LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-JRNL-222324

Hexahedron Projection for Curvilinear Grids

N. Max

June 21, 2006

Journal of Graphics Tools

Hexahedron Projection for Curvilinear Grids

Nelson Max

Lawrence Livermore National Laboratory

ABSTRACT:

This paper presents a method of dividing into triangle fans the most common projections of hexahedra from curvilinear meshes, so that they can be volume rendered in hardware, with a fragment program for 32-bit floating point compositing.

CR Categories and Subject Descriptors: 1.3 Computer Graphics

Additional Keywords: volume rendering, polyhedron projection

1. INTRODUCTION

The polyhedron projection method for volume rendering divides the projection of each volume cell into polygons which lie inside the projections of a single front-facing and a single back-facing cell face, as shown for several hexahedra in figure 1. In an orthogonal view, the thickness, that is, the length of the viewing ray inside the cell, varies linearly across such a projection polygon if the faces are planar, as shown in a 2D analog in figure 2. The thickness can thus be linearly interpolated by the hardware, in preparation for shading to achieve back-to-front color-opacity compositing. This hardware method was pioneered by Shirley and Tuchman [1] for tetrahedra, and a corresponding method for parallel projection of rectilinear grids of identically shaped cells was described by Wilhelms and Van Gelder [2]. To form these polygonal regions in the general case, the image plane must be subdivided by the projections of all the edges of the volume cell. This is a computational geometry problem. Wilhelms and Van Gelder [2] described a line sweep method for constructing this subdivision, and Max, Williams, and Silva [3] described an incremental method which inserted the edge projections into the subdivision one at a time. Such methods are difficult to implement robustly, since they require topological consistency among multiple tests for questions like "does point *P* lie to the left, on, or to the right of line *L*?" The finite precision of floating point arithmetic can cause inconsistent results from such tests.

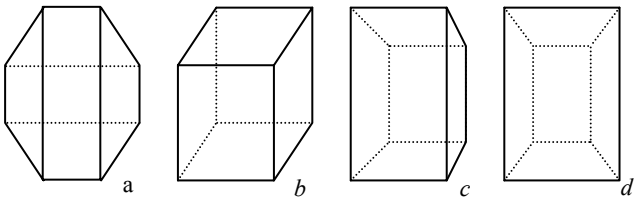

Figure 1. Four projections of a hexahedron, showing the four possibilities for the count discussed below: *a*: 0, *b*: 1, *c*: 2, and *d*: 4.

Schussman and Max [4] proposed a different sort of algorithm for a perspective view of a regular cubical grid, which classified the projections of a cube into one of a small number of cases, based on tests on the whole cube, guaranteeing topological consistency. Here we generalize this approach to hexahedra in a curvilinear grid.

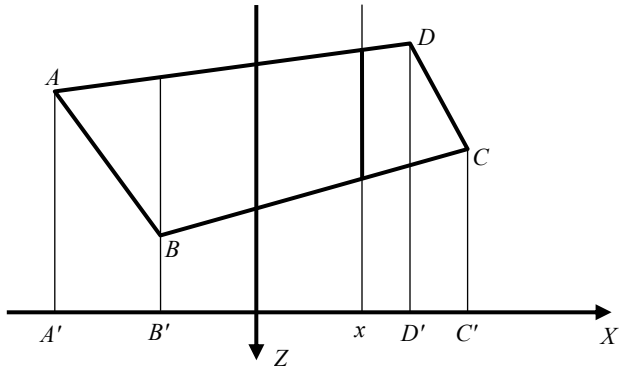

Figure 2. Parallel projection of quadrilateral *ABCD* to an image line along the *X* axis. The segment of the viewing ray through image point *x*, that lies within the quadrilateral, is shown as a bolder line. Its length varies linearly with the position *x* within the four segments *A'B'*, *B'D'*, and *D'C'*.

A cell in a curvilinear grid can be quite distorted, and one of its faces can project to a self-intersecting "bow-tie" quadrilateral, as shown in figure 3. For either way such a face is divided into two triangles, the two triangle projections overlap. In this case, for one of the hexahedra sharing the offending face, there is a viewing ray which exits the hexahedron through one of the overlapping triangles, and then re-enters it through the other. It is thus impossible to construct a visibility sort for back to front compositing. Similarly, a cell with a non-convex face, even without self-intersection, can result in overlapping quadrilaterals, which again allow a viewing ray to intersect the cell in two disjoint segments, as shown in 4. Therefore, we first test each hexahedron for non-convex faces. (Any self-intersecting face is non-convex.) If any are found, the cell is subdivided into five or six tetrahedra, and the Shirley-Tuchman triangle fans are used on the tetrahedra. There may also be degenerate cells, where one or more vertices coincide, for example, along the axis in cylindrical or spherical coordinates. Such cells are also divided up into tetrahedra, some of which may themselves be degenerate.

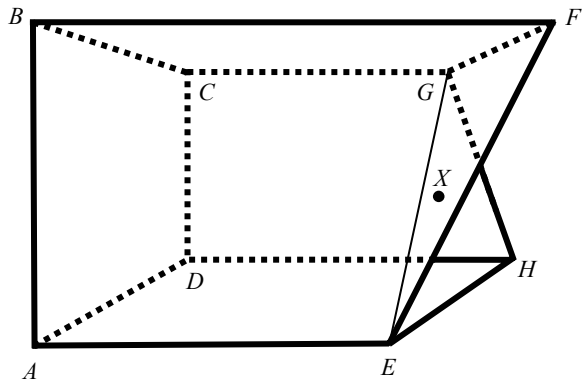

Figure 3. Hexahedron *ABCDEFGH* has a self-intersecting projected face *EFGH*. If diagonal *EG* is drawn to divide this face into two triangles, the ray at position *X* will intersect the resulting polyhedron in two disjoint segments. Note that there are three projected vertices *C*, *D*, and *G*, inside the projected face *ABFE*.

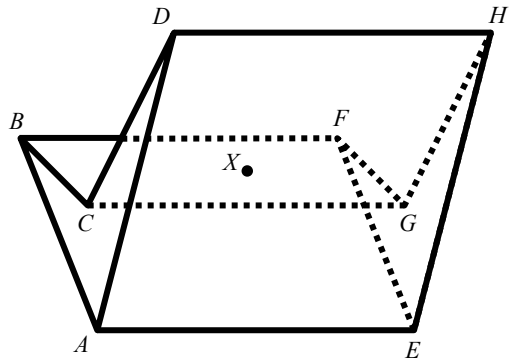

Figure 4. Hexahedron *ABCDEFGH* has two concave projected faces *ABCD* and *EFGH*. The ray at position *X* will intersect the concave polyhedron in two disjoint segments.

The goal of this paper is to classify the projections of the remaining hexahedra, and subdivide them into triangle fans or strips for hardware rendering. Since a single hexahedron can be rendered much more quickly than five tetrahedra, in terms of both vertex and fragment operations, this offers a significant speed up over subdividing all the cells into tetrahedra.

For our specific application in simulated X-ray images, we needed 32-bit floating point accuracy, which was available in the fragment programs of our graphics chips, but not in the compositing stage of the pipeline. In section 5 below, we explain how we did floating point compositing in the GPU.

In a curvilinear grid, the faces can be non-planar, so the assumption that the thickness varies linearly across the image plane polygons in the subdivision is not true even in an orthogonal projection, and is not true even for cubical cells in a perspective projection. The problems of this nonlinearity are discussed in Max, Williams, and Silva [5], and in section 7 below.

2. Projection Case Classification

The projection cases handled here include the three discussed in Schussman and Max [4], which can arise from the perspective projection of a cube. They are shown in figure 1 *b*, *c*, and *d*. There are several additional cases, such as the one shown in figure 1 *a*, which can occur only in curvilinear meshes. The test in [4] to distinguish the cases was simple, since it used the fact that the cell was a cube. For curvilinear grids, the tests are more involved, as described below.

The test first considers the six quadrilateral faces in turn, looking for non-convex faces, or faces which contain the projection of a vertex from the opposite face. This testing stops as soon as a face projection is found to contain one or more vertex projections. The test is done as follows. The line equations of the projections of the face's four edges are computed. For each line, the other two vertices of the face are checked to see if they are on the same side of the extended edge. If not, the projection of the face will be a concave or self-intersecting quadrilateral, and the cell is divided into tetrahedra. Next, the other four cell vertices which are not vertices of this face are tested with the four line equations, to see if any are contained in the face projection. If so, the number that do is saved in a variable called "count", and their vertex indices are also saved. There are four possibilities for count: 0, 1, 2, and 4, shown respectively in figure 1 *a*, *b*, *c*, and *d*. (A projection with count = 3 would necessarily have a bow-tie quadrilateral, as in Figure 3.) In figure 1 *b*, there are two quadrilaterals containing a vertex projection, but once the first one is found and processed, the containment testing will stop.

The vertices of our hexahedra are numbered as in figure 5. In order to label the vertices in the triangle fans in a standard order, a vertex index permutation corresponding to a 3D cube rotation is found so that the vertices of the face that contains the projected vertex or vertices end up with indices 0, 1, 2, and 3. A further rotation permutation insures that in the count = 1 case, the contained vertex has index 7, or in the count = 2 case, the contained vertices have indices 6 and 7.

3. Count 1 case

Let us start with the count = 1 projection topologies, which have the most different configurations in curvilinear grids.

In the cube projection situation shown in figure 5, edges V_7V_4 and V_0V_1 intersect in a new vertex V_8, and edges V_7V_6 and V_1V_2 intersect in a new vertex V_9. The two triangle fans list vertex indices 8, 1, 5, 4, 0, 3, 7, 1, and indices 9, 7, 3, 2, 6, 5, 1, 7.

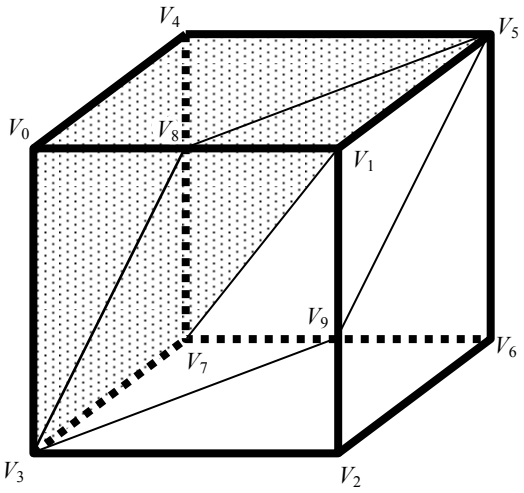

Figure 5. Vertex indices for the standard count = 1 case. The first triangle fan is shown shaded.

In curvilinear grids, one of these edge intersections may not be found. If vertex V_8 is not found because edges V_7V_4 and V_0V_1 do not intersect, as shown in figure 6, we look instead for V_8 at the intersection of edges V_7V_4 and V_1V_2, and another new vertex V_{10} at the intersection of edges V_7V_4 and V_1V_5. Three triangle fans are used. The first lists vertex indices 9, 6, 2, 3, 7, 8, 10, 5, 6; the second lists indices 8, 7, 3, 0, 1; and the third lists indices 10, 8, 1, 0, 4, 5.

In a similar case vertex V_9 is the one not found, and the revised vertex numbering is as shown in figure 7. We look for V_9 at the intersection of edges V_7V_6 and V_0V_1 and a new vertex V_{10} at the intersection of edges V_7V_6 and V_1V_5. There are again three triangle fans. The first lists vertex indices 10, 5, 6, 2, 1, 9, 8, 4, 5; the second list indices 9, 1, 2, 3, 7; and the third lists indices 8, 9, 7, 3, 0, 4.

Going back to the situation in figure 6, if V_8 is found at the intersection of edges V_7V_4 and V_1V_2, but V_{10} is not found at the intersection of edges V_7V_4 and V_1V_5, then we look instead for V_{10} at the intersection of edges V_1V_2 and V_4V_0. The configuration is then as in figure 8, and the two triangle fans list vertices 9, 2, 3, 7, 8, 4, 5, 6, 2, and 10, 7, 3, 0, 1, 5, 4, 8, 7.

There is a similar situation for the case in figure 7. If V_{10} is not found as expected at the intersection of edges V_7V_6 and V_1V_0, then we look for it at the intersection of edges V_2V_6 and V_1V_5. The configuration is then as in figure 9, and the two triangle fans list vertices 8, 4, 0, 3, 7, 9, 6, 5, 4, and vertices 10, 6, 9, 7, 3, 2, 1, 5, 6.

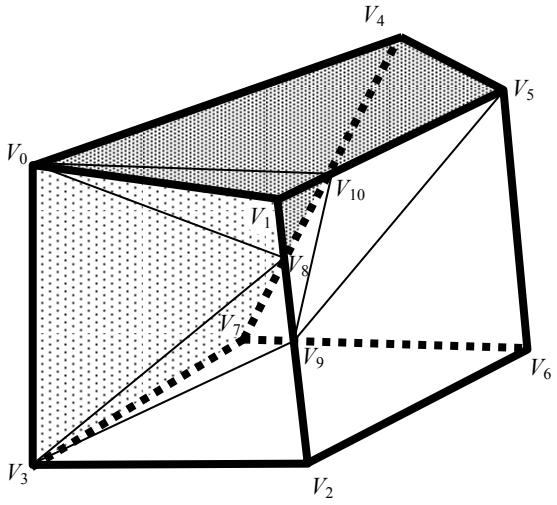

Figure 6. Vertex indices for alternate A of the count = 1 case. The three triangle fans are shown in different shades of grey.

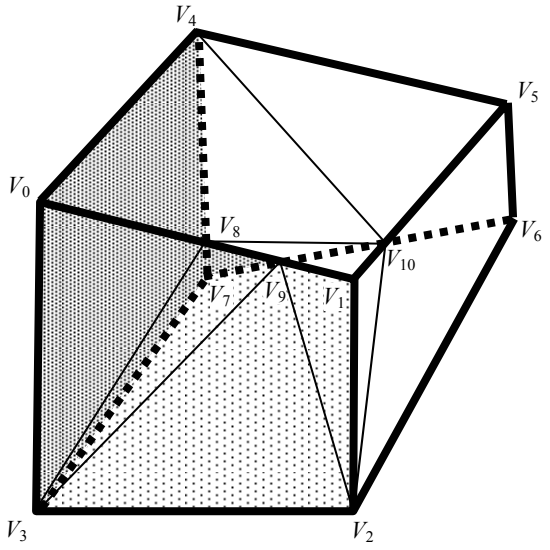

Figure 7. Vertex indices for alternate B of the count = 1 case.

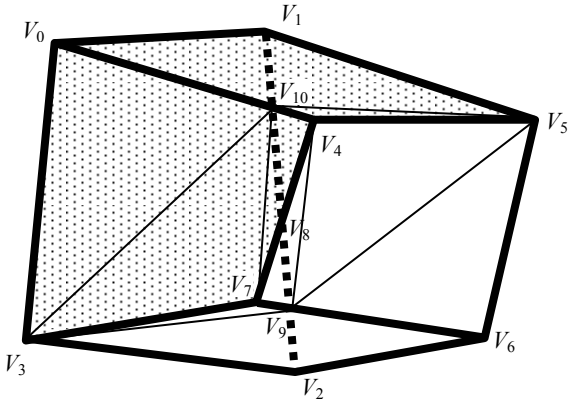

Figure 8. Vertex indices for alternate C of the count = 1 case.

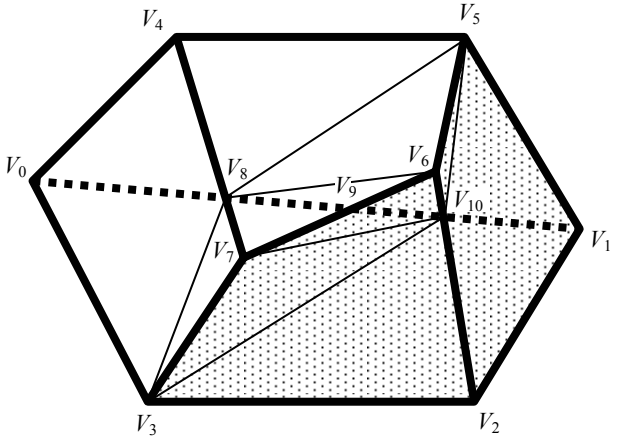

Figure 9. Vertex indices for alternate D of the count = 1 case.

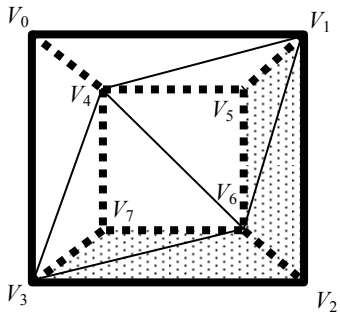

Figure 10. Vertex indices for the count = 4 case.

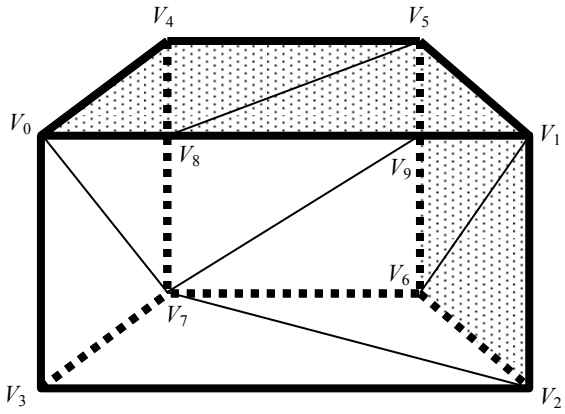

Figure 11.Vertex indices for the count = 2 case.

4. Other Counts

The count = 4 case shown in figure 10 needs no extra vertices. Two triangle fans are used. The first fan lists vertex indices 4, 0, 3, 7, 6, 5, 1, 0; and the second lists indices 6, 7, 3, 2, 1, 5.

The count = 2 case shown in figure 11 has a new vertex V_8 at the intersection of edges V_7V_4 and V_0V_1, a new vertex V_9 at the intersection of edges V_6V_5 and V_0V_1. As in [4], we use a triangle fan, with vertex indices 7, 8, 0, 3, 2, 6, 9, 8, and a triangle strip with vertex indices 0, 4, 8, 5, 9, 1, 6, 2.

The last case to consider is when count = 0. If no quadrilateral contains other projected vertices and there are no non-convex projections, the projected vertices form a convex octagon, as in figure 1 a, or figure 12. In this case, the vertex renumbering scheme is somewhat different. The vertex indices are permuted so that they run counter-clockwise around the octagon, as in figure 12. Each of the four diagonal projected edges that are not on the perimeter of the octagon belong to one quadrilateral whose other sides are part of the perimeter, and therefore must join a vertex i with vertex $(i + 3) \bmod 8$ or $(i - 3) \bmod 8$. If vertex 0 is connected by such a diagonal to vertex 5 (the $i - 3$ case), the vertices are renumbered by replacing index i by index $8 - i$, so that the projection topology is as in figure 12.

The new vertices are then found as shown in table 1. We use a triangle fan with vertex indices 8, 0, 1, 2, 9, 10, 11, 7, 0, and a triangle strip with vertex indices 7, 6, 11, 5, 10, 4, 3, 9, 2.

New Vertex	Intersecting Edges
V_8	V_0V_3 V_1V_6
V_9	V_0V_3 V_2V_5
V_{10}	V_4V_7 V_2V_5
V_{11}	V_4V_7 V_1V_6

Table 1.

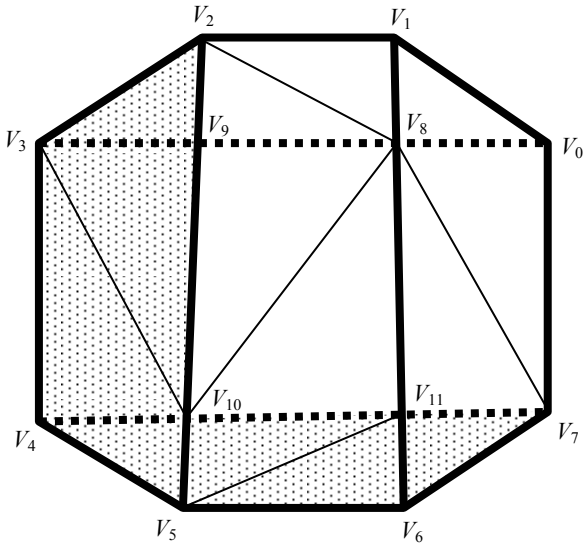

Figure 12. Vertex indices for count = 0 case.

5. Floating Point Compositing

In our simulated X-ray application, we needed floating point compositing. Initially, we considered absorption-only images. These involve only the total accumulated opacity along a ray, so that back to front visibility sorting is not required. We used a frame buffer object containing the accumulated image as a source texture, and also as a render target when we composited each new cell by a floating point fragment program, as described in sections 4 and 7.1 of Laney *et al*. [6]. It is a hazard to use the same frame buffer object as a source texture and a render target, because of the long delays in the pixel write pipeline, and because the texture cache may not be up to date. Instead of writing a large polygon to clear the write pipeline and texture cache, as described in [6], we now clear the write pipeline by reading out one pixel from the frame buffer object, and clear the texture cache by unbinding and rebinding the texture. In [6], rather than doing this after every cell is projected, we had an extra step to sort the cells into non-overlapping layers. One of the reviewers of [6] suggested that such layers could be found from standard visibility sort of the cells, and we now do this, as described below.

We are currently doing absorption plus emission X-ray simulations, to account for X-ray emission from hot materials. This is equivalent to standard color/opacity volume rendering, where the opacity may be wavelength dependent. We are using cell centered data, where the material variables affecting color (emissivity) and opacity (extinction coefficient) are constant on each cell. Since the RGBA fragment program can operate on four vector components in parallel, we can simulate four wavelength

bands at once. However, the exponential of minus the product of the extinction coefficient times the thickness (interpolated from vertex values by the scan conversion stage of the GPU), to get the transparency for compositing, must be done in four GPU instructions, because exponentiation does not currently operate on four-component vectors.

Color/opacity compositing requires a back-to-front visibility sort. The traditional directed graph visibility sort algorithm of Williams [7] has a node for each cell, and a directed edge for each face shared by two cells, pointing from the cell farther from the viewpoint to the adjacent one closer to the viewpoint, which directly occludes it. The algorithm of Cook *et al*. [8] augments this graph with directed edges for additional direct occlusion relations between cells across concavities or internal voids in the mesh. The "exterior" faces that are not shared by two cells are scan converted in software into an A-buffer (as in Carpenter [9]), which stores a depth-sorted list of such faces overlapping each pixel. The extra occlusion relations can then be discovered from adjacent pairs of entries in these lists.

Once the augmented visibility graph is constructed, a topological depth first search established the visibility order, as follows. An initial pass counts the number of incoming directed edges to each cell (from cells that it directly occludes), and puts those cells with no incoming edges into a first-in-first-out (FIFO) queue. Then, while the queue is non-empty, the cell at its head can be removed and composited onto the image, since it occludes no cell that has not already been composited. Each directed edge leaving the cell is followed to decrement the incoming edge count of the cell it points to, and if that cell's count becomes zero, it is added to the FIFO queue. (See [7] for details.)

In order to determine the non-overlapping layers of cells, a special token is inserted at the end of the FIFO queue after the initial pass. When this token is encountered and removed from front of the queue, a layer is complete, so the GPU write pipeline and texture cache are cleared as described above, and the token is inserted again at the end of queue to delimit the next layer. It costs time to clear the GPU pipeline after each layer. This is another reason to render hexahedra as a whole when possible, instead of dividing them all up into tetrahedra, which would increase the number of layers.

6. RESULTS

Figure 13 shows a volume made up of 32 curvilinear grids, with a total of 141,960 hexahedra. Among these hexahedra, 50,352 were degenerate, with two or more vertices coinciding, and 924 more had non-convex face projections. In both these cases, we subdivided the offending cell into six tetrahedra.

There remained 90,684 hexahedra. Among the count = 1 cases, there were 78,146 standard projections as in figure 5, 242 alternate projections as in figure 6, 532 as in figure 7, 16 as in figure 8, and 31 as in figure 9. There were 223 count = 4 cases as in figure 10, 11,333 count = 2 cases as in figure 11, and 161 count = 0 cases as in figure 12.

Using one processor of an 800 MHz dual Pentium4 Xeon PC, and an nVidia 5900FXUltra graphics card, the absorption only X-ray simulation in figure 13 took 5.27 seconds, of which 0.13 were used to read in the data, 3.5 were used to classify the cases, and 1.51 were used for preparing and rendering the triangle strips and fans.

Figure 14 shows a volume rendering of half a cylindrical shell, twisted to form non-planar faces for its 300 hexahedra. The sorting algorithm produced 43 layers, and drew 274 hexahedra and 156 tetrahedra in 1.25 seconds, of which .19 seconds were for sorting, .34 seconds were for classification and rendering, and the rest for input and output.

7. DISCUSSION

The projections discussed here were discovered one by one by analysing the cases that arose in projecting the data set in figure 13. No other cases were discovered among the projections of 10,000,000,000 hexahedra, with vertices chosen randomly inside a unit cube, but so far we do not have a proof that no others exist among projections of non-degenerate hexahedra with no bow-tie or concave quadrilaterals. In fact, the vast majority of the random hexahedra fell into these two rejected cases, but this is of course not the case for hexahedra arising from practical grids.

Figure 15 shows an extreme twisting of the cylindrical shell. The dark artifacts are caused by severely non-planar faces. The algorithm described here uses a piecewise-linear approximation to the thickness through the cell, which is linear on each triangle in the triangle fans. The subdivision of a face into triangles in these fans is not consistent for the two cells sharing the face, since the cells have other unshared edges contributing to their respective subdivisions. The artifacts in figure 15 are caused by this inconsistency.

A solution to this problem would be to compute for every pixel consistent depths for the front and back cell faces. For a bilinear parametric face interpolating four non-co-planar points, this involves solving a quadratic polynomial, which is unstable when the ray is close to tangent to the surface. Instead, we divide each quadrilateral into two triangles, using the diagonal starting from the vertex of lowest index. This choice of diagonal is consistent for the two cells sharing a face, and also with the method in Max [10] which we use to divide cells with non-convex face projections into tetrahedra. We use this method in software to find the thickness at triangle fan vertices like V_1 and V_7 in figure 5, whose viewing rays intersect the interior of the opposite face. It remains future work to compute such a thickness for every fragment in the GPU.

Another problem is with the degenerate tetrahedra which result from subdividing hexahedra with coinciding vertices. A tetrahedron with exactly two coinciding vertices will also have two coinciding non-degenerate faces. If the viewing ray is not tangent to these faces, then in the directed graph for the visibility sort, there must be one incoming directed edge and one outgoing directed edge for this cell. But since the cell has zero volume, it is impossible to decide which directed edge should be incoming, when considering the cell in isolation. Therefore, in order to get a consistent directed graph, this decision must be propagated incrementally across the grid, starting from some cell of non-zero volume, as described in section 4 of Williams [11].

Acknowledgement: This work was performed under the auspices of the U. S. Department of Energy by University of California, Lawrence Livermore National Laboratory under contract number W-7405-ENG-48. we thank Yang Liu and Henry Moreton for discussions on how to efficiently clear the GPU pipeline between layers, and Peter Williams for explaining the method in [11] for dealing with degenerate tetrahedra.

REFERENCES

[1] Peter Shirley and Alan Tuchman, "A Polygonal Approximation to Direct Scalar Volume Rendering", Computer Graphics Vol. 24, No. 5 (Special Issue on San Diego Workshop on Volume Visualization), ACM Press, pp. 63 – 70, 1990.

[2] Jane Wilhelms and Alan Van Gelder, "A Coherent Projection Approach for Direct Volume Rendering", Computer Graphics Vol. 25, No, 4 (Siggraph 1991 Proceedings), ACM Press and Addison Wesley, pp. 275 – 284, 1991.

[3] Nelson Max, Peter Williams, and Claudio Silva, "Approximate Volume Rendering for Curvilinear and Unstructured Grids by Hardware-Assisted Polyhedron Projection", International Journal of Imaging Systems and Technology, Vol. 11, pp. 53 – 61, 2000.

[4] Greg Schussman and Nelson Max, "Hierarchical Perspective Volume Rendering Using Triangle Fans", Volume Graphics 2001, Springer, Vienna, pp. 309 - 320, 2001.

[5] Nelson Max, Peter Williams, and Claudio Silva, "Cell Projection of Meshes with Non-Planar Faces", Data Visualization: The State of The Art (Post, Nielson, and Bonneau, editors), Kluwer, Boston, pp. 157 – 168, 2003.

[6] Daniel Laney, Steven Callahan, Nelson Max, Claudio Silva, Steven Langer, and Randall Frank, "Hardware Accelerated Simulated Radiography", Proceedings of IEEE Visualization 2005. pp. 343 - 350.

[7] Peter Williams, "Visibility Ordering Meshed Polyhedra", ACM Transactions on Graphics, Vol. 11 No. 2 (April 1992) pp. 103 – 126.

[8] Richard Cook, Nelson Max, Claudio Silva, and Peter Williams, "Image-Space Visibility Ordering for Cell Projection Volume Rendering of Unstructured Data", IEEE Transactions on Visualization and Computer Graphics, Vol. 10 No. 6, pp. 695 – 707.

[9] Loren Carpenter, "The A-Buffer, an antialiased hidden surface method", ACM Siggraph 1984 Conference Proceedings, pp. 1-3 – 108.

[10] Nelson Max, "Consistent Subdivision of Convex Polyhedra into Tetrahedra", journal of graphics tools, Vol. 6, no. 3, 2002, pp. 29 - 36.

[11] Peter Williams, "Parallel Volume Rendering Finite Element Data", in "Communicating With Virtual Worlds, Nadia and Daniel Thalmann, editors, Springer Verlag 1993, pp. 473 – 484.

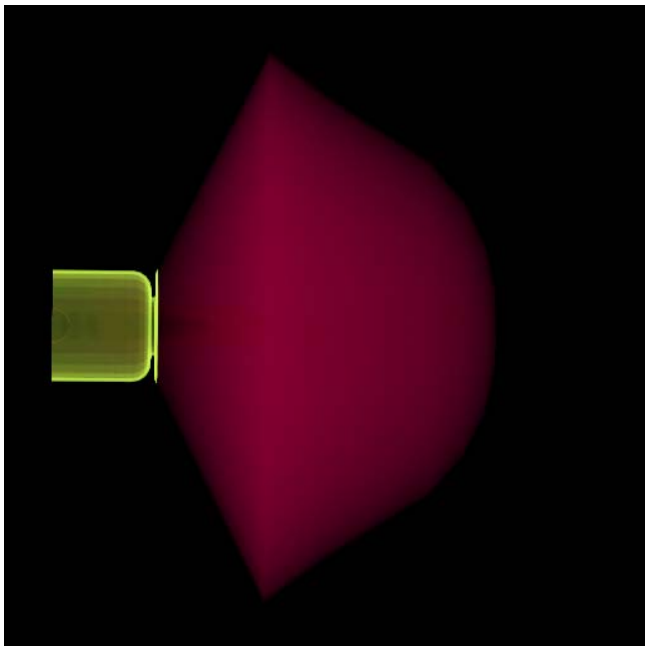

Figure 13. Projection of a half a laser fusion target. The RGB colors represent absorption at different wavelengths. The yellow region is half of the actual target, and the red region is a grid into which the target material spreads when heated by the laser.

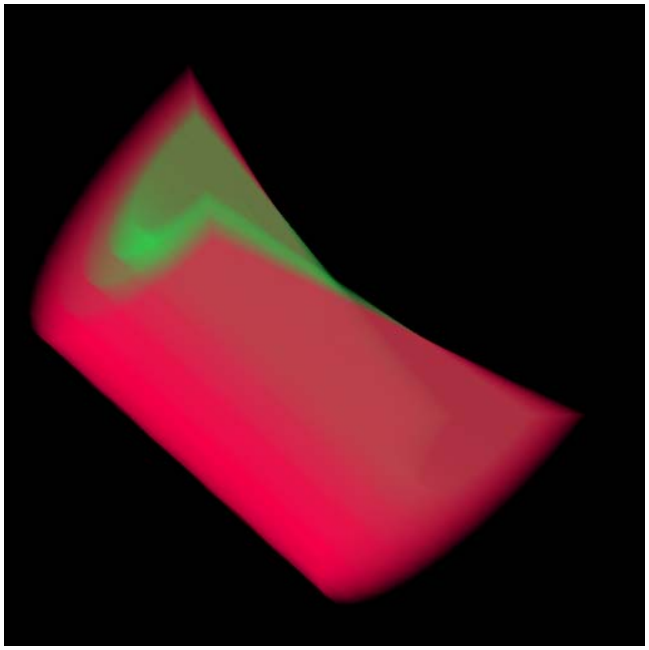

Figure 14. Volume rendering of a colored twisted cylindrical grid.

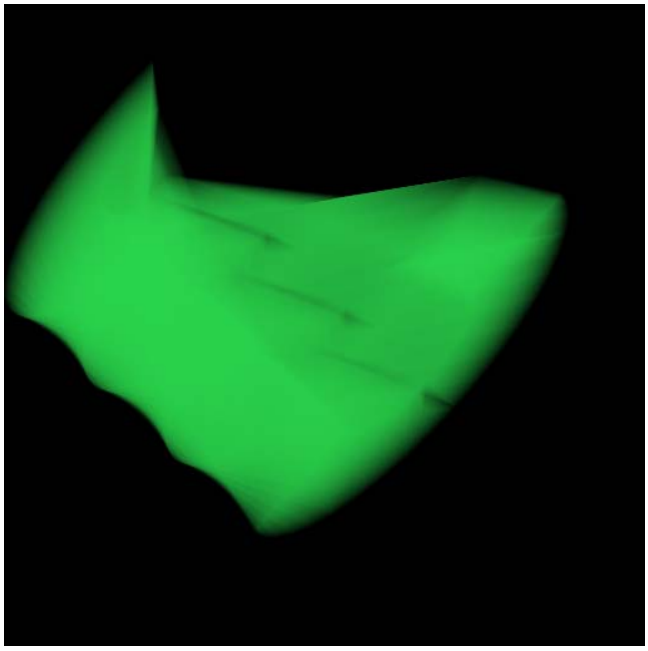

Figure 15. Artifacts from severely twisted hexahedra.