Fast Volume Rendering of Curvilinear Data Sets Using a Splatting Approach

Torsten Möller[1,2], Roger Crawfis[1,2], Nelson Max[3]

[1]Department of Computer and Information Science
[2]The Advanced Computing Center for the Arts and Design
The Ohio State University
Columbus, Ohio

[3]Lawrence Livermore National Laboratories
Mail Stop L-639
7000 East Avenue
Livermore, CA 94550, USA

{crawfis, moeller}@cis.ohio-state.edu, max2@llnl.gov

Abstract

We develope a new algorithm, that visualizes curvilinear grids fast and accurately. Our algorithm is based on the concept of splatting. In order to reconstruct the underlying function we use a Gaussian reconstruction kernel. In computational domain the sampling of the data field is regular and therefore we have spherical reconstruction kernels. However, the mapping into physical space is non-linear and the spherical reconstruction kernels from computational space are mapped to what we call "bean-shapes", which are usually not ellipsoids. We approximate these bean-shapes with ellipsoids and project them to the screen. Our algorithm is similar to Mao's splatting algorithm, but is much more efficient and less computationally involved. It also more faithfully represents the underlying data.

1. Introduction

Volume rendering displays a three dimensional sampled data cloud on a two dimensional computer screen. The data cloud is sampled at points on an underlying data grid. There are two different types of common data grids - *structured* and *unstructured* [8][11] (see Fig. 1). Structured data grids are based on an integer lattice (i, j, k). All data samples are placed on this integer lattice. The actual location of the sample points in 3D (or what we call *physical* space) is defined by a continuous mapping w of the integer lattice to physical space, i.e. $w(i, j, k) = \left(\begin{bmatrix} x & y & z \end{bmatrix} \right)^T$. According to the mapping we can further distinguish between several types of structured grids. *Curvilinear* grids impose few constraints to this mapping (e.g. non overlapping). Furthermore, *rectilinear* grids are structured grids, where the i, j, and k components each have only separate 1D mappings. *Uniform* grids are rectilinear

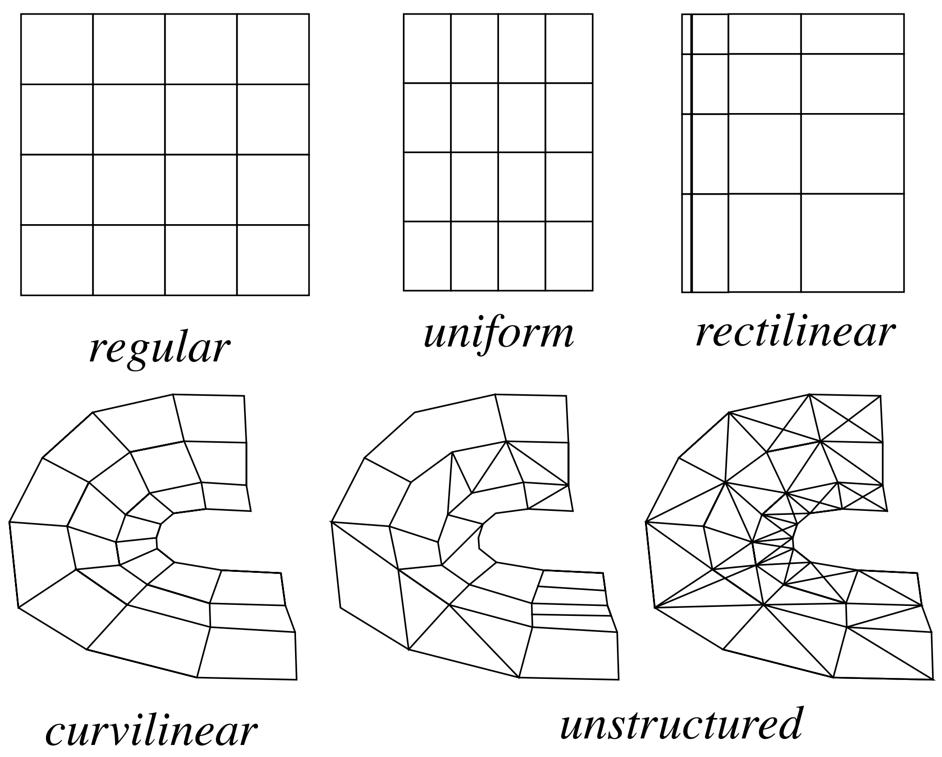

FIGURE 1. Different types of computational grids.(Taken from [10])

grids, where the mapping is linear. Lastly, *regular* grids are rectilinear grids, that are having the same mapping for each of the i, j and k components. The most general type of grid is an unstructured or irregular grid. These are usually specified by a simple listing of the location of the sample point and it's value without any specified connectivity information between the sample points.

Developing Volume Rendering techniques to visualize these different data sets has been a challenge in recent years. Interactive rates are needed so as to give better visual clues for the three dimensional appearance of objects. Some new algorithms achieve interactive rendering rates [2][10] using fast rendering hardware [1]. Unfortunately, these algorithms only work on regular or uniform data sets. The interactive volume rendering of curvilinear or unstructured data sets is more problematic here to date.

We present a new algorithm, that visualizes three dimensional curvilinear grids fast using a splatting technique for volume rendering. Section 2 of this paper summarizes recent work in the field. In Section 3 we introduce our new algorithm. The speed of our algorithm and the quality of the rendered images is assessed in Section 4. Finally in Section 5 we summarize our results and discuss possible extensions of our work.

2. Related Work

Garrity [4] uses an efficient ray tracing implementation to render unstructured data sets. He intersects the ray with all the faces of the current cell. The next intersection closest to the entry point is then the exit point of the cell. The exit point is also the entry point of the next (neighboring) cell. In order to find the first entry point into the volume, he intersects the ray with all exterior faces.

He computes the data values at the intersection points by linear interpolating between the data at the vertices.

Shirley and Tuchman's Projected Tetrahedra algorithm [7] exploits spatial cell coherency and uses hardware acceleration. Their algorithm requires a tetrahedral decomposition of the underlying data set. The basic idea is that a tetrahedra projects into at most four continuous triangular regions onto the screen. The opacity and color can be computed for the vertices of these triangles. Then the color and opacity are linearly interpolated across the triangles. All triangles produced can be rendered very quickly using special graphics hardware.

Max et al. [6] present a similar algorithm for convex polyhedra or scattered data sets. Under certain simplifying assumptions, they also compute the exact volume integral along the ray analytically.

In the slicing algorithm by Yagel et al. [10] the underlying grid is cut into slices parallel to the viewing direction. The slices consist of polygons, that again can be drawn quickly by special rendering hardware. Although the slicing operation needs to be repeated for each new viewing direction, it provides a direct image quality - render time trade-off through the choice of the number of slices.

Our algorithm is based on the splatting idea of Westover [9]. In most algorithms we reconstruct the sampled data set in three-dimensional space first and then project it to the screen. In splatting however, the contribution of a sample point (the splat) is projected to the screen and combined with previous splats in screen space for the final image. Traditional splatting uses spiracle reconstruction kernels. This is a sufficient reconstruction kernel for regular grids, but not for curvilinear or irregular grids. Fig. 4 illustrates now that spherical reconstruction kernels perform poorly on a curvilinear grid. Fig. 4 shows the blunt-fin data set. We can see, that the sampling is too sparse and in some areas the sampling is too dense.

Mao's et al. [5] solution is to redistribute many new points according to the curvature of the grid and the size of the grid cells using a Poisson disk distribution. They interpolate the values of these points from the eight neighboring grid points by trilinear interpolation. Then they approximate the density distribution of these new points with ellipsoids according to the local curvature of the grid, which is also interpolated from the curvature at the grid points. All ellipsoids are then splatted to the screen to produce a volume rendered image. There are several problems with this approach. First of all, they introduce many new points, resampling the mesh, increasing the computation time. Furthermore finding an accurate interpolated value of a new point can be difficult and expensive. Also, the practical computation of a Poisson sphere distribution is very time consuming.

Our algorithm, although similar to Mao's is based on a model that reconstructs the underlying function of the grid and then projects it to the screen. In order to reconstruct the underlying function we use a Gaussian reconstruction kernel. In computational domain the samples (grid points) are equally spaced (regular sampling of the data field). Therefore, we have spherical reconstruction kernels in computational domain, but not in the physical domain. The mapping into physical space is non-linear and therefore the spherical reconstruction kernels from computational space are mapped to what we call "bean-shapes". These are usually not ellipsoids and are therefore difficult to handle.

Our algorithm is similar to Mao's, but it differs in two critical areas. First of all, we avoid the costly Poisson disk distribution by calculating ellipsoidal footprints only at the grid data points.

This avoids the resampling of the data volume and ensures that all critical nodes contribute to the final image. The resampling in Mao's paper (as with all resampling) is subject to blurring and aliasing. Unless the sampling rate is adequately high, important features can be missed. This is true even if we guarantee at least one sample per computation cell, due to possible high gradients across the zone. Avoiding the resampling ensures that we don't miss important detail. Focusing solely on the grid data points also allows us to calculate the Jacobian (local curvature) more accurately and efficiently than Mao's trilinear interpolation. This is readily apparent in our image quality. Finally, we use the textured splats [3] to render the ellipsoids efficiently with common hardware. We show the necessary extensions needed for that algorithm to project the ellipsoids. In particular, we show how this can be accomplished with a simple scaling transform and a rotational transform.

3. Our Algorithm

We explained in Section 2 that the correct reconstruction kernels would be warped spheres or what we call bean-shapes. The computation of these bean-shapes is quite complex. The calculation of the screen extent of these shapes is also very complex.

The basic idea of our algorithm is to approximate these bean-shapes with ellipsoids. We do this by a linear approximation of the local shape of the mapping function from computational to physical space. In the following we denote points in computational space by c and points in physical space by p. Letting w denote the warping, we can thus write $w(c) = p$.

3.1 Preprocessing

In computational space all the sample data points use the same spherical reconstruction kernel. Using a spherical, normalized Gaussian reconstruction kernel of 'strength' s around a data point at c_0, we can express it's point distribution τ at a point c as:

$$\tau_{c_0}(c) = \frac{1}{\pi\sqrt{\pi}s^3} e^{-E(c-c_0)} . \tag{1}$$

The term

$$E(c-c_0) = 1/s^2 (c-c_0)^T (c-c_0)$$

describes a sphere, centered at c_0 with radius s (s basically indicates how 'big' our radius of influence is). This description is based in computational space. As we mentioned before, if we map this sphere $E(c-c_0)$ to physical space, we will end up with a bean-shape distribution. Instead of using the mapping w we only use a linear approximation of this mapping. Expressing the mapping w in a Taylor series expansion, we find:

$$p = w(c) = w(c_0) + J_w(c_0)(c-c_0) + O\left((c-c_0)^2 \right) ,$$

In other words, we can approximate w locally with a linear mapping around our point center c_0 using the Jacobian J_w of w at c_0. If we assume, that the mapping is (locally) invertible, a safe assumption for practical applications, we find (using, that $w(c_0) = p_0$):

$$J_w^{-1}(c_0)(p-p_0) \approx (c-c_0) .$$

From this we conclude that we can approximate the warped distribution function τ in Equation 1 by an ellipsoid in the following way:

$$\tau_{p_0}(p) \approx \frac{1}{\pi\sqrt{\pi}s^3} e^{-E_w(p-p_0)},$$

where

$$E_w(p-p_0) = E\left(J_w^{-1}(c_0)(p-p_0)\right) \approx E(c-c_0)$$
$$E(p-p_0) = \frac{1}{s^2}\left(J_w^{-1}(c_0)(p-p_0)\right)^T\left(J_w^{-1}(c_0)(p-p_0)\right) = (p-p_0)^T\left(\frac{1}{s^2}\left(J_w^{-1}(c_0)\right)^T J_w^{-1}(c_0)\right)(p-p_0).$$

The computation of this ellipsoidal extent needs to be done only once as a pre-process. Therefore we deduce the following pre-processing step:

```
1. For each data point $c_0$
2.         Compute $J_0$
3.         Compute $J_0^{-1}$
4.         Compute the quadratic form matrix
                      $Q = \frac{1}{s^2}\left(J_0^{-1}\right)^T J_0^{-1}$
```

FIGURE 2. The preprocessing step.

3.2 The rendering pipeline

Once the ellipsoidal kernels are computed, the rendering of the splats is similar to Mao's algorithm and rather straightforward (see Fig. 3). First we need to orient the ellipsoid in the viewing coordinate system (Line 4 and 5). Then we compute its integral along the *z* direction. It turns out that its integral can be approximated by a two dimensional ellipse (Line 6). Proper integration also produces an adjustment of the opacity component (Line 7). Next we compute the orientation and extent of the major axes of the 2D ellipse. Finally, we can scale a 2D (spherical) lookup table in the major directions and then rotate it appropriately (Line 9 through 11). This idea is very useful for our implementation using textured splats [3]. Here the lookup table is a 2D textured polygon (quadrilateral). This quadrilateral is scaled in the *x* and the *y* direction, rotated and finally projected onto the image plane.

4. Implementation and Results

As we have mentioned in Section 3.2 our implementation is based on textured splats as described in [3]. Since a Gaussian kernel has infinite extend we used the approximation to the Gaussian as proposed by Crawfis and Max [3].

We implemented this algorithm in OpenGL on an SGI Crimson RealityEngine with a 100 MHz R4000 processor. We tested our algorithm on 3 different data sets. The bluntfin data set (32x32x40) is displayed in Fig. 5 and we focus in on one section in Fig. 6. The delta wing data set

```
1. **For each** view
2.         **Sort** the points by distance from viewer at the origin.
3.         **For each** point in back to front order:
4.                 Transform center to $(x_0, y_0, z_0)$ by viewing matrix.
5.                 Transform $Q$ by rotation matrix $R$ only:

                   $Q' = \begin{bmatrix} A' & D' & E' \\ D' & B' & F' \\ E' & F' & C' \end{bmatrix} = RQR^T$

6.                 Project ellipsoid to ellipse equation $\begin{bmatrix} p & q \\ q & r \end{bmatrix}$ in image plane, then

                   $p = A' - \frac{E'^2}{I}$    $r = B' - \frac{F'^2}{I}$    $q = D' - \frac{E'F'}{I}$

7.                 Adopt the opacity $\alpha = \frac{1}{\sqrt{C'}}\alpha_{\text{node}}$
8.                 Find the extent $(s_x, s_y)$ and the rotation angle $\theta$ of the
                       major axis of the ellipse
9.                 scale the texture square by $s_x$ and $s_y$
10.                rotate about $z$ axis by $\theta$
11.                translate to $(x_0, y_0, z_0)$
```

FIGURE 3. The Rendering algorithm.

(66x28x15) is shown in Fig. 7. Finally the jet engine data set (250x180x9) is displayed in Fig. 8. For all data set we have chosen the overlap of the reconstruction kernels in computational space to be 0.6 (i.e. $s=1.6$). This overlap has been found to be optimal by others [3].

The timings for these images are summarized in Table 1. (user time and wall clock time are not synchronized, we are currently investigating this fact)

TABLE 1. Rendering times (in seconds)

	Preprocessing	**actual rendering**	**Mao**
Bluntfin	1.73	2.9	13.9 - 18.4
Delta Wing	1.2	2.0	?
Jet Engine	17.3	28.8	?

Note, that Mao's timings were based on a R4400 machine running at 200MHz. Her image size is 300 by 300, while we are rendering 600 by 600 images. It is expected therefore that our performance would be greatly improved on this class of machine. Our image quality also seems to be much improved, but a direct comparison is difficult. Since Mao's algorithm is a resampling, more sample points could greatly improve their images, trading rendering time for image quality.

5. Conclusions and Future Goals

We have presented an algorithm for fast splatting of curvilinear grids. It is based on a linear local approximation of the bean-shaped reconstruction kernels of the warped grid by ellipsoids. The screen extent of these ellipsoids are 2D ellipses. We compute the extent and orientation of these extents and scale and rotate a textured splat accordingly. This splat can then be rapidly drawn by special graphics hardware. The image quality of the rendered images is comparable or superior to other algorithms, with substantially faster rendering times.

Since perspective viewing gives a better insight into the third dimension of the data set, we are working right now on a perspective implementation of our algorithm. We also wish to extend our algorithm to unstructured grids. The difficulty here will be to accurately measure the local warping of the grid.

6. Acknowledgments

We thank the visualization group at the Lawrence Livermore National Laboratory and especially Rebecca Springmeyer for their encouragement and support. We also thank Gary Kerbel of the Lawrence Livermore National Laboratory for many stimulating discussions. We thank Dave Reed for help with the data sets and useful programming hints. We also thank Wayne Carlson and the Advanced Computing Center for the Arts and Design for the use of their computing facilities. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract number W-7405-ENG-48, with specific support from an internal LDRD grant.

7. References

[1] Akeley K., “RealityEngine Graphics”, *Computer Graphics (SIGGRAPH ‘93 Proceedings)*, pp. 109--116, August 1993.

[2] Cabral B., Cam N., Foran J., “Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware”, *Proceedings of the 1994 Symposium on Volume Visualization*, pp. 91-98, October 1996.

[3] Crawfis R., Max N., “Textured Splats for 3D Scalar and Vector Field Visualization”, *Proceedings of IEEE Conference on Visualization 1993*, Edited by Nielson and Bergeron, San Jose: IEEECS Press, pp. 261-266, October 1993.

[4] Garrity M.P., “Raytracing Irregular Volume Data”, *Computer Graphics (San Diego Workshop on Volume Visualization),* vol. 24(5), pp. 35-41, November 1990.

[5] Mao X., Hong L., Kaufman A., “Splatting of curvilinear Grids”, *Proceedings of IEEE Conference on Visualization 1995*, pp. 61-68, November 1995.

[6] Max N., Hanrahan P., Crawfis R., “Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions”, *Computer Graphics (San Diego Workshop on Volume Visualization),* vol. 24(5), pp. 27-33, November 1990.

[7] Shirley P., Tuchman A., “A Polygonal Approximation to Direct Scalar Volume Rendering”, *Computer Graphics (San Diego Workshop on Volume Visualization),* vol. 24(5), pp. 63-70, November 1990.

[8] Speray D., Kennon S., “Volume Probes: Interactive Data Exploration on Arbitrary Grids”, *Computer Graphics (San Diego Workshop on Volume Visualization),* vol. 24(5), pp. 5-12, November 1990.

[9] Westover L., “Footprint Evaluation for Volume Rendering”, *Computer Graphics (SIGGRAPH ‘90 Proceedings)*, vol. 24, pp. 367--376, August 1990.

[10] Yagel R., Ebert D., Scott J., Kurzion Y., “Grouping Volume Renderers for Enhanced Visualization in Computational Fliud Dynamics,” *IEEE Transactions on Visualization and Computer Graphics*, ITVCG 1(2), July 1995.

[11] Yagel R., Reed D., Law A., Shih P., Shareef N., “Hardware Assisted Volume Rendering of Unstructured Grids by Incremental Slicing”, *IEEE 1996 Symposium on Volume Visualization*, pp. 55-62, October 1996.

[12] Zienkiewicz O.C., Taylor R.L., *The finite Element Method*, Fourth Ed., vol. 1, McGraw-Hill, 1989.

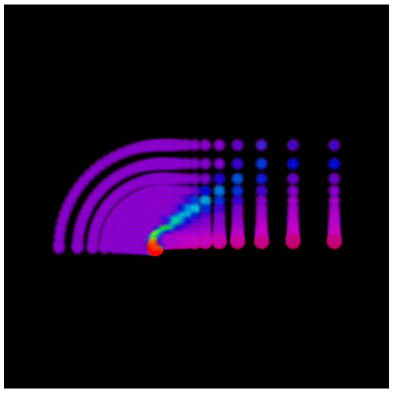

FIGURE 4. Bluntfin data set rendered using spherical splats

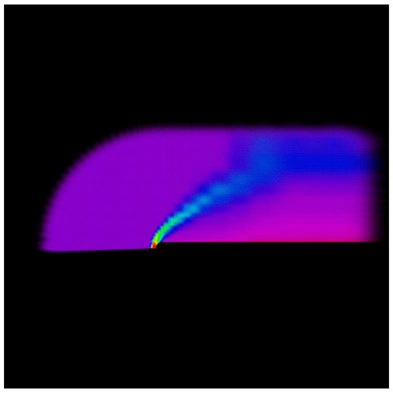

FIGURE 5. The bluntfin data set

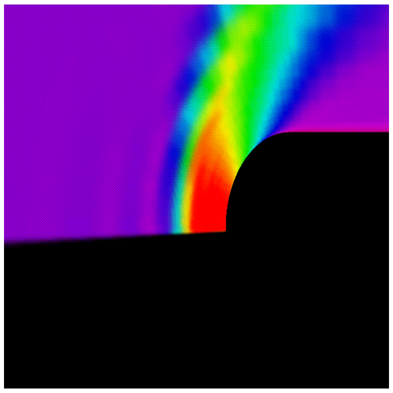

FIGURE 6. Focus on the bluntfin data.

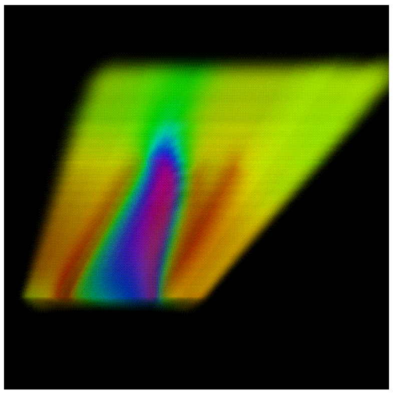

FIGURE 7. The delta wing data set

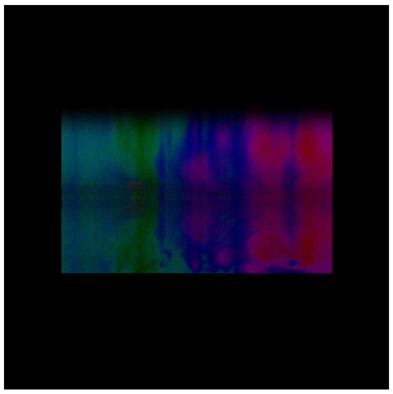

FIGURE 8. The Jet engine data set.