Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering

David Laur and Pat Hanrahan

Princeton University
Princeton, NJ 08544, USA

Abstract

This paper presents a progressive refinement algorithm for volume rendering which uses a pyramidal volume representation. Besides storing average values, the pyramid stores estimated error, so an oct-tree can be fit to the pyramid given a user-supplied precision. This oct-tree is then drawn using a set of splats, or footprints, each scaled to match the size of the projection of a cell. The splats themselves are approximated with RGBA Gouraud-shaded polygons, so that they can be drawn efficiently on modern graphics workstations. The result is a real-time rendering algorithm suitable for interactive applications.

CR Categories and Subject Descriptors: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism.

Key Words: volume rendering, coherence, progressive refinement, interactive techniques.

1 Introduction

Volume visualization is a collection of techniques for visualizing 3D functions. The earliest methods extracted conventional computer graphics primitives such as surfaces, curves, or points, and then displayed them. More recent methods render the volume directly, without this intermediate conversion. This involves forming an RGBA (color and opacity) volume, and projecting it from the desired point of view. RGBA volumes can represent both interiors and the surfaces representing the boundaries between different regions. If just surfaces are shown, the pictures look quite similar to those generated by first extracting surfaces and then rendering them. However, if interiors are also shown, they appear as clouds with varying density and color. A big advantage of volume rendering is that this interior information is not thrown away; a disadvantage is that cloudy interiors are hard to interpret.

This paper presents an algorithm for rendering opacity projections at interactive rates on a typical high performance graphics workstation. Motion is very helpful in understanding opacity projections. For example, the output of commercial medical imaging systems generate film loops and not just static imagery. Furthermore, the amount of information gained from a motion study is much greater if the motion is under interactive control, for then the user can vary the motion to highlight what they are currently focusing on.

Our algorithm is based on two key ideas: *coherence* and *progressive refinement*. Recent research has shown how to take advantage of coherence when performing opacity projections of large cells filled with cloudy material [9; 8; 6; 11]. The most relevant to the work reported here are those methods that approximate the projection with a collection of Gouraud-shaded RGBA polygons [8; 11]. Progressive refinement involves simplifying either the model or the rendering algorithm, or both, until pictures can be produced at interactive rates, and then computing successively better images when free time is available, for example, when the user pauses to examine an interesting image [1].

This paper proposes a *splatting algorithm* [3; 10] that works on a pyramidal representation of the volume. Splatting works by first sorting cells from back to front and then compositing the projection of each cell, called its *footprint*, into an accumulating projection image. Our algorithm builds a set of footprints at different sizes—one for each level in the pyramid. The time to draw a splat is constant, or at worst proportional to its area, so substantial time is saved by drawing a single large splat instead of a volume of smaller splats. More interestingly, the algorithm does not just draw a reduced resolution version of the volume, but determines the number of the splats by fitting a collection of cells at different resolutions in the pyramid to the original data based on a user-supplied *error criteria*. Progressive refinement proceeds by gradually reducing the error associated with the fit.

2 Reconstruction and Projection

The ideal volume rendering algorithm performs the following three steps: (i) reconstructs the continuous function from the discrete samples, (ii) transforms the continuous function for viewing, and (iii) evaluates the opacity integral along each line-of-sight. Splatting algorithms approximate this procedure. The reconstruction function is transformed according to the current viewing transformation, and then is projected using the opacity integral to form a 2D footprint. There is only one footprint per view per volume if the data is uniformly sampled and the viewing transformation is a parallel projection. To generate the complete image, the footprints are composited on top of each other in back to front order[1]. Splatting algorithms are not equivalent to the idealized rendering process outlined above. The reconstruction and projection steps cannot be reordered if the reconstruction functions from different samples overlap, but this is necessary to

[1] Actually Westover's algorithm reconstructs all the samples from a planar slice into a slice image, and then composites the slice image onto the accumulating final image.

Figure 1: Different splat shapes and their footprints.

generate a smooth continuous images.

Current splatting algorithms simplify this procedure even further. Westover approximates the footprint with a 2D elliptical Gaussian [10]. This is an approximation because the reconstruction function and scattering model are never specified. More recently, methods have been developed for rendering the opacity projections of polyhedral cells using RGBA polygons. Wilhelms breaks the projection of a cubical cell into topologically uniform regions, and then computes the opacity at the vertices according to various projection formula [11]. A similar approach can be applied to tetrahedra, which have the advantage that there are only two topologically distinct cases to consider [8].

We have built a flexible system based on approximating footprints with a collection of Gouraud-shaded polygons. These polygons can be used to build a piecewise linear approximation to any footprint function. A generalized "Gaussian" splat is defined by (1) the number of angular subdivisions, (2) the number of radial subdivisions, and (3) the opacity of the splat at a given radial distance. Some typical splats are shown in Figure 1.

The advantage of outputting Gouraud-shaded polygons is that workstations have been optimized to draw them for surface rendering applications. Recent workstations have added the ability to interpolate opacity along with color, and to provide hardware assist for compositing [7]. Also, since the graphics hardware is handling the transformation and scan conversion, both orthographic and perspective projections are possible. Furthermore, if the polygon is point sampled correctly and if the vertices have subpixel position, then the center of the splat can also be positioned accurately. Our algorithm approximates the volume at multiple resolutions, and hence, we need to draw different sized splats. This is easily done by simply scaling the polygons, and hence, the amount of data describing a splat is independent of splat size and the overhead involved in transferring the splat to the graphics engine is independent of size. The cost of rendering a splat, however, does involve an area dependent term, because the time needed for scan conversion and compositing depend on the actual number of pixels output.

Figure 2: Plot of opacity correction.

There is one major complication when rendering different sized splats, and that is that doubling the size of the splat more than doubles its opacity. This is an important effect that we would like to capture qualitatively. To approximate this effect, we compute the transparency of a single double-sized Gaussian splat that is equivalent to compositing two smaller unit-sized splats directly on top of each other.

$$(1-\alpha_2)e^{-x^2/\sigma_2^2} = ((1-\alpha_1)e^{-x^2/\sigma_1^2})^2$$

where α_1 and σ_1 are the opacities and standard deviations, respectively, of the smaller unit splat, and α_2 and σ_2 are the corresponding derived quantities for the double-sized splat. The effective opacity and sigma for the larger splat are

$$(1-\alpha_2) = (1-\alpha_1)^2$$

and

$$\sigma_2^2 = \sigma_1^2/2$$

Thus, when a splat is doubled in size, a new opacity can be computed using this formula. Figure 2 plots this *opacity correction* as a function of opacity and resolution. As can be seen, this is a large effect. Note also that the cross-sectional shape of the footprint also changes with scale. Regions close to the center are less transparent than regions far from the center, causing the splat to appear narrower and more concentrated. However, this effect is countered by the fact that not only are splats composited in depth, but they also cover the image in x-y; this causes the two smaller splats to actually be larger than the double-size splat. For this reason, we simply scale splats and ignore any shape changes.

Figure 3 shows a set of splats at different resolutions with and without opacity correction. Figure 4 shows the projection of a constant-valued volume with different sized splats, with and without the opacity correction. Ideally, changing the resolution should have no effect. Although our approximation does not work perfectly, it is much better than using no correction, as can be seen in the examples.

3 Hierarchical Traversal

To take advantage of the ability to draw different sized splats efficiently, we build a multiresolution representation of the original volume. In this paper, we use the word "pyramid" to indicate a complete resolution set, and the word "oct-tree" to indicate some subset of the pyramid that completely spans the volume. Pyramids and oct-trees have been used

Figure 3: Footprint of a single splat vs. power-of-two resolution. The upper row shows the uncorrected opacity projection, and hence, as the splats increase in size they become more transparent. The lower row shows the corrected opacity projection.

previously for volume rendering. Levoy [4] used a binary pyramid to indicate the presence of non-transparent material to accelerate ray tracing. Levoy [5] also used a pyramid of averaged values, commonly called a *mip-map* [13]. for gaze-directed rendering. Others have used *min-max* or range pyramids to allow efficient iso-surface extraction [14; 2; 12].

The regular structure of pyramids and oct-trees allows them to be traversed in front-to-back or back-to-front order just as easily as uniformly sampled voxel arrays. Note that the order in which the children are traversed is the same everywhere for a parallel projection, but may change for a perspective projection.

In the pyramid that we build, each node contains several pieces of information. As with a mip-map, every node contains the average RGBA value of all its children. These values are used for drawing the splats. Since the RGBA assignments depend on the current scalar to RGBA mapping, the mip-map needs to be recomputed every time the transfer function changes. Also since the RGBA assignment depends on shading, only view-independent shading formula are handled efficiently by this approach.

Every node in the pyramid also contains a variable indicating the average error associated with that node. This error term measures the average cost of approximating this region of space with a constant function equal to the node's average value, rather than with the original set of voxel values contained in the region. The root mean square error due to this approximation is

$$e_j^l = \sqrt{\frac{\sum s_i^2}{n_j^l} - (\frac{\sum s_i}{n_j^l})^2}$$

where e_j^l is the error associated with node j at level l, n_j^l is the number of voxels comprising that region, and s_i is the value of voxel i. This error measure can be computed efficiently using a single traversal of the pyramid.

To approximate the volume most efficiently, we would like the error per unit volume to be uniformly distributed throughout the volume. Moreover, we would like to use the lowest resolution in the pyramid that falls within the allowable error. Since the error in the approximation is data dependent, different regions in the volume will in general need different levels of detail. We can think of this process as fitting an oct-tree to a pyramid. This fit can be performed with a single traversal of the pyramid. At each level in the pyramid, the desired error per unit volume is compared to the average error for the given node, if the average error is less than the desired error, the traversal is terminated at this level, otherwise it proceeds downward.

Figure 4: Projection of a constant-valued volume with different sized splats. The left column shows the resolution, the middle column shows the uncorrected opacity projection, and the right column the corrected opacity projection.

Figure 5: Oct-trees with different error contours. The colors of the lines are derived from the data.

Several oct-trees, with decreasing total error, are shown in Figure 5. Note the desired effect, where the data changes rapidly it is approximated with many small nodes, and where it changes slowly, it is approximated with fewer large nodes. Furthermore, the number of nodes varies with error; the lower the error the more nodes. Thus, this error representation is ideal for automatically adjusting the number of nodes in the volume representation, as required for progressive refinement.

4 Results

Figure 6 shows an industrial CT-scan of an engine block drawn at three different resolutions. The resolutions were

Figure 6: Engine block (256x256x110).

chosen by the following three criteria: (i) What can be drawn 5 times a second—this is what the image looks like during motion, (ii) what can be drawn in 5 seconds—this is what the image looks like after motion has stopped and the image has been refined for a reasonable amount of time, and (iii) what the resulting image looks like at full resolution.

Several conclusions can be drawn. First, the highest resolution pictures do not look as good as the very best previous volume rendering techniques. The approximations introduced to achieve interactive rates, unfortunately, still sacrifices quality for speed. Second, although the lowest resolution image looks crude when viewed as a still, it is surprisingly effective when viewed in motion.

5 Discussion

Previous approaches to using multi-resolution volume representations were based on the discrete concept of pruning away regions which didn't contribute to the final answer. A desirable property of the algorithm described in this paper is that the pruning is function of both opacity and accuracy. Opacity indicates presence, and if no material is present, cells are pruned just as in the previous work. However, the error indicates the accuracy of the approximation, and low error in a large region indicates that it is homogeneous, and homogeneous regions can be drawn efficiently using large splats. We expect that these ideas also can be used to accelerate high-quality ray tracing-based volume rendering algorithms. We also think this multi-resolution approximation scheme has many other applications in computer graphics.

Acknowledgments

This research was supported by an equipment grant from Silicon Graphics.

References

[1] Larry Bergman, Henry Fuchs, Eric Grant, and Susan Spach. Image rendering by adaptive refinement. *Computer Graphics (SIGGRAPH '86 Proceedings)*, 20(4):29–38, August 1986.

[2] Jules Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5(4):341–355, November 1988.

[3] Harvey E. Cline, William E. Lorensen, Sigwalt Ludke, Carl R. Crawford, and Bruce C. Teeter. Two algorithms for the reconstruction of surfaces from tomograms. *Medical Physics*, 15(3):320–327, June, 1988.

[4] Marc Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.

[5] Marc Levoy and Ross Whitaker. Gaze-directed volume rendering. *Computer Graphics (Symposium on Interactive 3D Graphics)*, 24(2):217–223, March 1990.

[6] Nelson Max, Pat Hanrahan, and Roger Crawfis. Area and volume coherence for efficient visualization of 3d scalar functions. *Computer Graphics (San Diego Workshop on Volume Visualizaton)*, 24(5):27–33, November 1990.

[7] Thomas Porter and Tom Duff. Compositing digital images. *Computer Graphics (SIGGRAPH '84 Proceedings)*, 18(3):253–260, July 1984.

[8] Peter Shirley and Allan Tuchman. A polygonal approximation to direct scalar volume rendering. *Computer Graphics (San Diego Workshop on Volume Visualizaton)*, 24(5):63–70, November 1990.

[9] Craig Upson and Michael Keeler. V-buffer: Visible volume rendering. *Computer Graphics (Proceedings of SIGGRAPH '88)*, 22(4):59–64, August 1988.

[10] Lee Westover. Footprint evaluation for volume rendering. *Computer Graphics*, 24(4):367–376, August 1990.

[11] Jane Wilhelms. A coherent projection approach to direct volume rendering. *Computer Graphics (SIGGRAPH '91 Proceedings)*, July 1991.

[12] Jane Wilhelms and Allan Van Gelder. Octrees for faster isosurface generation. *Computer Graphics (San Diego Workshop on Volume Visualizaton)*, 24(5):57–62, November 1990.

[13] Lance Williams. Pyramidal parametrics. *Computer Graphics (SIGGRAPH '83 Proceedings)*, 17(3):1–11, July 1983.

[14] Brian Wyvill, Craig McPheeters, and Geoff Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, 1986.