing an efficient visibility sorting algorithm we gain a significant performance advantage.

A reader familiar with the topic of visibility sorting may skip the introduction in the following section and may jump directly to Section 3.

2 The PT Algorithm

The cell projection algorithm of Shirley and Tuchman is commonly called the projected tetrahedra (PT) algorithm. It takes a scalar volume constructed from tetrahedra as input, and composes the projected tetrahedral cells in a back to front fashion. The footprint of each projected tetrahedron either consists of three or four triangles centered around the "thick vertex" of the tetrahedron as illustrated in Figure 1.

The volumetric primitive of a tetrahedron is transferred into a triangular representation that can

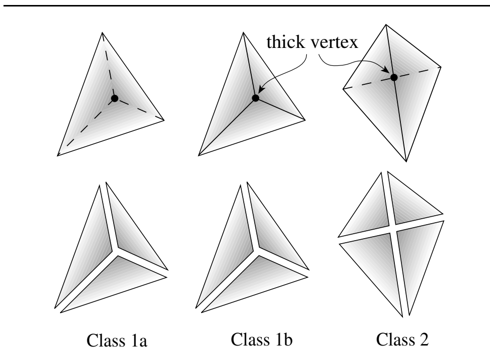

Figure 1: Classification of non-degenerated projected tetrahedra (top row) and the corresponding decomposition (bottom row) according to Shirley and Tuchman [13].

be rendered efficiently by the graphics hardware. This explains the popularity of the algorithm, since its performance directly relates to the number of cells in the data set. In contrast to ray casting and sweep plane methods the performance is almost independent of the size of the viewing window.

In recent years the original approach has been extended in numerous ways and is still under active research. The first improvement was presented by Stein et al. [16]. They used a more accurate exponential interpolation of opacities inside the tetrahedra instead of the linear approximation of the original approach.

In principle, the colors and opacities assigned to the tetrahedral decomposition correspond to the line integral along the intersection of each viewing ray with the tetrahedron. Using the volume density optical model of Williams [19, 10], the complexity of the line integral depends on the transfer functions of the optical model.

The line integral can be solved analytically for the special case of a linear transfer function. Later this was extended for piecewise linear transfer functions in the HIAC system [20]. For arbitrary transfer functions, however, a numerical integration of the transfer function is necessary. While the numerical integration cannot be performed on the fly, the line integral can be pre-computed and stored in a three-dimensional lookup table. This approach is called pre-integrated cell projection [12, 9].

Most recently, the increasing programmability of the graphics hardware has lead to further improvements: The large size of the three-dimensional pre-integration table prevented the use of a high-resolution transfer function. This drawback was resolved by a polynomial reconstruction of the pre-integration table in the pixel shader of modern graphics accelerators [5]. Using this approach only the coefficients for the Lagrangian polynomial approximation of the line integral need to be stored instead of the memory consuming pre-integration table itself.

Using the increasing capabilities of graphics accelerators the decomposition of the tetrahedra into triangles can also be performed in the vertex shader. This is called hardware-accelerated cell projection [17, 23]. Although this approach does not yet lead to a significant performance gain it is expected that graphics accelerators of the next generation will be much more efficient. Then the rendering speed primarily does not depend on the performance of the cell projection, but rather on the speed by which the CPU is able to feed the GPU over the AGP or PCI-Express bus.

Since the tetrahedra must be processed in a sorted order, that is usually in a back to front fashion, the overall system performance will be determined by the efficiency of the visibility sorting algorithm and the speed of the data transfer over the bus. Wittenbrink [22] points out that a read-write-read cycle of the tetrahedra is mandatory for visibility sorting and concludes that the memory access required for each tetrahedron is the main limiting factor of the PT algorithm.

Therefore, our prime goal in this paper is to devise a visibility sorting algorithm that keeps pace with the growing speed of the graphics accelerators. For this purpose we first give a brief survey of existing visibility sorting methods and discuss their advantages and their limitations.

3 Visibility Sorting

By definition an unstructured tetrahedral grid is a collection of tetrahedra, where it is assumed that the intersection of two tetrahedra is either empty or a common face. An unstructured grid is said to be convex, if the faces which are not shared between two tetrahedra form the convex hull of the data set. This definition of a convex grid excludes both the disconnectivity of the data set and the existence of cavities.

The task of visibility sorting is closely related to graph theory: for a convex grid the set of tetrahedral pairs (A,B) with a common face define the edges of a directed graph. The direction of each edge determines the 'behind' relationship, that is whether or not cell A occludes cell B. The direction of the

edge can be determined quickly by computing the dot product of the viewing direction with the normal of the common face.

The directed graph imposes an ordering on the set of tetrahedra which is said to be the visibility or depth ordering. If the directed graph is acyclic (DAG) then the ordering is total but it need not be unique.

The well known MPVO algorithm of Williams et al. [21] computes a visibility ordering for a convex grid by traversing the DAG: whenever the algorithm encounters a cell which does not occlude unvisited neighbours it outputs the cell, otherwise it traverses the graph in the direction of the edges. In this fashion a depth sorted list of tetrahedra is constructed for each specific point of view.

The cost of the MPVO algorithm is equal to the cost of a depth-first or breadth-first graph traversal, so that the run time complexity is linear in terms of the number of cells.

4 Treatment of Cycles

It has to be mentioned that in the majority of cases the sorting graph will be a DAG. However, a cycle may occur quite easily. For example a gear with slanted teeth may have a cycle: when looking along the axis of the gear the teeth may occlude each other in a cyclic way (see also Williams et al. [21]).

In the case of a cyclic graph any graph sorting algorithm will fail, but we still have two options to proceed. The first option is to cut the cycle apart by selecting an appropriate cutting plane. This is a difficult task even for simple cycles. A better option is to use the MPVO-C algorithm of Kraus et al. [8].

This algorithm can handle arbitrary meshes including cyclic meshes without the need of sorting, but has quadratic run time in contrast to the linear run time of the MPVO algorithm. The MPVO-C algorithm is the three-dimensional analogue to Snyder and Lengyel's algorithm of rendering cyclic triangles.

Since the run time is quadratic we need to use MPVO-C as a fall-back solution. In the case that the visibility sorting algorithm has detected a cycle we use MPVO-C to render the small group of cells that cause the cycle. In combination with MPVO the detection of a cycle and the identification of its cells is straight forward using standard graph theory and does not increase the run time complexity of the MPVO algorithm (also compare [8]).

If a cycle is detected on the fly, the MPVO-C algorithm is triggered for the set of cells that form the cycle. Since these usually only make up for a tiny fraction of the entire data set, the worst case run time complexity is quadratic but the average run time complexity is still linear.

5 Visibility Sorting of Concave Grids

Concave or disconnected grids cannot be handled correctly by the MPVO algorithm since the behind relations are not defined for the boundary faces. This fact is illustrated in Figure 2 which shows a gear rendered with correct ordering and a difference image showing the artifacts produced by the MPVO algorithm. A straight forward solution for this problem is simply to add the missing relations between the boundary faces to the DAG.

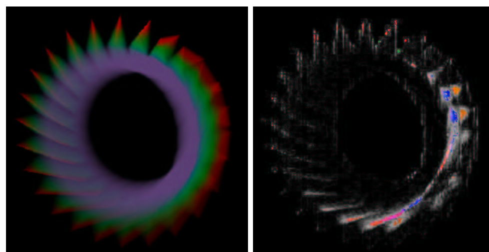

Figure 2: Artifacts produced by incorrect visibility ordering using the MPVO algorithm (correct image on the left and difference image on the right).

Several algorithms are known which compute the missing relations. The MPVO-NC algorithm is an extension presented by Williams in his MPVO paper. It uses a simple heuristic for the determination of the additional face relations. This heuristic is easy to compute but in many cases is not able to determine the correct set of relations. The X-MPVO algorithm of Silva et al. [15] utilizes a sweep plane parallel to the viewing plane to process the boundary faces in depth sorted order and thus is able to find the correct relations. Because of the expensive sweep plane calculations this method was later improved by Comba et al. [1] who introduced a BSP tree for the efficient computation of the boundary relations. This method is called BSP-XMPVO.

Another fast sorting algorithm for concave meshes is the so called tangential distance or power sort [6] which requires the mesh to fulfill the Delaunay criterion. Although the asymptotic run time complexity is $O(n \log n)$ the average run time complexity is $O(n)$. For meshes which are not Delaunay meshes, a new mesh can be constructed that fulfills the Delaunay criterion. In three dimensions, however, this often requires the addition of new vertices

to the data set, thus the mesh topology cannot be preserved in general.

Note that all these sorting algorithms for concave grids assume the mesh to be acyclic.

6 Convexification of Tetrahedral Grids

An analysis of current visibility sorting algorithms shows that BSP-XMPVO on the one hand produces correct results for all types of meshes excluding cyclic meshes. On the other hand it is significantly slower than MPVO, which fails to sort concave meshes. Cyclic meshes can be handled with the MPVO-C method. An optimal graph sorting algorithm would combine the advantages of these methods.

Such an optimal sorting algorithm has already been proposed by Peter Williams [21]. He suggested to add imaginary cells to a concave data set in order to transform it into a convex mesh, that is to fill out the cavities of a data set with auxiliary cells so that the extended mesh could be handled by using the plain MPVO algorithm.

He also noted that "The implementation of the preprocessing methods [...] for converting a nonconvex mesh into a convex mesh could take a very significant amount of time; they are by no means trivial. [...] Therefore, the MPVO algorithm for nonconvex meshes, which has been found to be easy to implement, may fill an immediate need despite its shortcomings".

Convexification algorithms are known from computational geometry [4] and are well studied in theory. But to our knowledge a convexification algorithm that is easy to implement has not yet been utilized for the purpose of depth sorting an unstructured grid.

In this paper we implement such a convexification algorithm. The key idea is not to try to add imaginary tetrahedra to the concave data set but rather to subsequently break up the cavities into sets of convex polyhedra and treat these polyhedra as imaginary cells. For that purpose we combine several standard algorithms dealing with triangle meshes in a new and unique way. If the original concave mesh also contains cycles we can use the MPVO-C algorithm to take care of the cycles as outlined in Section 4.

Since the run time of the MPVO algorithm is linear in terms of the number of cells, the performance decreases with the number of auxiliary cells. In practice the number of auxiliary polyhedra is small in comparison to the total number of cells so that the sorting performance is still linear on the average. This is analyzed in more detail in Section 7.1. In the following we give an algorithmic description of our proposed tetrahedral convexification algorithm.

6.1 Basic Algorithm

Let S be a set of triangles that form the closed boundary surface of a volume. We assume that the normals of such a triangle set point outwards. Then the volume is said to be concave if the opening angle at the common edge of two triangles is less than $180°$. The volume is said to be connected if all triangles can be reached by traveling along the edges of the boundary. With these definitions the convexification of an unstructured tetrahedral grid can be described with the first step as follows:

1) S_0 initially contains the boundary faces of the unstructured grid
2) flip the normals of all triangles in S_0
3) add the triangles of the convex hull of the grid to S_0 with all normals pointing outwards
4) remove all triangles from S_0 that appear twice

Now S_0 contains the boundary description of the smallest exterior volume that needs to be added in order to generate a convex mesh (compare left side of Figure 3). This volume can be concave or even disconnected. If S_0 is empty, the tetrahedral mesh is already convex and can be fed directly into the MPVO/MPVO-C sorting algorithm. Alternatively, we can replace Steps 3) and 4) by just adding the faces of a bounding box (see right illustration in Figure 3).

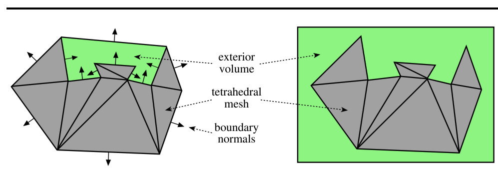

Figure 3: Determining the exterior volume (depicted in light green). **Left**: minimal volume using convex hull. **Right**: easy setup with bounding box.

As mentioned above, we do not try to fill the exterior volume with tetrahedra but we rather break it up into a set of n convex polyhedra $S_i, i = 1..n$. This is achieved by cutting away one concavity after another by using a combination of simple operations on triangle meshes. For each detected concav-

ity the exterior volume is split into two sub-volumes similar to binary space partition. This operation is repeated until all sub-volumes have been split into convex polyhedra:

$n = 1;\ S_1 = S_0$
repeat
  if S_i is disconnected for any $i = 1..n$
    1) separate S_i into a connected component S_i' and the remaining sub-volume S_i''
  else if S_i is concave
    2) choose triangle $T \in S_i$ so that the plane P through T cuts S_i into two non-empty sub-volumes S_i' and S_i''
    3) move each triangle $T \in S_i$ to its corresponding sub-volume
    4) triangulate the intersection of the cutting plane P with S_i and add the resulting triangles to both S_i' and S_i''
  endif
  $n = n + 1;\ S_i = S_i';\ S_n = S_i''$
until S_i is convex and connected for $i = 1..n$

In Step 3) the triangles that intersect with the cutting plane P have to be split and the resulting sub-triangles have to be moved into the corresponding sub-volume S_i' or S_i''. Note that the tetrahedral mesh is not split at all. Only a boundary face may be split so that the DAG has multiple dependencies for this face (also compare bottom right of Figure 5). The intersection of the cutting plane P with a sub-volume S_i is a polygon which may be concave or even disconnected. This polygon has to be triangulated and added to both subsets S_i' and S_i'', since otherwise the sub-volumes would not be closed. Triangles which lie in the cutting plane are a special case and must be added to only one sub-volume. To avoid numerical instabilities we move vertices with a very small distance to P onto the cutting plane, so that impossible cutting configurations cannot occur due to floating point errors.

The described convexification algorithm does not require complex volumetric operations but rather is a combination of well known algorithms working on polygons. In this way the goal of filling an arbitrarily complex volume with tetrahedra is broken down to a number of well analyzed operations on triangle meshes. To our knowledge this specific approach is new in the research area of visibility sorting (also compare recent research on convexification [2]). In the worst case one iteration of the cutting algorithm is needed for each face of the boundary. For each cut the triangulation in Step 4) is the most expensive operation with $O(b \log b)$ worst case run time and b being the number of boundary faces in the sub-volume. Therefore, the worst case run time for the preprocessing of the tetrahedral mesh is $O(b^2 \log b)$. In practice, however, each cut approximately halves the number of triangles in a sub-volume. Thus, the average preprocessing time is $O(b \log^2 b)$.

6.2 Cutting Plane Selection

The run time of the MPVO algorithm for a convexified mesh directly relates to the number of auxiliary cells. So our goal is to keep the number of generated auxiliary cells as low as possible. This is achieved by an appropriate selection of the cutting plane.

A first approach would be to select the cutting plane which bisects each sub-volume. This is a similar strategy to the construction of BSP trees in computer games. Here the BSP performance relies on equal sized leave nodes, for which the bisection strategy works well. In our case, however, the size of the nodes is not relevant, since the cells are imaginary and thus need not be rendered. Instead we want to minimize the total number of auxiliary cells.

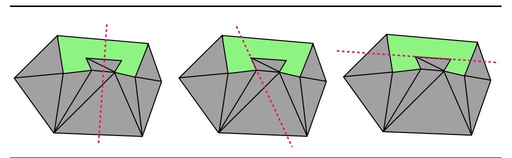

Figure 4: Selection of cutting plane. **Left**: bad BSP strategy (no concavity cut away). **Middle**: elimination of one concavity. **Right**: elimination of two concavities (but total number of auxiliary cells is the same, since the bottom sub-volume is disconnected).

In principle, the cutting plane P should be chosen so that the corresponding triangle T has at least one neighbour with an opening angle less than $180°$. This ensures that at least one concavity is cut away from the sub-volume. Since there are usually many triangles that fulfill this condition (see middle and right case in Figure 4), we propose the following selection criterion.

The criterion is based on the fundamental observation that the number of generated auxiliary cells at each cutting step depends on the number of intersections of the cutting plane with the sub-volume. The more intersections the cutting plane has with

the boundary of the sub-volume, the more auxiliary cells are generated.

As an example, Figure 5 shows the convexification of a simple two-dimensional object. After checking possible cuts we could vote for the horizontal cut on the top right of Figure 5 which has 4 intersections with the boundary. But then the small cell depicted in bright red would be generated. This cell is redundant, since it does not eliminate any concavity. So we better vote for the vertical cut which has only 2 intersections. The resulting left sub-volume is convexified easily by one additional cut. The right sub-volume requires another two cuts. We can choose either of three possible cuts, since all have the same intersection count. Finally we have found a convexification consisting of five auxiliary cells (bottom right of Figure 5).

In contrast to the two-dimensional example, the mesh topology can be more complex in three dimensions. But cuts with few intersections with a sub-volume also tend to produce few auxiliary cells. This is not a strict property, since a seemingly bad first cut may in some cases enable a better second cut. The determination of the best possible cut is a very expensive optimization problem. Therefore it is infeasible in practice. The described selection heuristic, however, works well in practice as shown in Section 7.

In order to speed up the determination of a good cut we just randomly select a small fixed number of candidates and choose the best cut of this group. A very similar strategy is used by the BSP-XMPVO sorting algorithm for the construction of its BSP tree.

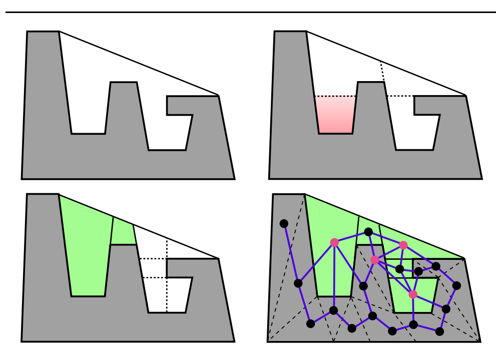

Figure 5: 2D convexification example with resulting sorting graph. The light red balls depict cells with multiple dependencies for a face. The generated auxiliary (imaginary) cells have been marked light green.

Using the described convexification approach, Figure 6 shows the convexified Blunt Fin data set (S_0 was derived from a bounding box). For this data set a single auxiliary cell is generated. Only this imaginary cell needs to be added to the original data set to build a convex mesh. Similarly, the convexification of the Oxygen Post, Tapered Cylinder and Heat Sink data sets only requires the addition of one auxiliary cell. Many other data sets encountered in practice cannot be handled as easy as this, but the Blunt Fin example illustrates that automatic convexification is straight forward in many cases. Due to the single auxiliary cell the performance of the MPVO for the convexified Blunt Fin is virtually the same as the performance of the original MPVO-NC algorithm.

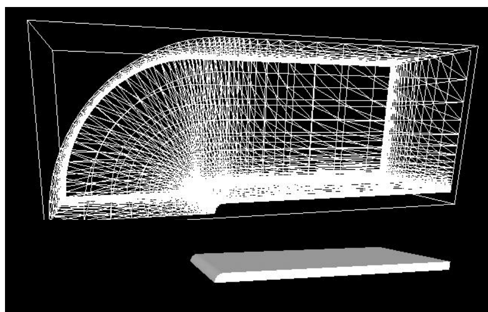

Figure 6: In order to convexify the Blunt Fin data set only one auxiliary cell needs to be added (depicted below the wire frame model).

Figure 7 illustrates the process of splitting the exterior volume into convex polyhedra. The corners of the surrounding bounding box are cut away until finally a large imaginary cylinder and imaginary convex polyhedra between the teeth remain. Note that the cylinder needs to be split because the quadrilaterals on its surface are nonplanar and thus have concavities.

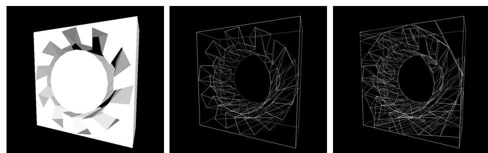

Figure 7: Convexification example of a gear: exterior volume (left), wire frame of unprocessed exterior volume (middle), and exterior volume split into convex polyhedra.

7 Implementation

Our implementation of the MPVO algorithm has a performance of about 2,1 million tetrahedra per second on an an Intel Pentium 4 processor with 3,0 GHz.

At these clock speeds the sorting algorithm is mainly memory bound (as pointed out by Wittenbrink). To reduce the traffic on the memory bus we use an indexed data structure. Additionally, we also utilize an indexed normal data structure which leads to a performance increase of about 30-60% depending on the regularity of the data set (e.g. Blunt Fin 59% and SPX 31%). Together with an ISSE enhanced cell projection algorithm we achieve an overall rendering performance of about 1,25 million tetrahedra per second on an NVIDIA GeForce FX 5800 graphics accelerator.

The Blunt Fin data set in Figure 6 with 224874 tetrahedra renders at approximately 5.5 frames per second. The performance for the SPX data set is given in Table 8. The first row shows the results of using the plain MPVO algorithm which is fastest but does not render the SPX data set correctly. With an increasing number of tested cutting planes the total number of auxiliary cells decreases significantly and so does sorting time.

MPVO	grid cells	sorting	total
	12936	4.1 ms	10.8 ms
# cuts	aux. cells	sorting	total
1	1562	10.2 ms	
5	1115	7.8 ms	
10	1000	7.4 ms	
20	836	7.0 ms	14.4 ms

Figure 8: Sorting and rendering times for the SPX data set with 12936 tetrahedra (Coolant flow simulation in the Super PhoeniX reactor).

Taking the number of generated auxiliary cells into account, the total number of cells is increased by only 13% but sorting time increases by 70% and total rendering time by 33%. The reason for this over-proportional increase is the high irregularity and the comparably large number of faces of auxiliary cells (7 faces per auxiliary cell on the average for the SPX).

The latest hardware-accelerated cell projection algorithms [17, 23] achieve approximately 550,000 tetrahedra per second on a NVIDIA GeForce4 excluding times for sorting. This corresponds to about 2 frames per second for the Blunt Fin data set using a purely emissive optical model.

Our performance is on par with the latest hardware-accelerated ray caster of Weiler et al [18]. However, our performance is almost independent of the size of the viewing window. Additionally, our proposed convexification algorithm is not limited to the application area of direct volume rendering but could also be used for a variety of tasks in finite element simulations, for example.

7.1 Performance Analysis

The BSP-XMPVO algorithm is the fastest known sorting algorithm for concave meshes known today. Our convexification method has a strong affinity to BSP-XMPVO. While the latter uses a BSP tree on the boundary faces to calculate the boundary edges of the sorting graph, we use a cutting strategy to add auxiliary cells at the boundary.

The main difference between the two methods is not the different partitioning scheme. In fact, the BSP tree is just another means of applying cutting planes to sub-volumes. The main difference, however, is that the calculation of the boundary dependencies is performed during convexification, which is a preprocessing step. Therefore we do not need to traverse a BSP tree whenever we encounter a boundary face. In other words, the time consuming BSP traversal is performed implicitly by building the adjacency graph from the convexified grid. Since the BSP traversal accounts for a large fraction of the sorting time, we achieve a considerable speedup in comparison to BSP-XMPVO.

The penalty of BSP-XMPVO over MPVO(NC) is in the range of 320-530% as stated in [1]. The penalty of the convexification however is below 70% in our tests. In the past the MPVO algorithm has proven to be the fastest solution for convex meshes. Our experimental results show that the MPVO algorithm is also the best option for convexified concave meshes.

Due to the similarity of the BSP tree construction and the application of the cutting planes we can deduce the asymptotic run time of our algorithm as follows: the run time of BSP-XMPVO is $O(bp+n)$, where b is the number of boundary faces, n is the number of grid cells, and p is the number of faces that are cut by more than one face of the BSP. The fact that b is usually below 5% of the total number of cells, and p is usually much smaller than b, makes BSP-XMPVO linear in terms of n.

The same argument holds for convexification: the asymptotic run time of the MPVO for the convexified mesh is $O(a+n)$ with a being the number of auxiliary cells and n being the number of grid

cells. The number of auxiliary cells is proportional to the number of boundary faces b plus the number of auxiliary cells that are cut by another cutting plane. Since the latter only account for a tiny fraction of all cells, the run time in practice is essentially linear in terms of n.

Another advantage of the presented convexification algorithm is the fact that cyclic meshes can be handled easily. The implicit storage of the dependencies in a BSP-Tree makes it hard to detect and manipulate cycles. A convexified sorting graph, however, can be checked easily, since the boundary dependencies are stored explicitely by means of the auxiliary cells (compare also Kraus et al. [8]).

8 Tetrahedral Lighting

Once the tetrehadra are sorted they can be cell-projected and rendered. In this section we describe how to improve the visual appearance of the tetrehadra. As stated in the last section, the rendering performance mainly depends on the sorting time and the vertex performance. Therefore, there is no performance penalty in using quiet long fragment programs. We use this opportunity to introduce high-quality tetrahedral lighting. In general, high-quality rendering of unlit tetrahedra has been introduced by Guthe et al. [5], but high-quality lighting has not yet been applied in the area of unstructured volume rendering.

We adapt the approach of Meissner et al. [11] who introduced ambient, diffuse and specular lighting coefficients for direct pre-integrated volume rendering of regular meshes. In order to implement efficient lighting for unstructured meshes, we have to take a closer look at the underlying scenario as illustrated in Figure 9.

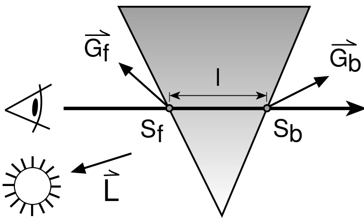

Figure 9: Each viewing ray within a tetrahedron is defined by two sample points and two gradients. The scalar values and gradients are interpolated linearly.

Since the scalar values and the gradients are interpolated linearly within each tetrahedron, they are also linear along each viewing ray. To allow for efficient pre-integration (see [12] for an introduction to pre-integrated cell-projection), we assume that the intensity of the light varies linearly. Therefore the light intensity I at position x of the ray segment can be calculated as

$$\begin{aligned} I(x) &= (x\vec{G}_f + (1-x)\vec{G}_b)\cdot\vec{L} \\ &\approx x(\vec{G}_f\cdot\vec{L}) + (1-x)(\vec{G}_b\cdot\vec{L}). \end{aligned}$$

$I(x)$ can now be split into $I_f(x) = x(\vec{G}_f\cdot\vec{L}) = xI_f$ and $I_b(x) = (1-x)(\vec{G}_b\cdot\vec{L}) = (1-x)I_b$. The ray integral $C(x)$ and its approximation $C'(x)$ for the diffuse part of the emission are then given as

$$\begin{gathered} C(x) = \\ = \int_0^x e^{-\int_0^t \rho(S(u))\mathrm{d}u}\kappa(S(t))\rho(S(t))I(t)\mathrm{d}t \\ \approx \int_0^x e^{-\int_0^t \rho(S(u))\mathrm{d}u}\kappa(S(t))\rho(S(t))(I_f(t)+I_b(t))\mathrm{d}t \\ = C'(x). \end{gathered}$$

Using the linearity of the integral, the calculation can be split into two integrals that do not depend on the light intensity along the ray:

$$\begin{gathered} C'(x) = \\ = \int_0^x e^{-\int_0^t \rho(S(u))\mathrm{d}u}\kappa(S(t))\rho(S(t))I_f(t)\mathrm{d}t + \\ \int_0^x e^{-\int_0^t \rho(S(u))\mathrm{d}u}\kappa(S(t))\rho(S(t))I_b(t)\mathrm{d}t \\ = I_f\int_0^x e^{-\int_0^t \rho(S(u))\mathrm{d}u}t\kappa(S(t))\rho(S(t))\mathrm{d}t + \\ I_b\int_0^x e^{-\int_0^t \rho(S(u))\mathrm{d}u}(1-t)\kappa(S(t))\rho(S(t))\mathrm{d}t. \end{gathered}$$

With $\kappa_f(S(t)) = t\kappa(S(t))$ and $\kappa_b(S(t)) = (1-t)\kappa(S(t))$, a pre-integration table is able not only to represent the ambient, but also the diffuse part of the emission along the ray segments. We need three separate tables, i.e. an ambient and two diffuse ones for κ_f and κ_b, to reconstruct the color along each ray segment. The lighting of the tetrahedra is equivalent to using Gouraud shading on a per ray segment basis. To change the weighting of ambient and diffuse lighting, we use different κ for each of these lighting terms. Figure 10 shows the improved visual perception of isosurfaces for the Bucky Ball data set.

Meissner et al. [11] also showed how to implement a specular highlight for non-iso-surface-like transfer functions by integrating a weighting coefficient w for the front and back gradients. In addition their approximation also needs a weighting coefficient s for the intensity of the highlight. If we adapt the same approach for tetrahedra, we end up with a

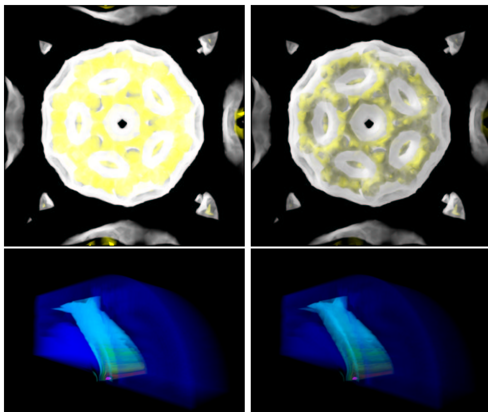

Figure 10: Bucky Ball and convexified Blunt Fin data set with per-ray lighting. With ambient only (left) and diffuse lighting (right).

pre-integration table that contains 12 entries, 3 for the integrated ambient color, 6 for the diffuse color, 1 for the integrated opacity and 2 for the specular highlight. These 12 values can easily be stored in 3 RGBA textures.

Although this approach is almost a straight forward adaption of [11] we propose the following necessary modifications for unstructured grids: We need to account for the fact that tetrahedra have varying ray segment length l as opposed to texture based pre-integrated volume rendering where l is assumed to be constant. Since we do not want to recompute the three-dimensional pre-integration table for different maximum ray segment lengths l_{max}, we use $l' = 1 - 2^{-l}$ rather than l as a reparametrization of the third texture coordinate. In fact, this reparametrization also improves accuracy. It is a known fact that the emission becomes almost constant for large l. Therefore we can represent this parameter range with fewer sample points and use more sample points for small values of l. To further improve the accuracy of the integrated opacity, we do not actually store the opacity but rather the integrated optical density and reconstruct the opacity in the pixel shader. Then the shader has to carry out the following calculations:

1) Compute the perspectively interpolated scalar values (S_f and S_b), the gradients ($\vec{G}_f$ and $\vec{G}_b$) and the ray segment length l.
2) Calculate $l' = 1 - 2^{-l}$.
3) Lookup all three pre-integration tables at position (S_f, S_b, l').
4) Perform diffuse lighting of front and back gradient ($I_f = \vec{G}_f \cdot \vec{L}$ and $I_b = \vec{G}_b \cdot \vec{L}$).
5) Calculate representative gradient for specular highlight $\vec{G_{spec}} = w\vec{G}_f + (1-w)\vec{G}_b$.
6) Calculate highlight $I_{spec} = s(\vec{G_{spec}} \cdot \vec{L})^p$.
7) Reconstruct opacity from integrated optical density.
8) Add all colors and blend with frame buffer.

With this approach the specular highlight is reproduced correctly over multiple opaque iso-surfaces within a single tetrahedron and is approximated efficiently for semi-transparent transfer functions as seen in Figure 11. In order to further increase the accuracy of both diffuse lighting and the specular highlight, additional samples on each ray segment can be used. The resulting image quickly converges to the correct solution, so four samples per ray segment are usually sufficient. A larger number of samples significantly lowers the fill rate.

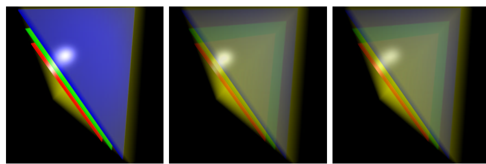

Figure 11: Specular highlight on opaque isosurfaces within a single tetrahedron (left), approximation for a semi-transparent setting (middle) and “correct” solution with four samples per ray segment (right).

9 Conclusions

We have described an efficient convexification algorithm for the purpose of visibility sorting arbitrary unstructured volumetric grids. We proposed an easy to implement convexification algorithm by reducing the complex geometrical operations to a sequence of operations on triangle meshes. We used this method to transfer concave (and not necessarily acyclic grids) into convex meshes which can be sorted efficiently with the plain MPVO/MPVO-C algorithm. As a result, our method is two to three times faster than the fastest known sorting technique, the BSP-XMPVO algorithm. Our approach is on par with the latest hardware-accelerated ray casting algorithm but our performance is independent of the window size. Finally, we also addressed

the high-quality reconstruction of the ray integral using the pre-integrated lighting technique.

10 Acknowledgements

We would like to thank Martin Kraus and Günther Greiner for interesting discussions on the topic of unstructured volume rendering. We would also like to thank Marc Stamminger for valuable suggestions. Last but not least, we gratefully acknowledge Hanspeter Pfister for requesting tetrahedral lighting at Graphics Hardware '02. Part of this work was sponsored by the NVIDIA fellowship grant.

11 Additional Material

For the convenience of the reader, we provide a movie which illustrates the convexification process for two interleaving gears. In each frame the cutting operation with the lowest intersection count is performed. Imaginary cells outside the convex hull of the data set are discarded.

References

[1] J. Comba, J. T. Klosowski, N. L. Max, J. S. B. Mitchell, C. T. Silva, and P. L. Williams. Fast Polyhedral Cell Sorting for Interactive Rendering of Unstructured Grids. *Computer Graphics Forum (Proc. Eurographics '99)*, 18(3):369–376, 1999.

[2] J. Comba, J. Mitchell, and C. Silva. On the Convexification of Unstructured Grids From A Scientific Visualization Perspective. Technical Report UUSCI-2004-004, Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, July 6, 2004.

[3] R. Farias, J. S. B. Mitchell, and C. T. Silva. Zsweep: An efficient and exact projection algorithm for unstructured volume rendering. In *IEEE Symposium on Volume Visualization '00*, pages 91–99, 2000.

[4] J. E. Goodman and J. O'Rourke. *Handbook of Discrete and Computational Geometry*. CRC Press LLC, ISBN 0-8493-8524-5, 1997.

[5] S. Guthe, S. Roettger, A. Schieber, W. Strasser, and Th. Ertl. High-Quality Unstructured Volume Rendering on the PC Platform. In *Proc. EG/SIGGRAPH Graphics Hardware Workshop '02*, pages 119–125, 2002.

[6] M. Karasick, D. Lieber, L. Nackman, and V. Rajan. Visualization of Three-Dimensional Delaunay Meshes. *Algorithmica*, 19(1–2):114–128, 1997.

[7] D. King and C. M. Wittenbrink. An Architecture for Interactive Tetrahedral Volume Rendering. In *Proc. IEEE/EG Workshop on Volume Graphics '01*, pages 163–180. Springer Verlag, Wien, 2001.

[8] M. Kraus and Th. Ertl. Cell-Projection of Cyclic Meshes. In *Proc. IEEE Visualization '01*, pages 215–222, 2001.

[9] N. L. Max, P. Hanrahan, and R. Crawfis. Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions. *Computer Graphics (San Diego Workshop on Volume Visualization)*, 24(5):27–33, 1990.

[10] Nelson L. Max. Optical Models for Direct Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

[11] M. Meissner, S. Guthe, and W. Strasser. Interactive Lighting Models and Pre-Integration for Volume Rendering on PC Graphics Accelerators. In *Proc. Graphics Interface '02*, pages 209–218, 2002.

[12] S. Roettger, M. Kraus, and Th. Ertl. Hardware-Accelerated Volume and Isosurface Rendering Based on Cell-Projection. In *Proc. Visualization '00*, pages 109–116. IEEE, 2000.

[13] 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.

[14] C. T. Silva and J. S. B. Mitchell. The Lazy Sweep Ray Casting Algorithm for Rendering Irregular Grids. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):142–157, 1997.

[15] C. T. Silva, J. S.B. Mitchell, and P. L. Williams. An exact interactive time visibility ordering algorithm for polyhedral cell complexes. In *IEEE Symposium on Volume Visualization '98*, pages 87–94, 1998.

[16] C. M. Stein, B. G. Becker, and N. L. Max. Sorting and Hardware Assisted Rendering for Volume Visualization. In *Symposium on Volume Visualization '94*, pages 83–89. IEEE, 1994.

[17] M. Weiler and Th. Ertl. Hardware-Based View-Independent Cell Projection. In *Proc. IEEE Symposium on Volume Visualization '02*, pages 13–22, 2002.

[18] M. Weiler, M. Kraus, M. Merz, and Th. Ertl. Hardware-Based Ray Casting for Tetrahedral Meshes. In *Proc. Visualization '03*, pages 333–340. IEEE, 2003.

[19] P. L. Williams and N. L. Max. A Volume Density Optical Model. In *Computer Graphics (Workshop on Volume Visualization '92)*, pages 61–68. ACM, 1992.

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

[21] Peter L. Williams. Visibility Ordering Meshed Polyhedra. *ACM Transactions on Graphics*, 11(2):103–126, 1992.

[22] Craig M. Wittenbrink. CellFast: Interactive Unstructured Volume Rendering. In *IEEE Visualization '99 Late Breaking Hot Topics*, pages 21–24, 1999.

[23] B. Wylie, K. Moreland, L. A. Fisk, and P. Crossno. Tetrahedral Projection using Vertex Shaders. In *Proc. IEEE Symposium on Volume Visualization '02*, pages 7–12. ACM Press, 2002.