FUNDAMENTAL ALGORITHMS FOR PROJECTIVE VOXELIZATION

Reginald C. Jegathese
CAD Group, National Informatics Center
CGO Complex, New Delhi - 110 003, INDIA
crj@ciml.delhi.nic.in

Eustace Painkras
Department of Electrical Engineering
State University of New York at Stony Brook
Stony Brook, NY - 11794, USA
epainkra@sparky.ic.sunysb.edu

Edmond C. Prakash
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
Urbana, IL - 61801, USA
eprakash@ncsa.uiuc.edu

Abstract

We develop a voxel-based approach to volume modeling, in which the 3D object is represented as a set of voxels rather than a collection of surfaces. Interpretation with voxel-based representations is an intuitive paradigm, which has been shown to be theoretically sound and possesses enormous computational advantages over modeling with surface-based representations of 3D objects. However, there is no efficient algorithm for volume modeling of graphics primitives. In this paper, we attempt to bridge the gap by an efficient projective voxelization technique suitable for all existing 3D graphics primitives.

Keywords: Volume Graphics, Voxelization, CAD/CAM

1 INTRODUCTION

A widely accepted framework for modeling in graphics systems is the surface-based approach. The idea is to represent an object as a set of well defined primitives. These primitives are stored in an object data base which is combined with a rendering mechanism, used to visualize the object from its surface representation. Given the object database, which is assumed to capture the real object as a collection of primitives, the question is how close does this represent the real object? Is this the efficient, fast, and accurate way to consistently represent the real world object?

Solving this question is a real hard problem due to the discrete nature of the computational process. Many other forms of modeling and rendering have been developed over the past two decade at least partly to avoid the computational difficulties. These were also shown hard to compute. In order to mimic the real world object, increasing the number of primitives for representation by automatic sub-grid generation and global rendering techniques such as ray casting and radiosity have been attempted which take not less than a day on the most sophisticated systems today. This is likely to worsen as the model complexity grows exponentially. Not to add the texture modeling schemes which suffer from fast and large memory bandwidth requirements for real world textures which require multiple very large resolution textures.

A significant amount of recent work on modeling is influenced by convincing arguments of [KCY93] wherein volume modeling is a distinct mode of modeling and there is a computational theory that accounts for both its speed and flexibility. Most of the work in this direction still views modeling as a research topic as described in [WK93, WK94]. None of these works, however, meet the strong day-to-day usage requirements for efficient implementation and use of volume modeling.

Kaufman[Kau87b, Kau87a] presents voxel modeling algorithms for scan-plane voxelization of primitives. Abundant memory availability does not improve the performance due to the scan-line intersection approach in the design of these algorithms. Unfortunately the intersection computation is an expensive process which needs to be eliminated. Hardware schemes obviously accelerate performance. But an algorithm by Naeem and Yagel [SY95] which entirely depends on a particular hardware is not usable on any other system because of the dependency on a specific hardware. Implementing the same scheme on software is not computationally efficient either. Selecting a different model which is most suitable to the real world, is computationally hard. So we select a new method which solves all of these problems and therefore supports the voxelization based volume modeling of the entire spectrum of 3D graphics primitives which we show is feasible in our approach.

1.1 Voxel-Based Modeling

In this work we embark on the development of a voxel-based approach to volume modeling. It is not hard to motivate a voxel-based approach to modeling from a visualization point of view and indeed, many of the proponents of this approach to modeling have been visualization scientists. In the CAD/CAM community this approach can be seen as an example of CSG Boolean operations on voxels for geometric modeling and has already been studied in [CMP95]. The aim of this work is to further this concept for day-to-day use of volume graphics.

1.2 Recent Research

We now briefly describe the main contributions of the voxel-based approach developed in this paper. Our results can be grouped under 3 categories that can be informally described as follows:

- We show that projective voxelization supports correct determination of voxels.
- We define the set of primitives for projective voxelization and we show that these primitives can be used for efficient modeling.
- We show that projective voxelization lowers the dimensionality of the voxel modeling problem and therefore reduces the computational complexity of the problem.

We show that in many cases where modeling with the traditional surface representation is hard, we can perform efficient modeling with voxels. This includes operations such as voxelization and CSG.

1.3 Results

In this paper we characterize a set of projective algorithms for computationally fast and efficient functions for volume modeling. The preliminary results from this work indicates the feasibility of volume modeling by projective voxelization of primitives. The performance indicates a linear improvement in time with increasing number of primitives.

2 PRELIMINARIES

Rasterization: The process of determining the pixels that will provide the best approximation to the desired 'graphics primitive'. The resulting pixels are stored in the graphics memory known as the frame-buffer.
Graphics Primitive: is a set of functions which are not limited to 2D and 3D points, lines, polygons, curves, surfaces and polyhedra.
Voxelization: The process of determining the voxels that will provide the best approximation to the desired 'graphics primitive'. The resulting voxels are stored in the graphics memory known as the volume-buffer. Voxelization for volume graphics is analogous to rasterization for surface graphics.
Volume-buffer: A volume-buffer is a 3D logical frame-buffer, which can be assumed to have N_z frame-buffers to store all the N_z slices of the 3D world in discrete/voxel form. Since all the N_z slices are available, the traditional Z-buffer is made obsolete in volume graphics.
Addressing the volume-buffer: It is conceptually easiest to consider a voxel in a volume-buffer to have three-dimensional coordinates x, y, and z. Digital memory is however organized as a single linear list of addresses. Thus it is necessary to convert from the three-dimensional x, y, z representation to the linear list. Address is computed using the expression:

$$Addr = z \times x_{max} \times y_{max} + y \times x_{max} + x$$

where $x_{max} \times y_{max} \times z_{max}$ is the resolution of the 3D volume buffer.

3 MOTIVATION

Why is this type of volume modeling important? One reason is historical, the obvious evolution from pixels to voxels for graphics modeling. The second reason is functional, special hardware and software with abundant memory which helps realize a volume model that conveys lot more information than traditional models. The third reason is economical, where volume graphics is no longer expensive. With a GB of memory on a workstation becoming a reality, this cost effectiveness pushes the technology ahead at affordable cost. Finally, a new synergy among several modalities, in a seamless manner is achieved with volume modeling. With the integration of CAD and CAM, medical with synthetic simulations, etc., volume modeling is now practical with this emerging modeling scheme.

4 PROJECTIVE THEORY

4.1 Projective Line Voxelization

Definition 1. The voxels of a line segment in three dimensional Euclidean space E^3 can be determined by projecting the line onto that axis, to which it is most inclined.

Claim 1: The projected length or cartesian components of a line segment indicates the orientation of the line segment with the orthogonal axis.

Proof: Three mutually orthogonal unit vectors denoted by i, j and k known as the cartesian unit vectors, are parallel to the positive x, y and z directions respectively. A vector $\mathbf{v} = a\mathbf{i} + b\mathbf{j} + c\mathbf{k}$ is then a linear combination of cartesian component vectors, where a, b and c are the cartesian components. In 3D graphics, a line segment is represented as an end-point form (x_1, y_1, z_1) and (x_2, y_2, z_2). When represented in vector form it becomes $\mathbf{v} = (x_2 - x_1)\mathbf{i} + (y_2 - y_1)\mathbf{j} + (z_2 - z_1)\mathbf{k}$. The projected length $< dx, dy, dz >$ on the orthogonal axis $\mathbf{i}, \mathbf{j}, \mathbf{k}$ are then respectively, $|x_2 - x_1|$, $|y_2 - y_1|$ and $|z_2 - z_1|$. The maximum the projected length, the maximum is the orientation of a line to that particular axis. We use this property to determine the number of voxels for the discrete representation of a line segment.

Claim 2: For any line in E^3 all the voxels which lie along the line can be identified from the maximum projected length on the major axis.

Proof: We define a voxel-line or a discrete-line as a set of voxels which approximate a 3D line. Let us consider a line segment which lies on the X-axis. The projection of this line on the X-axis determines the number of voxels for that line segment.

Claim 3: A line in E^3, when projected, reduces to a computational problem in E^1.

Proof: A continuous line is represented as $y = f(x)$ in E^2 and $z = f(x, y)$ in E^3. While projecting this, we fix the increment to be 1 along the projected axis. This reduces the complexity of computation from E^3 to an one-dimensional problem which is a function of the projected length. In the case of a line, we index only on the axis which has the maximum length of projection. For example, if the line projects on the X-axis, then x is incremented by 1, the y component of the voxel is computed as a function of dx, the z component of the voxel is also computed as a function of dx. If there is a variation in scalar/color between the end points of the line segment, they are also computed as a function of dx.

Claim 4: The voxels obtained from projective voxelization are continuous in voxel space.

Proof: This condition where we increment the maximum projection by 1 voxel and other projection with a fraction less than 1, ensures all voxels determined by the projection approach is continuous. What we mean by continuous here in discrete voxel is they touch each other at least at a vertex, edge or face of a voxel.

Construction: Figure 1. shows the different buffers used in voxelization of a $3D$ line segment. The line $< x_1, y_1, z_1 >$ to $< x_2, y_2, z_2 >$ needs to be discretized into the component voxels. The first step is to identify the orientation of the line segment. In this test case the line projects on the X axis. The next step is to compute the amount of increment for x, y, z and *color*. Since the line projects on X axis, the increment along the X is 1. The difference in y, z and *color* are divided into $x_2 - x_1$ increments and are added as we march along X. The increment for the y component when we move from y_1 to the next voxel along the line is

$$\Delta y = (y_2 - y_1)/(x_2 - x_1)$$

$$\Delta z = (z_2 - z_1)/(x_2 - x_1)$$

$$\Delta c = (color_2 - color_1)/(x_2 - x_1)$$

For all the voxels along the x-direction we just increment as follows:

$$y_{i+1} = y_i + \Delta y$$

$$z_{i+1} = z_i + \Delta z$$

$$color_{i+1} = color_i + \Delta c$$

If the maximum orientation of the line is along Y, then the projection is done on the Y-axis. And similarly for Z. The $1D$ - buffers in the figure is just to indicate the components to be estimated for all voxels in the line. But in actual practice, we store the voxel values directly into the volume buffer and not in the $1D$ buffers. However the $1D$ buffers are used when we voxelize triangles. The complete algorithm is shown in Figure 2.

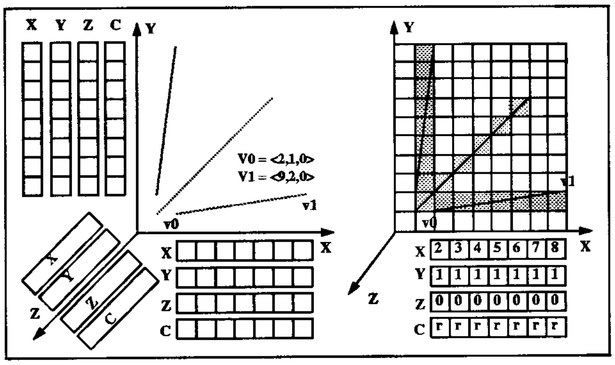

Figure 1: 3D Projective Voxelization

```
vox_line3d( x1, y1, z1, x2, y2, z2, color1, color2 )
 {   dx = x2-x1; dy = y2-y1; dz = z2-z1;
     adx = abs(dx); ady = abs(dy); adz = abs(dz);
     axis='X';
     if (ady > adx && ady > adz) axis = 'Y';
     else if (adz > adx && adz > ady) axis = 'Z';

     voxel(x1,y1,z1,color1);
     switch (axis) {
         case 'Y':
               compute increment x, z, color;
               for ( y = 1 to ady ) {
                   increment x, z, color;
                   voxel(x, y, z, color);
               }
               break;
         case 'X':
               compute increment y, z, color;
               for ( x = 1 to adx ) {
                   increment y, z, color;
                   voxel(x,y,z,color);
               }
               break;
         case 'Z':
               compute increment x, y, color;
               for ( z = 1 to adz ) {
                   increment x, y, color;
                   voxel(x,y,z,color);
               }
               break;
     }
 }
```

Figure 2: 3D Line Voxelization algorithm

4.2 Polygon Voxelization

Definition 2. The voxels of a convex polygon in E^3 can be determined by projecting the polygon onto the orthogonal plane to which the polygon is more inclined.

Claim 1: The projected length or cartesian components of a normal vector of a polygon indicates the orientation of the polygon with the orthogonal axis.

Proof: Figure 3. shows a triangle which lies on the XY plane. the normal to this plane is denoted by 0i + 0j + k. Hence the projected length is 1 on the Z-axis, which indicates the plane of projection is XY, which is identified from the plane perpendicular to k. In 3D voxelization of a polygon, the maximum the projected length of the polygon normal to an axis, the maximum the orientation of a polygon to the plane perpendicular to that particular axis. We use this property to determine all the voxels that lie on the polygon.

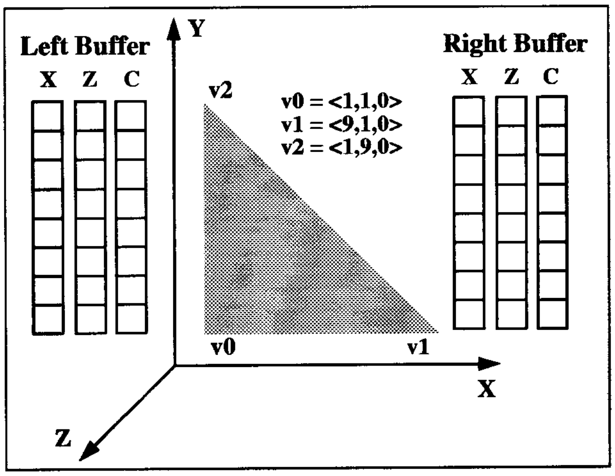

Figure 3: Projective Triangle Voxelization

Claim 2: A polygon in E^n when projected reduces to a computational problem in one dimensional space.

Proof: This is done as a two stage process: In stage 1, we process the boundary of the polygon by rasterization of the edges, by voxelizing along the edges. In the second stage, When the left and right voxel for each scan-line is known, the problem is to incrementally fill the interior, which is nothing but a one-dimensional linear interpolation problem. The total number of lines processed for a polygon will be the number of edges plus the number of scan-lines in the projection.

Construction: In our new projective voxelization technique, instead of layer-order, we traverse in triangle-order. Figure 3. shows the voxelization of a triangle. In the traditional rasterization, a scan-line intersects with the projection of a triangle. In our approach, voxelization of a triangle is however obtained by first traversing the boundary of the triangle. The three edges of the triangle are stored as components of x, y, z and *color* for the left and right extents of each scan-line in the triangle. As mentioned in the line voxelization earlier, if it is a right edge the start value of the line segment x_1, y_1, z_1 and *color*1 are stored in the $1D$ buffer. The next voxel along the line is obtained by incrementing $\Delta x, \Delta y, \Delta z$ and Δc. The same steps are repeated for all the edges of the triangles. If the polygon is facing the observer,i.e., if $(y_2 > y_1)$ that indicates that the edge should be stored in the right buffer. If $(y_1 > y_2)$ the edge goes to the left buffer. If the polygon is facing away from the observer, then the right and left buffers are swapped.

For a given scan-line the left buffer and the corresponding right buffer has the values of a line segment in $3D$ space. All the voxels between the left and right buffer are computed by incremental interpolation. The left buffer value is the starting voxel for this scan-line and the color is stored in volume buffer. All other voxels along the scan-line are computed by incremental interpolation. This shows that all our operations are just linear interpolation which is therefore an 1 dimensional problem. When all the scan-lines are computed, the volume buffer has the voxel triangle. Figure 4. shows the algorithm for voxelization of a test polygon, where the vertices lie on the XY plane. Since the normal is oriented towards the Z axis, the projection of polygon to XY is done for voxelization. Once all the edges are traversed, we use the left and right buffer x and z values to incrementally interpolate the interior information. If the orientation is towards Y the polygon is projected to XZ. If it is oriented towards X, then projection is done onto YZ. It is an extension of the rasterization technique, but instead of $2D$ pixels, $3D$ voxels are stored. A triangle is visited only once and all computations are therefore reduced to incremental interpolations.

```
vox_polygon3d( xyz[N][3], color[N] )
  {  compute_normal(xyz, a, b, c);
     axis = 'Z';
     if ( abs(a) > abs(b) && abs(a) > abs(c) ) {
         axis = 'X';
         interchange x and z for each vertex;
         interchange a and c normal components;
     }
     if ( abs(b) > abs(a) && abs(b) > abs(c) ) {
         axis = 'Y';
         interchange y and z for each vertex;
         interchange b and c normal components;
     }
     if (c > 0) { // front facing polygon
         scan_edges( to store in left & right buffers );
     }
     if (c < 0) { // back facing polygon
         scan_edges( to store in right & left buffers );
     }
     scan_face( y_min_max, left & right buffers )
  }

scan_edges()
  {  for ( each edge in polygon ) {
         compute increment x, z, color;
         initialize start x,z,color;
         if ( left edge ) { // y2 > y1
             for ( each y in the edge ) {
                 store x, z, color in left buffer;
                 increment x,z, color;
             }
         }
         if ( right edge ) { // y1 > y2
             for ( each y in the edge ) {
                 store x, z, color in right buffer;
                 increment x, z, color;
             }
         }
     }
  }

scan_face()
  {  for ( y = ymin to ymax in the polygon ) {
         lookup xleft and xright for y
         compute increment z, color;
         for ( x = xleft to xright ) {
             initialize start x, z, color;
             switch ( axis ) {
                 case 'Z':
                     voxel( x, y, z, color );
                     break;
                 case 'Y':
                     voxel( x, z, y, color );
                     break;
                 case 'X':
                     voxel( z, y, x, color );
                     break;
             }
             increment z, color;
         }
     }
  }
```

Figure 4: 3D Polygon Voxelization

4.3 Polyhedron Voxelization

Definition 3. Polyhedra: We define a voxel-polyhedron as a collection of all the voxels which lie in the interior and on the boundary of the polyhedron. This is carried out in two stages. In the first stage all the front and back facing polygons are projected separately. The interior of the polyhedron is then voxelized as a Beam of Voxels (BOV). This work has been reported earlier in [CMP95, PM95c, PM95b].

5 IMPLEMENTATION

The projective voxelization algorithms have been implemented and tested successfully with several polygonal mesh data. The results of line voxelization of a Flange and Space shuttle are shown in Figure 5 and Figure 7. The triangle voxelization is shown in Figure 6 and Figure 8 through Figure 10. This shows that this approach can be used for any 3D model consisting of lines, polygons and polyhedra.
The performance of the line and triangle voxelization algorithm on a R10K based SGI O2 system is shown in Table 1 and 2. The performance obtained shows that for practical applications we get a performance of 0.9 million lines/sec. The triangle voxelization performance of 0.13 million triangles/sec have been obtained using our method. These results show that our method is suitable for day-to-day voxelization of poyhedral models for applications such as RPT, 3D Volumetric Fax and for interactive volume visualization of solid models.

6 CURVES & SURFACES

In successful traditional graphics hardware like Reality Engine and Infinite Reality all primitives such as Circles, Curves, Cylinders, Spheres, and Surface are first converted into triangles or lines. Once they are converted into lines and triangles our voxelization

Table 1: Performance of line voxelization

Graphics Object	Number of Lines (lines)	Rate of Voxelization ($\times 10^6$ lines/sec)
Shuttle	4350	.311
Flange	15912	.418
Bunny	208353	.665
Dragon	2614242	.942

Table 2: Performance of triangle voxelization

Graphics Object	Number of Triangles (triangles)	Rate of Voxelization ($\times 10^6$) triangles/sec)
Shuttle	1450	.018
Flange	5304	.086
Bunny	69451	.102
Dragon	871414	.139

scheme can be applied for all such curves and surfaces.

7 DISCUSSION

Volume graphics has its own bottlenecks and as time progresses, with faster CPUs and large memory on low-end desktops the ultimate dream of aving a volume graphics station is within reach. We discuss some of the issues and how our method handles them.

- **Comparison with Surface Rendering:** The advantages/disadvantages of *volume graphics* versus *surface graphics* has discussed in greater detail by Kaufman et. al. [KCY93]. The algorithm described in this paper caters to the demand for faster voxelization to generate volume data.
- **Comparison with Bresenham, DDA, Kaufman for Line Voxelization:** Bresenham and DDA computes pixels, whereas our method computes voxels. The Kaufman's Line Voxelization extends to a 3D-DDA with 6 or more connected voxels. Our implementation is faster since it does not have the overhead of the error term in Bresenham, DDA or 3D-DDA.
- **Comparison with Kaufman's Polygon Voxelization:** The major computation in Kaufman's Polygon voxelization [Kau87a] is the intersection computation of the polygon with each scan-plane. Our method doesn't require that.
- **Quantity of Voxelization:** Our emphasis in this work is to develop an algorithm which can do practical day-to-day voxelization for volume graphics. And we have shown this from the timing shown on the smallest low cost US$5000 workstation available in the market today.
- **Quality of Voxelization:** As in 2D rasterization, aliasing is inherent to 3D voxelization. Several anti-aliased schemes for projection algorithms are under implementation. The antialiasing techniques proposed in [PM95a] and [WK94] can be applied during voxelization to achieve better quality.
- **Memory Requirement:** It is indisputable that volume graphics is memory intensive. This problem has been handled by researchers in several directions: compression, rendering compressed volumes, hierarchical organization, etc. Our method currently uses the full volume buffer, but work is in progress to reduce the memory use.
- **Volume Morphing after Voxelization:** Another section of work under progress is to make these algorithms efficient so that we can perform volume morphing and volume manipulation. This can be achieved by implementing fast voxelization algorithms on parallel systems and in graphics hardware.

8 CONCLUSION

New simple and efficient algorithms for incremental voxelization for Lines, Polygons, and Polyhedra have been developed. The algorithm has been tested for a variety of polygonal meshes. These primitives can also be used for all other graphics primitives, including Circle, Cylinder, Curves, and Surfaces. Our modeling scheme is independent of the rendering method but on the other hand compatible with all existing rendering schemes.
As in 2D scan-conversion, aliasing is inherent to 3D voxelization. Several anti-aliased schemes for projection algorithms are under implementation. Another section of work under progress is to make these efficient algorithms for volume modeling, volume manipulation and voxel representation as a library for public use. The topic for future research is realization of the fast voxelization algorithms for parallel and hardware implementations.

References

[CMP95] V. Chandru, S. Manohar, and C. E. Prakash. Voxel-based modeling for layered manufacturing. *IEEE CG & A*, 15(6):42–47, Nov 1995.

[Kau87a] Arie E. Kaufman. An algorithm for 3D scan-conversion of polygons. *Eurographics '87 Proceedings, G. Marechal(ed.), North Holland, Amsterdam*, pages 197–208, Aug 1987.

[Kau87b] Arie E. Kaufman. Efficient algorithms for 3D scan-conversion of parametric curves, surfaces and volumes. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):171–179, Jul 1987.

[KCY93] Arie E. Kaufman, Daniel Cohen, and Roni Yagel. Volume graphics. *IEEE Computer*, 26(7):51–64, Jul 1993.

[PM95a] C. E. Prakash and S. Manohar. Error measures and 3D anti-aliasing for voxel data. *Proc. of Pacific Graphics '95, World Scientific Publishing*, pages 225–239, Aug 1995.

[PM95b] C. E. Prakash and S. Manohar. Hardware architecture for voxelization-based volume rendering of unstructured grids. *Proc. of Workshop on Eurographics Hardware, Maastricht*, pages 103–115, Aug 1995.

[PM95c] C. E. Prakash and S. Manohar. Volume rendering of unstructured grids: A voxelization approach. *Computers and Graphics*, 19(5):711–726, Sep/Oct 1995.

[SY95] Naeem Shareef and Roni Yagel. Rapid previewing via volume-based solid modeling. *Proc. of third Symposium on Solid Modeling '95, ACM Press*, pages 281–291, May 1995.

[WK93] Sidney W. Wang and Arie E. Kaufman. Volume sampled voxelization of geometric primitives. *Proc. of IEEE Visualization '93*, pages 78–84, Oct 1993.

[WK94] Sidney W. Wang and Arie E. Kaufman. Volume sampled 3D modeling. *IEEE CG & A*, 14(5):26–32, Sep 1994.