Hybrid Forward Resampling and Volume Rendering

Xiaoru Yuan, Minh X. Nguyen, Hui Xu, and Baoquan Chen

Department of Computer Science and Engineering
University of Minnesota at Twin Cities
http://www.cs.umn.edu/∼baoquan
Email:{xyuan, mnguyen, hxu, baoquan}@cs.umn.edu

Abstract

The transforming and rendering of discrete objects, such as traditional images (with or without depths) and volumes, can be considered as resampling problem – objects are reconstructed, transformed, filtered, and finally sampled on the screen grids. In resampling practices, discrete samples (pixels, voxels) can be considered either as infinitesimal sample points (simply called points*) or samples of a certain size (*splats*). Resampling can also be done either forwards or backwards in either the source domain or the target domain. In this paper, we present a framework that features hybrid forward resampling for discrete rendering. Specifically, we apply this framework to enhance volumetric splatting. In this approach, minified voxels are taken simply as points filtered in screen space; while magnified voxels are taken as spherical splats. In addition, we develop two techniques for performing accurate and efficient perspective splatting. The first one is to efficiently compute the 2D elliptical geometry of perspectively projected splats; the second one is to achieve accurate perspective reconstruction filter. The results of our experiments demonstrate both the effectiveness of antialiasing and the efficiency of rendering using this approach.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation-Display Algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

Resampling, an operation usually related to image processing, is the process of transforming a discretely sampled image from one coordinate system to another. This definition can be extended to the transformation of any other discrete object, such as depth images and volumes. With perspective transformation, resampling amounts to 3D rendering; the issue becomes how to properly resample the discrete objects using the regular screen grids after projection. Conceptually, an ideal discrete resampling operation consists of four basic steps: reconstruction, transformation, prefiltering, and sampling [5, 19]. In practice, reconstruction and prefiltering filters are usually combined together with embedded transformation. Additional operations, such as visibility testing and lighting estimation, are then conducted for 3D rendering. The fundamental issue of 3D rendering is in which domain (source or target) to evaluate these operations and how to accumulate each contribution so that the original discrete object can be faithfully represented on the 2D screen

grids. In this paper, we offer a hybrid resampling framework by converting discrete rendering into a resampling problem, to ensure effective antialiasing with efficiency. Our hybrid method features forward processing. When samples are minified after transformation, they are taken simply as points, and filtering is done in the target domain; while samples are magnified, they are taken as splats which are reconstructed before projection to the screen. This framework can be generally applied to the rendering and transformation of discrete objects, such as texture mapping, 3D image warping, and volume rendering.

The volumetric splatting introduced by Westover [18] takes an object order approach in contrast to volumetric ray casting. In splatting, a volume is represented as an array of reconstructed voxel kernels; each of these kernels is first classified to have color and opacity based on the transfer function applied and is then projected to the image plane, yielding a footprint or splat on the screen, and finally composited with the framebuffer. Splatting has a number of fea-

tures that make it attractive, mainly as follows: (1) its object processing order makes it feasible to store and access volumes in an efficient fashion, especially for sparse volumes; (2) splats on screen and their reconstruction filters can be precomputed and stored in lookup tables so that only table lookup is needed during the run-time rendering. Recently, image-based [14, 13] and point-based [12, 22] systems have gained in popularity; without explicit connectivity available, splatting becomes a suitable rendering method in these systems. Forward splatting has also been used in texture mapping [1].

In recent years, a number of methods have been proposed to improve the original splatting method, including methods for delivering correct perspective rendering [11, 10], antialiasing [16, 21], and efficiency [6]. Aliasing occurs when voxels are minified (i.e., their splats project to less than one pixel size on screen). The approach taken by Swan et al. [16] inflates the splats so that they cover at least one pixel. While they perform this inflation uniformly Zwicker et al. [21] have proposed techniques to non-uniformly inflate splats to accommodate anisotropic filtering. The original splatting methods of Westover [18] do not work well for perspective projection because the projections of voxels on screen are view-dependent, therefore straightforwardly looking up pre-computed, view-independent reconstruction filter leads to artifacts. Mueller and Yagel [11] have proposed a ray-driven approach for perspective volume rendering, in which a spherical voxel is replaced with a disk perpendicular to the ray passing through the center of the voxel; the disk is mapped with the pre-computed reconstruction filter. For every pixel on screen covered by the disk, a ray is shot to the disk and the intersection is used as the index to the filter. Nevertheless, the filter index that is computed is still inaccurate due to the replacement of spheres with disks. Mueller et al. [10] have also proposed an image-aligned sheet buffer approach. In this method, voxels are cut into thin slabs parallel to the viewing plane; each slab of a voxel is reconstructed separately by looking up pre-computed, view-independent filter. Although the reconstruction is inaccurate for each slab, however, when the slab is thin enough the error becomes ignorable. Voxel slabs within the same slab are summed together into a sheet buffer, which is then composited to the framebuffer. This method also eliminates the popping effect presented in the original axis-aligned sheet buffer approach [18].

In this paper, we employ a hybridity of points and splats as rendering primitives. While regular splatting is performed for magnified voxels, for minified voxels, rather than inflating their splats, we simply replace them with points. To compute their contributions to the final image, we place a circular filter on each pixel in-screen to filter the incoming points. Filtering points in the screen space is much simpler than splatting them. Therefore, our method features a hybrid approach using both forward point projection and forward splat projection. Within this framework, we have designed and implemented a number of techniques for accurate perspective splatting calculation. First, we develop methods for

more efficiently estimating the elliptical shape of 2D screen splats. Second, we present techniques to achieve accurate perspective reconstruction filter. The key is to generate a correct index to the lookup table of the view-independent reconstruction filter. The results of our experiments demonstrate both effective antialiasing and efficient rendering.

2. Hybrid Forward Resampling Framework

The key task in resampling is for each screen pixel to determine the contributing input discrete samples and to convolve the samples with a proper filter to obtain the final pixel color. This filter is usually defined with a circular footprint with a Gaussian profile and can be applied to either screen pixels or input samples. The processing can be done in the source domain (object order) through forward projection, or in the destination domain (screen order) through backward projection. This amounts to four practical approaches: forward point projection (forward projection; filtering on screen); forward footprint projection (forward projection; filtering on object); backward point projection (backward projection; filtering on object); and backward footprint projection (backward projection; filtering on screen). In the past, all methods have been practiced. A detailed survey can be found in [2, 3]. Here we present a hybrid resampling framework that combines forward point projection and forward footprint projection.

For the forward point projection method, each input sample is taken as a point and projected into screen space. A circular filter is placed at each pixel for prefiltering the projected samples [4, 13]. For minified samples, where aliasing usually arises, point projection effortlessly performs quality antialiasing; however, holes may appear in magnified samples after projection because reconstruction is not performed. For the forward footprint projection method, each input sample is considered a circle (or a sphere for voxels) forwardly projected to the screen, creating a screen-space conic. The conic is then scan-converted and the energy of the sample is convolved with a filter on screen. This effectively 'splats' the energy of the sample to the pixel within its projected footprint and thus effectively solves the problem for the magnification situation. This method, however, presents a potential problem for the minification situation, however, because the entire conic may actually fall between the pixels, therefore making no contribution to any of them. Computing and evaluating a conic or even the approximated ellipse shape as well as convolving texel energy are expensive operations (especially in the minification situation).

Therefore, our hybrid forward resampling framework performs forward point projection for minified samples and forward footprint projection for magnified samples. The general concept, based on volume rendering, is illustrated in Figure 1. For magnified voxels (e.g., voxel c in Figure 1), we perform regular splatting. However, for minified voxels (e.g., voxel a and b), we take them as points and directly project them to the image plane; their contributions are then

computed by circular filters placed at pixels. This framework takes advantage of both points and splats: points are more efficient for the minification region, while splats are advantageous for the magnification region.
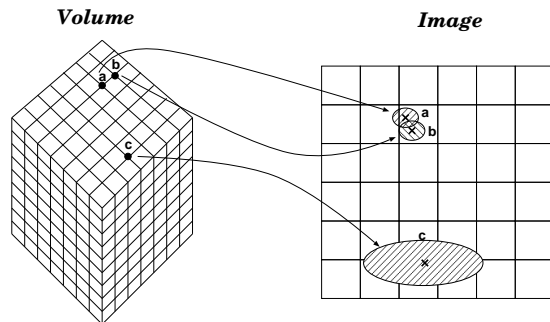

Figure 1: Voxels are taken as either splats or points depending on their projection size on screen (e.g., points: voxels a and b, splat: voxel c).

3. Hybrid Forward Image Warping

We first introduce the application of hybrid forward resampling to improve existing forward texture mapping methods. Forward point projection has previously been employed for texture mapping. Ghazanfarpour et al. [4] have proposed a technique that takes every texel as a point and projects it into screen space. A filter is placed at each pixel for filtering the projected texels. This amounts to effective antialiasing in the minification region, and can be almost effortlessly performed (see Figure 5a). Compared to forward splatting where splats have to be estimated (by ellipses) and scan-converted, here no splat is computed and approximated. Holes, however, may appear for magnification regions (also see Figure 5a), one solution for which is to supersample in texture space along the magnification axis and forwardly project subtexels (as points) into screen space. But because this amounts to an expensive approach, splatting becomes more suitable here. Because the reconstruction filter kernel (usually a Guassian kernel) of a splat can be pre-generated and stored in a table, computing its contribution to several pixels can be done in one scan-conversion path. Figure 5b demonstrates that our hybrid forward method produces higher quality antialiasing than does traditional backward texture mapping with bilinear interpolation (Figure 5c).

Another application of the hybrid resampling framework is in the warping of depth images. Traditional 3D image warping is done using McMillan's warping equation [8], which allows one to take advantage of the regular structure of images to perform incremental transformation. McMillan's method takes a forward point projection approach — that is, after projection, a sample's color is written to its nearest pixel, overwriting whatever color the pixel has. Likewise, two problems exist with this approach: (1) holes in the

magnification region and (2) aliasing in the minification region. A few approaches have been proposed for solving this problem, such as connecting the samples of the reference image into a polygonal mesh [7] or using splats for each sample [14]. Both methods are expensive for the minified region, in which either a triangle or splat projects to the subpixel area. WarpEngine [13] connects samples, but avoids the overhead of general polygonal rendering by generating subsamples before (or after) projection. Because subsamples have to be dense enough to completely avoid incorrect holes, this amounts to an expensive solution in the magnification region.

Here we apply our hybrid forward method to image warping. For magnified samples, we employ the splatting method. The splat size of a source sample can be estimated based on its adjacent samples (e.g., their maximum distance to the current sample). For minified samples, we employ points; it is important that we do not overwrite visible samples projecting to the same pixel. Rather, we need to accumulate each sample's contribution through screen space filtering for antialiasing. Visibility is calculated using depth, normal, as well as detected boundary discontinuity information [13]; samples within the same boundary are assumed to be on the same surface. Finally, for both magnified and minified cases, we detect backfacing samples by checking their normals. Figure 6 compares the two warped images (Figures 6b and 6d) and their zoom-ins (Figure 6c and 6e) generated from our hybrid method and McMillian's original warping algorithm, respectively. This clearly shows that higher quality can be produced using the hybrid method.

4. Hybrid Forward Volume Rendering

Similarly, we apply our hybrid resampling framework to forward volume rendering (i.e., volumetric splatting). Although the framework is general enough to be applied to enhance various existing methods, we here discuss one specific implementation.

4.1. Volume Processing Order

For volume rendering, care must be taken to ensure correct compositing because of the translucent nature of voxels. Splatting methods have been designed that feature different volume processing orders: (1) always performing 'over' (compositing) operation on voxels [17], which processes voxels in depth-sorted order and then composites voxel's splat one over another; (2) slice-by-slice processing order with sheet buffer, which accumulates voxels's colors within each slice into a sheet buffer using summation and then composites the entire sheet to the intermediate framebuffer [18]; and (3) image-aligned sheet buffer order, in which sheets are always parallel to the image plane [10]. Each method has its advantages and disadvantages. For example, the always compositing method amounts to an efficient approach, but can

lead to color bleeding artifacts as each voxel is processed independently, thus ignoring the fact that adjacent splats overlap. The axis-aligned sheet buffer approach solves the bleeding problem, but results in popping artifacts — that is, when the viewpoint shifts at points that require switching to a different axis-aligned sheet buffer, the summation and compositing order changes between voxels of the same slice, hence resulting in changes in the shade of colors. The image-aligned sheet buffer approach further solves the popping effect because The sheet's orientation is incrementally updated rather than switched at some discrete points. However, even with optimizations [6], this approach has added several degrees of complexity to the original splatting approaches because every sheet defines a thin slab in 3D space that cut a voxel sphere (splats) into several parts; different parts are then summed to different sheets. Theoretically, our hybrid framework can be applied to all these methods. For example, for a image-aligned sheet buffer approach, when processing distant sheets in minification region, voxels can be represented using simple points, with no need to cut through splats that are eventually projected to subpixel size. Filtering is then done in sheet buffer on points.
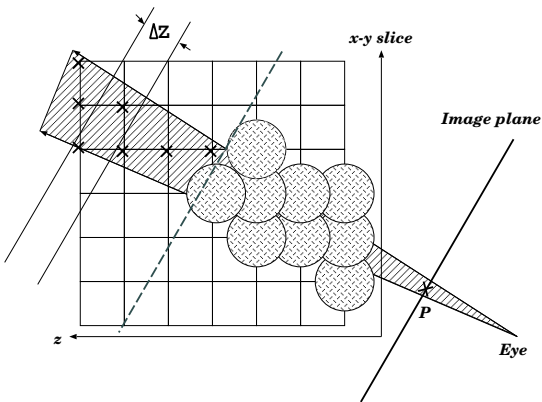

Figure 2: Volume processing order.

In this paper we take the original composite-every-sample approach [17], as for most cases bleeding effects are not observable and this approach can be implemented efficiently. The key to this method is processing voxels in strict depth order. This ordering can be implemented by first transforming voxels to the image space and then sorting them based on their depth values as done in [10]. Another sorting method that can be used is described by Swan [15]; here, once the order is determined, voxels are processed either as splats or points based on their projection size. The dotted line in Figure 2 indicates the boundary between splats and points. While we always perform 'over' operation (compositing) for splats, we choose not to do so for points. The reason is that points do not fully cover a pixel, and therefore two points may be in fact side-by-side rather than occluding one another. Therefore, we instead first sum points within a certain depth range
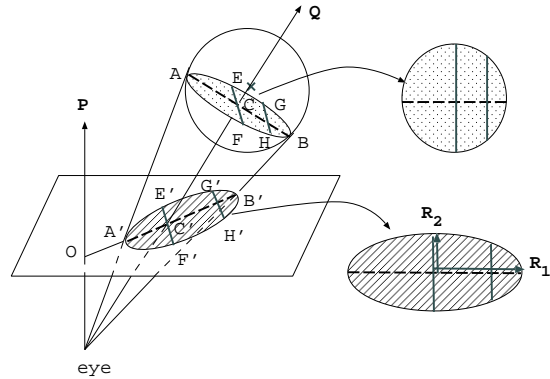

Figure 3: Elliptical splat geometry.

(e.g., 1 unit) and then composite the result to the framebuffer. This summation is done using the circular filter at the pixel.

4.2. Elliptical Splat Geometry and Perspective Reconstruction Filter

There are two important issues in implementing perspective splatting. The first is estimating the accurate geometry of 3D splats' projection on screen; the second is computing the correct reconstruction filter of perspectively projected splats. We have developed techniques for addressing both issues.

In Figure 3, the eye rays tangent to the sphere form a cone; the 2D projection of the sphere on the horizontal image plane is then the intersection between the cone and the image plane. By definition, this intersection is an ellipse. As an ellipse is defined by its major and minor axes (R_1 and R_2, simply call them major axes thereafter), once we have computed these two axes, we can draw the ellipse. Here we present an efficient way of computing major axes. We have discovered that one of the two axes is the intersection between the plane formed by vector $\vec{P}$, the eye ray vector perpendicular to the viewing plane, and vector $\vec{Q}$, the eye vector passing through the voxel center. The proof and details about computing major axes appear in Appendix A.

In splatting, both volume reconstruction and integration are performed at each voxel by projecting and compositing weighted reconstruction kernels. A reconstruction kernel is centered at each voxel, and its contribution is accumulated onto an ellipse on the image plane. This projection (or line integration) contains the integration of the kernel along a ray from infinity to the viewing plane. The splat is typically integrated by parallel rays and is pre-computed and resampled into what is called a footprint table (see right diagram of Figure 4). This table is then re-sampled into the framebuffer for each splat. For orthogonal projection and radially symmetric interpolation kernels, a single footprint can be precomputed and used for all views. The table is indexed by distance to the circle center (e.g., i in the right diagram of
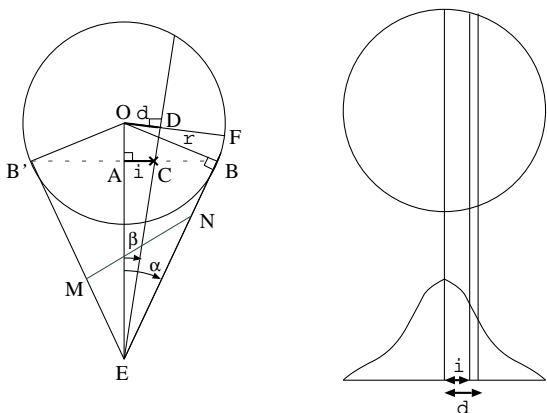
Figure 4: Perspective splatting reconstruction filter.

Figure 4). This is not the case with perspective projections or non-uniform volumes, however, which require that the splat size be changed and that the shape of the distorted reconstruction kernel be non-spherical. Mueller's image-aligned sheet buffer approach [10] helps improve the integration accuracy by slicing the 3D reconstruction kernel into thin slabs. It is perspectively accurate between slabs, however, within each slab all kernels are orthographically projected. While this significantly improves image quality, it requires much more compositing and several footprint sections per voxel to be scan-converted.

Here our goal is to perform accurate reconstruction for each voxel only once and to make it accurate for perspective projection. We still aim to use table lookup for efficient computation. The key is to generate correct index to the lookup table. Illustrated in Figure 4 (left diagram) in 2D, the sphere has radius r. After we have obtained the screen ellipse (line MN) for each voxel using the aforementioned method, we then use an affine transformation to transform it back to the circle indicated by the line BB'; for each pixel within the ellipse, we obtain index i in the circle ($i = \frac{AC}{AB}$). This can be done by affine transformation of the ellipse to the circle [21]. As we can see from Figure 4, the accurate index should be d, which is the distance between the sphere center to the ray. Here we derive equation for computing d from i:

$AB = r\cos(\alpha)$.

$AE = OE - AO = \frac{r}{\sin(\alpha)} - r\sin(\alpha) = \frac{r\cos^2(\alpha)}{\sin(\alpha)}$.

$OD = OE\sin(\beta)$,

where $\tan(\beta) = \frac{iAB}{AE}$.

By replacing AB and AE,

$OD = OE\sin(\arctan(i\tan(\alpha)))$.

Therefore, The correct index is then

$d = \frac{OD}{OF} = \frac{OE}{OF}\sin(\arctan(i\tan(\alpha)) = \frac{\sin(\arctan(i\tan(\alpha))}{\tan(\alpha)}$.

From the above derivation, the correct index d is deter-

mined by the half cone angle α of the sphere. We can construct a 2D table to speed up the computation of the above equation. The two indices to the table are i and α. The number of entries for i is determined by the size of the precomputed reconstruction kernel table. To maintain a modest number of entries for α, we create α entries only for certain ranges of angles that appear in common situations (less than $15°$ are usually sufficient). For angles outside this range, we simply evaluate the equation for computing d. Assuming that these angles represent rare cases, the effort is spent only on a small percentage of voxels.

5. Implementation and Results

We have implemented our hybrid volume rendering on a Linux 1.7GHz Pentium 4 PC with 2GB memory. Figure 7 and 8 present images generated from different volume data. We have used five data for testing; while the 3D Checker Box volume is an artificial data set created by ourselves, the other four data are public domain data, including Aneurism ($256 \times 256 \times 256$, rotational b-plane x-ray scan of the arteries of the right half of a human head), Lobster ($301 \times 324 \times 56$, CT scan of a lobster contained in a block of resin), Engine ($256 \times 256 \times 128$, CT scan of two cylinders of an engine block), and Skull ($256 \times 256 \times 256$, rotational b-plane x-ray scan of phantom of a human skull). All images generated have a resolution of 400×400.

Figure 7 demonstrates that our method is free of the popping effect. Figures 7a and 7b present two images with view directions that cross the $45°$ viewing angle. The shading is consistent between the two images. The checkerboard images also demonstrate the effective antialiasing of our method, which is further demonstrated in the animation sequence (CheckerBox.mpg). (All animation files mentioned in this section can be accessed at *http://www.cs.umn.edu/~baoquan/vg03/*.) Figure 8a depicts a rendering of Aneurism with splats marked out with red color while points with regular color. The accompanying movie shows this dynamically in motion (AneurismZoom-Color.mpg). Figures 8b-8f provide more examples of volume rendering. Figures 8b and 8c present two images of Aneurism with different viewing distances. These two images are extracted from an animation sequence (Aneurism-Zoom.mpg). Note that the scattered points in the images are noise from the original data; they are not aliasing artifacts. This can be best evaluated from animation (Aneurism.mpg); these points' presence in the animation are consistent. Figures 8d and 8e are images of Engine and Skull. Figure 8f depicts a lobster with shell and meat classified to red and white, respectively.

Our next experiment evaluates the efficiency of our algorithm. Although our implementation has not been optimized, our preliminary results show encouraging speed. Figure 9 tabulates timings of rendering different volume data. The number of non-transparent voxels' is determined by the

specified transfer function for each volume data and indicates the number of voxels being processed for each frame. In the three columns regarding timing, the percentages in brackets indicate the percentage of non-transparent voxels that are processed as splats. The timings for "all splats" represent the average time for frames that all voxels are processed as splats (close view); correspondingly, "all points" timings measure for distant view that all voxels are taken as points. The timings labeled as "mixed" are for those frames that have a mixture of both splats and points. For points, we have implemented occlusion culling since it is straightforward to do, but it has yet been done for splats. Comparing the "all splats" and "all points" columns clearly shows that splats are several times more expensive than points. The processing time for splats depends on the size of their coverage, as a larger splat takes more time for scan conversion and compositing; therefore, the time is not simply proportional to the number of splats. The mixed column represents the common situation in which voxels are either magnified or minified.

6. Conclusions and Future Work

We have presented a hybrid forward resampling framework and its application to various discrete rendering problems, such as texture mapping, 3D image warping, as well as volume rendering, with both effective and efficient antialiasing. Specifically, for volume splatting we have further developed techniques for performing perspectively correct splatting with high quality and efficiency. As part of an on-going project, we are working on further improving the overall performance by implementing occlusion culling for splatting [10, 20]. The approach is to employ an occlusion map to accumulate opacities at run-time as the renderer processes in front-to-back order; a splat can be culled by checking whether the pixels that it projects have reached full opacity. A hierarchical occlusion map can help to quickly check if a certain pixel region has reached full opacity without checking every pixel in it. Other enhancement techniques can be applied to our method, such as Post-classification and shading for eliminating a sometimes blurry effect [9].

Other future work will extend and apply this technique to other problems where spatial sampling is the main operation. One specific area of application that we will look at is point-based rendering in our digital scanning project.

Acknowledgements

References

1. B. Chen, F. Dachille, and A. Kaufman. Forward perspective image warping. *Proc. of IEEE Visualization '99*, pages 89–96, Oct. 1999.

2. B. Chen, F. Dachille, and A. Kaufman. High-quality efficient texture mapping. *IEEE Transaction on Visualization and Computer Graphics (to appear)*, 2003.

3. B. Chen, A. Kaufman, K. Mueller, and A. Varshney. State of the art in data representation for visualization. *IEEE Visualization Tutorial*, Oct. 2002.

4. D. Ghazanfarpour and B. Peroche. A high-quality filtering using forward texture mapping. *Computers and Graphics*, 15(4):569–577, 1991.

5. P. S. Heckbert. Fundamentals of texture mapping and image warping. MS thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, June 1989.

6. J. Huang, K. Mueller, N. Sharef, and R. Crawfi. Fastsplats: Optimized splatting on rectilinear grids. In T. Ertl, B. Hamann, and A. Varshney, editors, *Proceedings Visualization 2000*, pages 219–226. IEEE Computer Society Technical Committee on Computer Graphics, 2000.

7. W. R. Mark, L. McMillan, and G. Bishop. Postrendering 3D warping. In M. Cohen and D. Zeltzer, editors, *1997 Symposium on Interactive 3D Graphics*, pages 7–16. ACM SIGGRAPH, Apr. 1997. ISBN 0-89791-884-3.

8. L. McMillan. *An Image-Based Approach to Three-Dimensional Computer Graphics*. PhD thesis, Department of Computer Science, University of North Carolina at Chapel Hill, 1997.

9. K. Mueller, T. Möller, and R. Crawfis. Splatting without the blur. In D. Ebert, M. Gross, and B. Hamann, editors, *Proceedings of the 1999 IEEE Conference on Visualization (VIS-99)*, pages 363–370, N.Y., Oct. 25–29 1999. ACM Press.

10. K. Mueller, N. Shareef, J. Huang, and R. Crawfis.

High-quality splatting on rectilinear grids with efficient culling of occluded voxels. *IEEE Transactions on Visualization and Computer Graphics*, 5(2), Apr. 1999.

11. K. Mueller and R. Yagel. Fast perspective volume rendering with splatting by utilizing a ray-driven approach. In R. Yagel and G. M. Nielson, editors, *Proceedings of the Conference on Visualization*, pages 65–72, Los Alamitos, Oct. 27–Nov. 1 1996. IEEE.

12. H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In *SIGGRAPH '00 Proc.*, pages 335–342, 2000.

13. V. Popescu, J. Eyles, A. Lastra, J. Steinhurst, N. England, and L. Nyland. The WarpEngine: An architecture for the post-polygonal age. In *SIGGRAPH '00 Proc.*, pages 433–442, 2000.

14. J. W. Shade, S. J. Gortler, L. He, and R. Szeliski. Layered depth images. In *SIGGRAPH '98 Proc.*, pages 231–242, July 1998.

15. E. Swan. *Object-Order Rendering of Discrete Objects*. PhD thesis, Department of Computer and Information Science, the Ohio State University, 1998.

16. J. E. Swan II, K. Mueller, T. Möller, N. Shareef, R. A. Crawfis, and R. Yagel. An anti-aliasing technique for splatting. In *IEEE Visualization '97*, pages 197–204, Nov. 1997.

17. L. Westover. Interactive volume rendering. *Proceedings of the Chapel Hill Workshop on Volume Visualization*, pages 9–16, May 1989.

18. L. A. Westover. *Splatting: A Parallel, Feed-Forward Volume Rendering Algorithm*. PhD thesis, The University of North Carolina at Chapel Hill, Department of Computer Science, July 1991. Technical Report, TR91-029.

19. G. Wolberg. *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, CA, 1990.

20. H. Zhang, D. Manocha, T. Hudson, and K. Hoff. Visibility culling using hierarchical occlusion maps. *SIGGRAPH '97 Proc.*, pages 77 – 88, Aug. 1997.

21. M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Ewa volume splatting. *Proc. of IEEE Visualization '01*, 2001.

22. M. Zwicker, H. Pfister, J. van Baar, and M. us Gross. Surface splatting. *SIGGRAPH '01 Proc.*, pages 371–378, 2001.

Appendix A: Elliptical Splat Geometry Formation

We use these simple facts for the proof: for an ellipse, the major and minor axes are perpendicular to each other and are symmetric axes, in fact the only symmetric axes (except in a circle, which is a special case of the ellipse). Then to find the major or minor axis, we only need to find one symmetric axis; the other one will be orthogonal to it. See Figure 3, the light shaded circle (center C) in the sphere is the intersection between the ray cone and the sphere and is perpendicular to $\vec{Q}$. Vectors $\vec{P}$ and $\vec{Q}$ form a plane that is perpendicular to the viewing plane. AB is the intersection of the light shaded circle and the plane $\vec{P}\vec{Q}$ and is also the diameter of the circle. Line EF is another diameter of the circle that is perpendicular to AB. Therefore, line EF is perpendicular to plane $\vec{P}\vec{Q}$; this means it is parallel to the viewing plane. Consequently, all lines parallel to line EF (e.g., GH) are parallel to the viewing plane. Using similar triangle geometry, these lines' projections on the viewing plane are linearly scaled. Before projection, these lines are symmetric along line AB. Because of the linear scaling of the projection, these lines' projections on the viewing plane remain symmetric along $A'B'$, the projection of line AB. Therefore, line $A'B'$ is a symmetric axis of the ellipse. Hence it is one of the major axes (R_1). Notice that even though R_2 direction is parallelly projected, R_1 direction is not. Therefore, $A'C' \neq C'B'$, the center of the ellipse is not C', but rather the midpoint of $A'B'$. Through these conclusions, computing an ellipse becomes efficient. Here a slightly approximative approach could be used when the angle between $\vec{Q}$ and $\vec{P}$ is small. We can approximate R_1 with vector $\vec{C'B'}$ and the center of the ellipse as C', therefore, R_2 is approximated by vector $\vec{C'F'}$. Under other situations, the midpoint of $A'B'$ should be used.

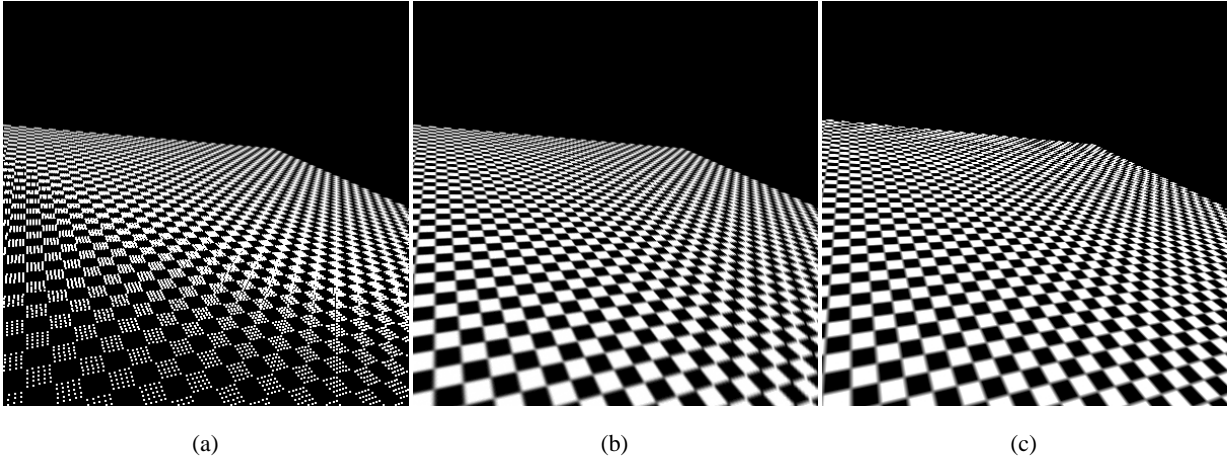(a)   (b)   (c)

Figure 5: Texture mapping using (a) forward point projection, (b) hybrid forward projection, and (c) traditional backward projection with bilinear interpolation.
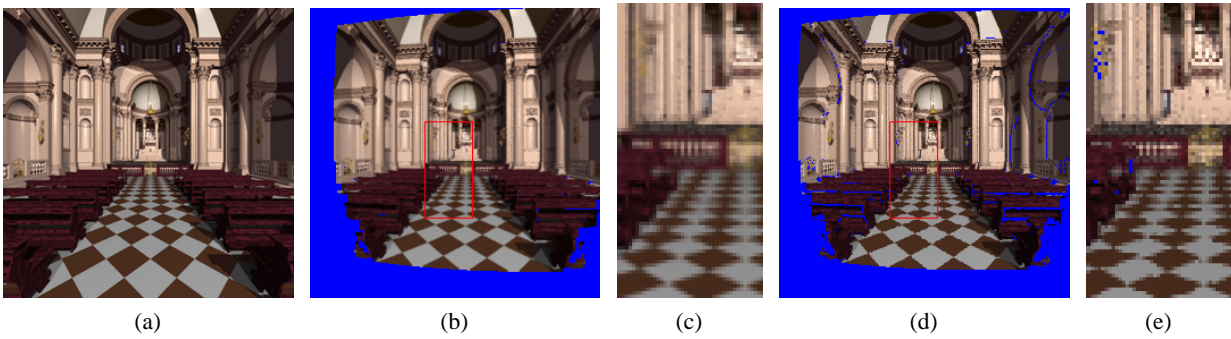

(a)   (b)   (c)   (d)   (e)

Figure 6: 3D image warping: (a) reference image, (b) warped image with antialiasing (our hybrid method), (c) zoom-in of marked region in (b), (d) warped image without antialiasing (McMillian's method), (e) zoom-in of marked region in (d).
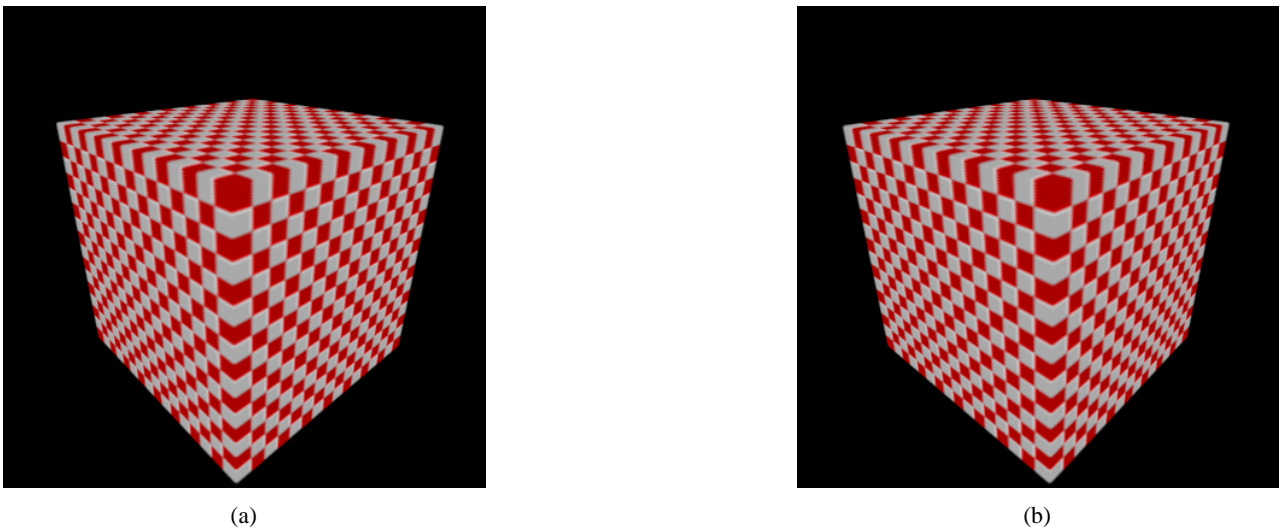

(a)   (b)

Figure 7: 3D Checker Box viewed at different viewing directions crossing the $45°$ angle.
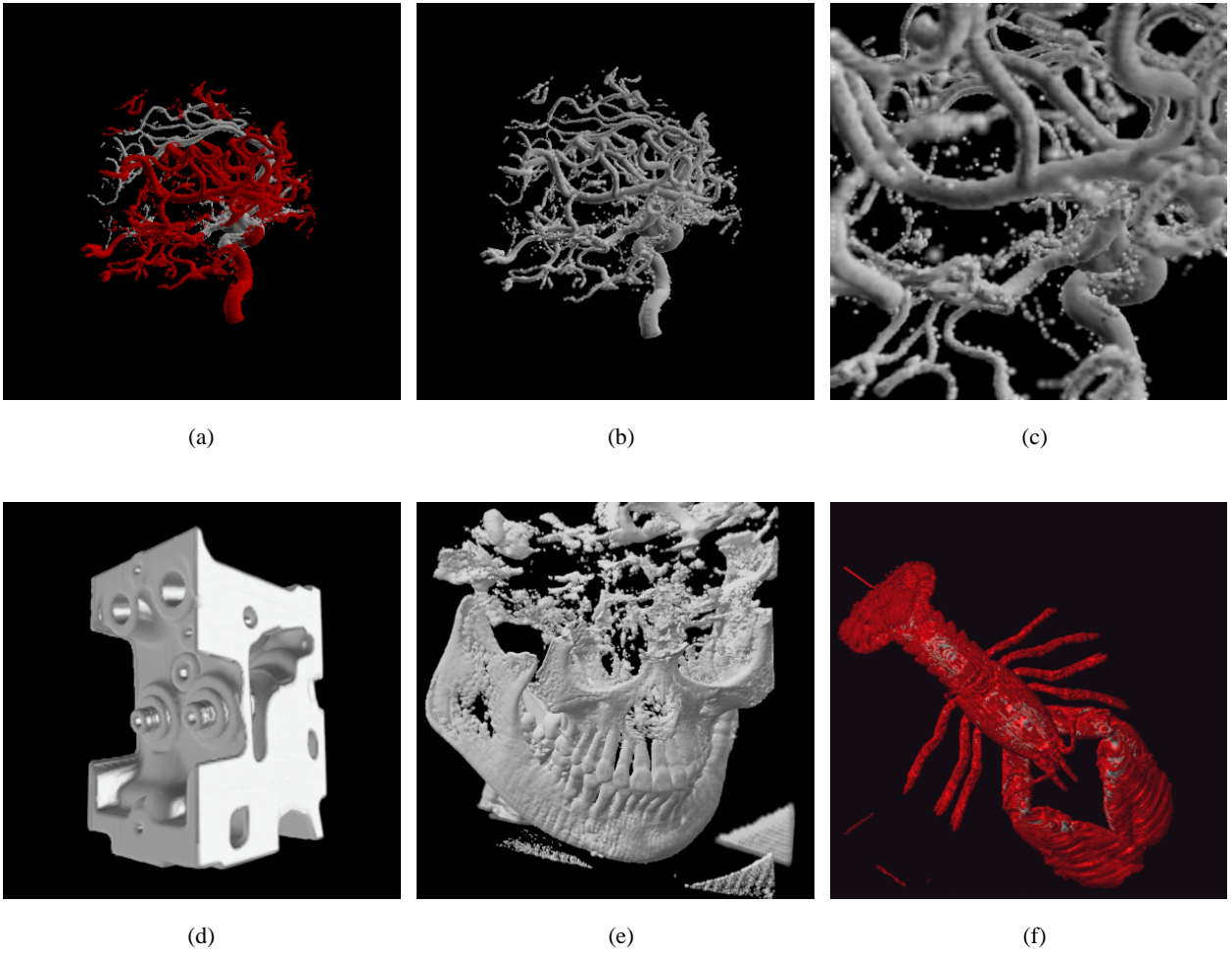
(a) (b) (c)

(d) (e) (f)

Figure 8: Images generated by hybrid method: (a) pseudo-colored Aneurism (red for splats), (b) distant view of Aneurism, (c) close view of Aneurism, (d) Engine, (e) Skull, and (f) Lobster.

Model	Resolution	Number of non-transparent voxels	Timing (second)		
			all spats	mix	all points
Engine	$256 \times 256 \times 128$	1,160,038	15.83 (100%)	9.76 (56.73%)	1.59 (0%)
Lobster	$301 \times 324 \times 56$	190,815	3.18 (100%)	1.97 (48.20%)	1.52(0%)
Aneurism	$256 \times 256 \times 256$	114,695	3.78 (100%)	3.36 (76.38%)	2.67(0%)
Skull	$256 \times 256 \times 256$	775,985	10.39 (100%)	9.29 (88.74%)	4.87 (0%)
Checker Box	$100 \times 100 \times 100$	1,000,000	20.48 (100%)	6.72 (46.60%)	2.78 (0%)

Figure 9: *Timing for rendering different volume data.*