A Framework for Sample-Based Rendering with O-Buffers

Huamin Qu Arie Kaufman Ran Shao Ankush Kumar *

Center for Visual Computing (CVC) and Department of Computer Science
Stony Brook University, Stony Brook, NY 11794-4400

Abstract

We present an innovative modeling and rendering primitive, called the *O-buffer*, for sample-based graphics, such as images, volumes, and points. The 2D or 3D O-buffer is in essence a conventional image or a volume, respectively, except that samples are not restricted to a regular grid. A sample position in the O-buffer is recorded as an offset to the nearest grid point of a regular base grid (hence the name O-buffer). The offset is typically quantized for compact representation and efficient rendering.

The O-buffer emancipates pixels and voxels from the regular grids and can greatly improve the modeling power of images and volumes. It is a semi-regular structure which lends itself to efficient construction and rendering. Image quality can be improved by storing more spatial information with samples and by avoiding multiple resamplings and delaying reconstruction to the final rendering stage. Using O-buffers, more accurate multi-resolution representations can be developed for images and volumes. It can also be exploited to represent and render unstructured primitives, such as points, particles, curvilinear or irregular volumes. The O-buffer is therefore a uniform representation for a variety of graphics primitives and supports mixing them in the same scene. We demonstrate the effectiveness of the O-buffer with hierarchical O-buffers, layered depth O-buffers, and hybrid volume rendering with O-buffers.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Antialiasing; I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types; I.4.10 [Image Processing and Computer Vision]: Image Representation—Hierarchical

Keywords: Sample-based rendering, image-based rendering, hybrid rendering, irregular sampling, hierarchy, offset, frame buffer, layered depth image

1 Introduction

Sample-based primitives, such as images, volumes and points have been used widely in visualization. Compared to traditional geometry-based modeling and rendering methods, sample-based approaches have the following advantages: (1) they can avoid the step of explicitly creating a 3D polygonal model for a scene; (2) they can model objects which are hard to model by surfaces, and (3) rendering time of a sample-based scene is somewhat independent of the scene complexity. Image-based rendering methods use an image (e.g., a texture, a depth image [McMillan 1997; Popescu et al. 2000], a layered depth image (LDI) [Chang et al. 1999; Lischinski and Rappoport 1998; Shade et al. 1998]) as the rendering primitive. All these images are digital, have a finite resolution, and are stored in a regular 2D gridded buffer. Similarly, 3D volumes, whether samples of the real world, simulation of a 3D phenomena, or voxelization of a geometric model [Kaufman 1987], have a finite resolution and are stored in a regular 3D gridded buffer.

The regular sampling pattern and limited resolution of images and volumes result in limited position precision for sample points. This makes conventional images and volumes inefficient in representing precisely high-frequency details (e.g., edges of 3D objects) and in handling multiple samples in one grid point. Also, there is an abundance of irregular samples in computer graphics, such as point clouds from range scanners, curvilinear and irregular grids from scientific computations, and samples from adaptive or jittered sampling for antialiasing. However, regular images and volumes have accuracy limitation in handling these irregular samples.

As computer graphics primitives diversify, one challenge for sample-based rendering is how to organize and mix different sample primitives into one scene and render them in correct visibility order, especially when volume rendering is involved. Typically, geometric models, images, volumes, and point clouds represent some objects in 3D space. Therefore, one may think that a volume representation is an ideal framework to represent all these models. However, when pixels and points are projected back to 3D space, or when geometric models are discretized, they have to be resampled into a limited resolution 3D grid. Some information in the original samples may then be lost.

We believe that the regularity of conventional images and volumes limits their modeling power and might actually be unnecessary. For example, some image representations, such as LDI, are assembled from pictures of the real world. When pixels in these pictures are warped to a new viewpoint, they no longer fall on a regular grid. Usually, a resampling process is needed. However, these representations only serve as intermediate models and are not used directly for display. Therefore, it is unnecessary for pixels in these images to be defined on a regular grid. In the past, the regular pattern of images and volumes was critical for rendering. However, depth images, layered depth images, point samples, and volumes can all be rendered by some kind of 3D warping or splatting algorithm [Grossman and Dally 1998; McMillan 1997; Shade et al. 1998; Westover 1990; Zwicker et al. 2001]. When samples are projected onto an image display plane, they do not fall on the grid points, requiring reconstruction and resampling anyway. Therefore, whether or not samples of the original images and volumes are on a regular grid is not that important.

To face these challenges, we propose a new modeling and rendering primitive, the O-buffer. The O-buffer is essentially a conventional image or volume, except that samples are no longer restricted to grid points on a regular grid and their position is recorded as an offset to the nearest point of a regular base grid. The offset is usually quantized for compact storage and efficient rendering. Therefore, the O-buffer can represent the position of samples more precisely without increasing the sampling rate of the image or the

*e-mail: {huamin|ari|rshao|ankush}@cs.sunysb.edu

IEEE Visualization 2003,
October 19-24, 2003, Seattle, Washington, USA
0-7803-8120-3/03/$17.00 ©2003 IEEE

Proceedings of the 14th IEEE Visualization Conference (VIS'03)
0-7695-2030-8/03 $ 17.00 © 2003 IEEE

volume, can eliminate the need to have a regular grid for them, and can serve as a container for intermediate and final results to avoid multiple resamplings during transformation and to reduce aliasing. In addition, the O-buffer can naturally be utilized to represent and render unstructured primitives, such as points, particle systems, and curvilinear and irregular volumes. Consequently, the O-buffer can be regarded as a uniform representation for a variety of primitives, and can thus support the mixing of several different primitives (e.g., images, points, and volumes) in the same scene.

O-buffers are related to the concepts of subpixels, subvoxels, subgrids, and offset buffers, which have been used before, mostly for antialiasing in geometry-based rendering [Carpenter 1984; Schilling 1991]. The O-buffer, however, is primarily designed for sample-based models. The O-buffer is inspired by the pioneering work of Popescu et al. [2000; 1999], who used an offset buffer to resolve the visibility of pixels and for antialiasing purposes in image-based rendering. However, their work is limited to 2D and used mainly for better reconstruction after image warping. We have used an offset buffer for consecutive warping that resulted in an IBR method with stable frame rates for a real-time system [Qu et al. 2000]. Botsch et al. [2002] attached offset attributes to the samples of their point sampled geometry to guarantee water tight surfaces.

In contrast, the O-buffer is a general 2D or 3D sample-based primitive. We have substantially extended the forms and utilization of O-buffers. The forms of O-buffers now include 3D O-buffers, nonuniform O-buffers, adaptive O-buffers, and hierarchical O-buffers. In addition to using O-buffers to cache 2D intermediate warping results for better reconstruction and consecutive warping, O-buffers are used to provide a uniform framework for various sample primitives, a unified way for hybrid rendering, and more accurate and flexible level-of-detail management for samples. To the best of our knowledge, these representations and functionalities have not been used before.

The main contribution of this paper is the introduction of the O-buffer as a uniform rendering primitive for sample-based entities and a unified way for hybrid rendering. Our O-buffer representation is novel and unique in various ways and has the following features:
Accuracy: The O-buffer can provide much higher spatial precision for samples than the same resolution images and volumes. Even though the O-buffer still has a limited resolution, its spatial precision is often adequate or can be tailored to the application.
Efficiency: Sample position in the O-buffer is encoded as offsets to an implicitly defined regular grid, thus it is compact compared to other sample-based representations (e.g., sample lists). The O-buffer can be warped efficiently by incremental computation. It can further provide better image quality by avoiding multiple resamplings and delaying reconstruction to the final rendering stage.
Semi-regularity: The O-buffer strikes a middle ground between regular representations (images and volumes) and irregular representations (sample lists) for samples. It is a semi-regular structure which lends itself to efficient construction and rendering.
Uniformity: The O-buffer provides a uniform framework to represent various irregular and regular sample primitives in computer graphics, such as images, points, and volumes, and thus support mixing these primitives in the same scene.
Versatility: The O-buffer is a versatile representation and can be used to solve a variety of problems in graphics, such as limited resolution of images and volumes, antialiasing for surface rendering, sample caching for image-based or point sample rendering, data mixing for hybrid volume rendering, irregular sample organization, and level-of-detail management for samples, to name a few.
Flexibility: Storing more spatial information with samples makes it possible for O-buffers to store multiple samples in one cell. Therefore, more flexible and accurate multi-resolution schema can be developed for images and volumes.

In the next sections, we introduce the O-buffer, and related quantization, data structures, sources, hierarchical structures, and rendering algorithms. Section 6 introduces the layered depth O-buffer as a case study for nonuniform O-buffers. Section 7 presents hybrid volume rendering with 3D O-buffers. Section 8 describes our experimental results.

2 O-Buffer Representations

The O-buffer is basically a conventional image or volume except that the sample points in the O-buffer are no longer on a regular grid. A sample point position is recorded by its offset to the nearest point in a regular base grid. The base grid can be rectangular, cylindrical, spherical, etc. For simplicity, we will use a rectangular grid as the base grid, which is the most common case. The O-buffer can be either 2D or 3D. The number of samples in every grid cell of an O-buffer can be the same or different. In order to simplify the presentation, we use the following terms: *regular samples* for samples on a regular grid; *irregular samples* for unorganized samples; *offset samples* for samples whose positions are recorded by offsets to a regular base grid; *uniform O-buffers* where the number of offset samples stored in every grid cell of the O-buffer is the same; and *nonuniform O-buffers* where the number of offset samples stored in the grid cells may vary across the O-buffer. The regular grid can also be adaptive [Frisken et al. 2000], where the O-buffer is called an *adaptive O-buffer*.

We use the 2D O-buffer (offset image) to demonstrate the O-buffer concept in this section. The extension to the 3D case (offset volume) is straightforward. Figure 1 shows a small portion of a 2D uniform, nonuniform, and adaptive O-buffer.

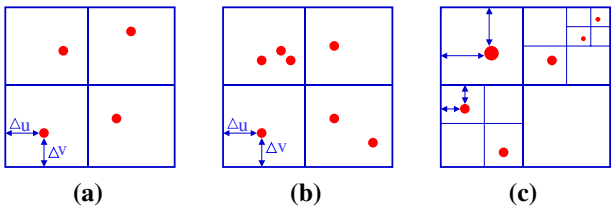

Figure 1: (a) A uniform O-buffer; (b) A nonuniform O-buffer; (c) An adaptive O-buffer.

2.1 Offset Quantization

By using the O-buffer, we can record the position of a sample point much more precisely than a conventional image with the same resolution. The offset can be recorded by using two floating point numbers which represent the offsets along two axes of the image coordinates. However, in most cases, a floating point number is an overkill. Usually, the offset can be quantized into one or two bytes for compact representation and efficient rendering. With a one byte offset, the offset is quantized into 16 levels in each axis. With a two byte offset, the offset is quantized into 256 levels in each axis.

We find that one-byte offset is useful in practice. First, one-byte offset for every pixel does not increase much the storage requirement for an image. A typical depth pixel needs 7 bytes (3 bytes for color, 4 bytes for depth). Therefore, one-byte offset is only 12.5% of the storage requirement for the O-buffer. Second, one-byte offset can provide reasonable spatial precision. Suppose the 2D base grid is 512×512. O-buffers with one-byte offset can provide spatial precision for sample points just as that of a grid of 8192×8192. We find that this is usually good enough for many applications. It can reduce the position error to be less than $1/32$ pixel distance in

Proceedings of the 14th IEEE Visualization Conference (VIS'03)
0-7695-2030-8/03 $ 17.00 © 2003 IEEE

each direction in the image plane. Figure 2 shows the error caused by the quantization. Suppose the projection of a point P is quantized into one of the $16 \times 16 = 256$ subpixels. Let the distance between this point P and its quantized point P' in the image plane be m. Then m should be less than half the diagonal length of a subpixel ($m \leq \sqrt{2}/32$ pixel). When we warp the image to a new image plane the error caused by the quantization is n, which is actually the projection of the 3D line PP' onto the new image plane. From another point of view, n/m represents the zoom factor of the line PP' from camera $C1$ to camera $C2$. If the zoom factor around the pixel P is f, then the error caused by the quantization is less than $\sqrt{2}f/32$. This means that even when the zoom factor is 10, which is unusual in practice, the error caused by the quantization is still less than half a pixel.

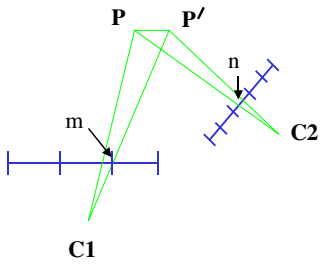

Figure 2: The error caused by the quantization of the offset. m is the quantization error at the O-buffer. n is the quantization error caused by warping this O-buffer to a new image plane.

By using quantization, O-buffers can achieve substantial savings of storage for irregular samples compared with conventional images and volumes. Suppose a uniform O-buffer has a base grid of 512 $\times$ 512. For each cell, a one-byte attribute value as well as a one-byte offset value are stored. The total storage requirement is 0.5MB ($512 \times 512 \times 2$). If we use a high-resolution buffer to store these samples, we need 64MB ($512 \times 16 \times 512 \times 16$) to achieve the same spatial precisions as that provided by the O-buffer.

2.2 Nonuniform O-Buffers

Nonuniform O-buffers provide more flexibility than uniform ones, thus may be more useful in some applications. However, the data structures for a uniform O-buffer and a nonuniform O-buffer are different. For a uniform O-buffer, we can use the same data structure as a conventional image or volume, which is just a 2D or 3D array of offset samples.

We propose two data structures for nonuniform O-buffers. During the construction stage of an O-buffer, it is more important to efficiently insert and delete offset samples. Therefore, we use a 2D or 3D array of linked offset sample lists to store O-buffers. During the rendering stage, it is more important to maintain spatial locality of offset samples in order to most effectively take advantage of CPU cache coherency. We can reorganize the offset samples into a linear array ordered bottom up and left to right in the image space. We calculate the location of the beginning of each scanline in this array and store them into a 1D array. Within each scanline, for each cell location, we store the offset from that location to the beginning of the scanline. We then use a double array of offsets to locate each offset sample. In order to find offset samples stored at a specific cell, we can simply use one offset to find the beginning of the scanline and then further use another offset to find the first offset sample at that location. This data structure is somewhat similar to the one proposed by Shade et al. [1998] for layered depth images. For an adaptive O-buffer, we can use a quadtree or octree structure.

3 Conversion to O-Buffers

Other typical computer graphics primitives such as triangle meshes, images, points, and volumes can be converted into O-buffers. A major motivation of converting other primitives to O-buffers is for hybrid rendering. After the conversion, we only need to render one primitive whose rendering can then be highly optimized and can even be realized in hardware. Also, we can determine the visibility of samples in a 3D O-buffer very easily. Therefore, different samples can be rendered and composited in correct visibility order, which is critical especially for volume rendering.

The conversion algorithms for uniform O-buffers and nonuniform O-buffers are different. When we convert other models to a uniform O-buffer, we want to minimize the error between the original model and the O-buffer, given a predetermined resolution for the O-buffer. When we convert other models to a nonuniform O-buffer, we want to minimize the number of samples in the O-buffer, given a predetermined error between the original model and the O-buffer.

Regular volumes can be treated as special 3D O-buffers with zero offsets. Converting depth images to O-buffers can be achieved by first converting depth images into triangle meshes and then converting triangle meshes into O-buffers. In this section, we focus on converting triangle meshes and irregular samples to O-buffers.

3.1 Triangle Meshes

Discretizing triangle meshes is an important research topic for sample-based graphics. Available solutions such as ray tracing [Grossman and Dally 1998; Lischinski and Rappoport 1998; Pfister et al. 2000] and scan conversion [Kaufman 1987] usually convert triangle meshes to samples on a regular grid. Other methods [Chen and Nguyen 2001; Cohen et al. 2001; Rusinkiewicz and Levoy 2000] convert triangle meshes to irregular point samples.

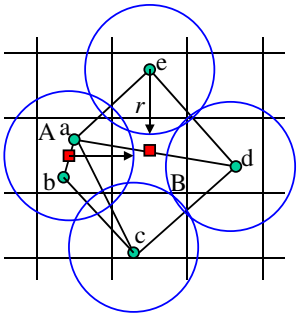

Figure 3: Conversion of a triangle mesh into an O-buffer. The small green circles represent vertices of a triangle mesh. The red squares represent offset samples in the corresponding O-buffer. The big blue circles are Poisson disks.

We develop a grid-based vertex clustering algorithm which can convert triangle meshes into a 3D uniform O-buffer in linear time. Figure 3 shows our method. The algorithm consists of three steps: First, we project all vertices of the triangles to the base grid of an O-buffer and scan all triangles. We mark all cells which the triangle mesh passes through. Second, we compute a representative sample for each marked cell. For each cell with vertices falling into it such as cell A in Figure 3, we perform vertex clustering and compute the position of the representative sample by the quadric error metrics [Garland and Heckbert 1997; Lindstrom 2000]. For each cell without vertices falling into it such as cell B in Figure 3, we still need to compute one representative sample in order to avoid holes in the O-buffer representation. We can get the triangles passing through the cell and compute the position of the representative sample so as

Proceedings of the 14th IEEE Visualization Conference (VIS'03)
0-7695-2030-8/03 $ 17.00 © 2003 IEEE

to minimize the distance from this sample to all triangle surfaces in this cell. Because the offset is quantized, the possible positions for the representative sample are limited. However, there still may be more than one possible position. For example, in Figure 3, any position along the segment of line *ad* in cell B may minimize the distance. We then use a Poisson disk technique [Mao 1996] to pick up the final position for the sample so that the distance from this sample to neighboring offset samples is not less than a threshold. Figure 3 shows Poisson disks with radius r. The final position of the offset sample in cell B must be outside the Poisson disks of its neighboring offset samples. The radius r can be decided on by trial and error. We start from a large number and then gradually reduce it until some position is available for the offset sample in the cell. By this way, we can both minimize the error and guarantee the uniformity of overall sample distribution. Finally, we convert the representative samples into offset samples, quantize the offset, and organize them into the O-buffer.

For nonuniform and adaptive O-buffers, we can first convert triangle meshes to uniform O-buffers using the method described before. Then, for each cell of the base grid, we compute the error between the representative sample and the original model. If the error is less than a predefined threshold, conversion for this cell is done. Otherwise, we subdivide this cell into subcells, compute a representative sample for each subcell, and compute the error again. We repeat this process until the predefined error threshold is satisfied. We can accelerate this process by using the normal error metrics proposed by Brodsky and Watson [Brodsky and Watson 2000] to directly decide if a cell needs to be subdivided without computing a representative sample first.

3.2 Irregular Samples

Irregular samples are abundant in computer graphics. One typical example is point clouds from 3D range scanners. Using O-buffers to organize irregular samples is straightforward. After a base grid is selected, irregular samples are then projected to 2D/3D space, and their offsets to the base grid are quantized.

In the remainder of this section, we compare the O-buffer with two other data structures developed to organize irregular samples: quadtree/octree and QSplat [Rusinkiewicz and Levoy 2000]. Figure 4 shows the comparison in 2D. Figure 4a shows some irregular samples we want to organize. There are different ways to encode these samples with the same spatial precision. Figure 4b uses a high-resolution buffer, which is sparse and wasteful. The buffer can be compressed by using a quadtree as in Figure 4c. Figure 4e shows a uniform O-buffer to organize these samples. For the quadtree representation to achieve the same spatial precision as the O-buffer, the depth of the quadtree needs to be higher.

Figure 4d shows a QSplat representation. QSplat also uses relative offsets from base positions to record the position of samples. However, the base positions of QSplat is a hierarchy of bounding spheres. Each node of the tree contains the sphere center and radius. The position and radius of each sphere is encoded relative to its parent in the bounding sphere hierarchy. QSplat is efficient to encode a very large number of nonuniform point samples in a hierarchy. However, QSplat needs more effort to establish initial clusters and build the tree. Compared to QSplat, the base positions of the O-buffer are on a regular grid. This makes construction and rendering of the O-buffer easier and faster. Therefore, the O-buffer is a more efficient representation for relatively uniform samples and for certain applications that need construction on the fly.

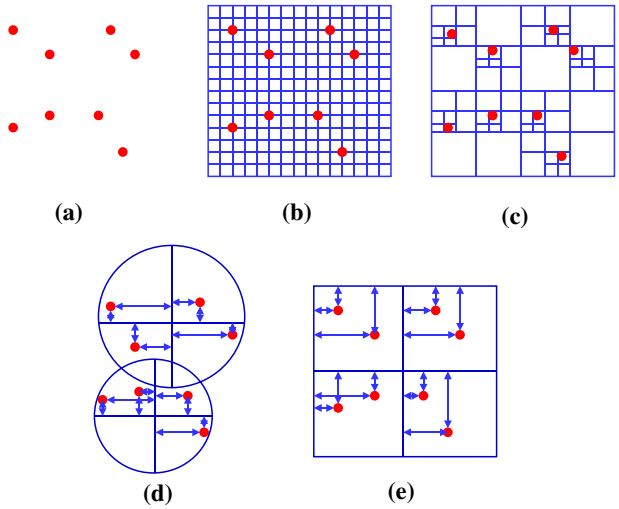

Figure 4: Different ways to organize samples with the same spatial precision: (a) Some samples; (b) A 16×16 uniform buffer; (c) A quadtree with a depth of 5; (d) Bounding spheres used in QSplat. (e) A 2×2 O-buffer. Each grid cell contains two offset samples.

4 Rendering of O-Buffers

There are two basic methods to render the O-buffer: either by splatting or by ray tracing. Considerable speedup can be gained for these two methods by exploiting the semi-regular structure of O-buffers and the quantization of offsets.

Splatting is an object order approach and memory coherence of data can be exploited. Incremental computation has been used to speed up the rendering of samples on a regular grid [Grossman and Dally 1998; McMillan 1997]. O-buffers with quantized offset can also be rendered rather efficiently by using incremental computation and lookup tables [Popescu and Lastra 1999; Qu et al. 2000].

Splatting consists of two steps: warping samples to the image plane followed by reconstruction and resampling. There are several ways [Grossman and Dally 1998; McMillan 1997] to warp samples in depth images and volumes to a new image plane. In this paper, we base our presentation on a method and notation proposed by Shade et al. [1998]. We use 2D O-buffers (i.e., offset images) to demonstrate our method. Let T be the 4×4 matrix to transform a point from an offset image plane to a destination image plane, and $(x_1, y_1, z_1, 1)$ be the homogeneous coordinates of samples in the offset depth image. Then, the warped sample position at the destination image plane can be computed by:

$$T\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = T\begin{bmatrix} x_1 \\ y_1 \\ 0 \\ 1 \end{bmatrix} + z_1 T\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \mathbf{start} + z_1.\mathbf{depth} \quad (1)$$

To compute the position of the next sample along the same scanline, **start** can be simply incremented. For the O-buffer, the next sample along the same scan line is $(x_1 + 1 + u, y_1 + v, 0, 1)$, where (u, v) is the offset of this sample. If the offset is quantized to one byte we can precompute $T\begin{bmatrix} u, v, 0, 0 \end{bmatrix}$ for all possible levels of the offset (256 levels in our case) and store them in one lookup table, **table**. Before rendering a new image, we use the new camera information to precompute the lookup table indices. Then, we can use incremental computation:

Proceedings of the 14th IEEE Visualization Conference (VIS'03)
0-7695-2030-8/03 $ 17.00 © 2003 IEEE

$$T\begin{bmatrix} x_1+1+u \\ y_1+v \\ 0 \\ 1 \end{bmatrix} = T\begin{bmatrix} x_1 \\ y_1 \\ 0 \\ 1 \end{bmatrix} + T\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + T\begin{bmatrix} u \\ v \\ 0 \\ 0 \end{bmatrix}$$
$$= \mathbf{start} + \mathbf{xincr} + \mathbf{table}[u,v] \quad (2)$$

Compared to regular images, there is only a slight increase in the rendering time for O-buffers. After projecting these samples to the image plane, we need to reconstruct and resample. This has been a well studied problem in the literature [Mark et al. 1997; Zwicker et al. 2001].

Ray tracing can generate high quality images and is the best method for global illumination. O-buffers have some advantages over other sample-based models for ray tracing. First, the semi-regular structure of the O-buffer lends itself well to a grid-based ray traversal acceleration method. Second, locating the neighboring samples for a sampling point along a ray is trivial. A look-up table can be used to accelerate the calculation of the distance between the sampling points and the O-buffer samples. Third, the quantization of offsets can avoid many floating point arithmetic operations.

5 Hierarchical O-Buffers

Multi-resolution is a core technology in computer graphics for managing the complexity of models. O-buffers open the possibility to develop more accurate and flexible multi-resolution representations for regular images and volumes. Hierarchical O-buffers can be divided into two categories: hierarchical uniform O-buffers and hierarchical nonuniform O-buffers. Figure 5 shows a hierarchical uniform O-buffer and a hierarchical nonuniform O-buffer, both in 2D.

The O-buffers provide the flexibility for the number and position of samples in low resolution O-buffers. How to compute the number and position of samples is based on the application. In this section, we present an algorithm which can generate more accurate low resolution offset images for regular or depth images.

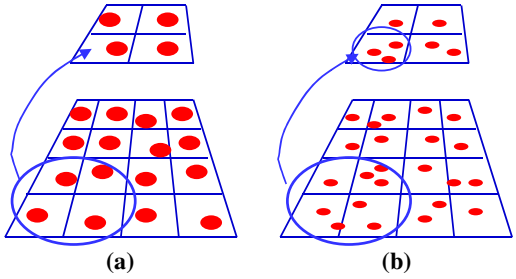

Figure 5: Hierarchical O-buffers: (a) A hierarchical uniform O-buffer; (b) A hierarchical nonuniform O-buffer.

The conventional way to generate low-resolution images is to downsample the data by some low-pass filter and store the result in a regular grid. Instead, our algorithm first convert the original image to a triangle mesh with color or geometry attributes. Then, we collapse every four neighboring vertices into one based on some triangle mesh simplification method such as the quadric error metrics [Garland and Heckbert 1998; Hoppe 1999] which can preserve color or geometry attributes of the mesh. After that, the position and color of new vertices are stored in a low-resolution offset image. This framework can be extended to 3D volumes. Similarly, a regular volume can be first converted into a tetrahedra mesh. Then, we use a tetrahedra mesh simplification method [Cignoni et al. 2000] to simplify this mesh until only one vertex left for each cell in the regular grid of the low resolution 3D O-buffer.

6 Layered Depth O-Buffers

The layered depth image (LDI) proposed by Shade et al. [1998] is an important extension to the depth image. It provides the information for the occluded parts of an object. The representation is compact compared to multiple images because LDI reduces the redundancy in multiple images.

Having multiple depth images, an LDI can be constructed by warping these depth images into a common camera view. As Shade et al. [1998] pointed out, there are some disadvantages to the LDI. First, pixels undergo two resampling steps in their journey from input image to output. This can potentially degrade image quality. Second, some information in the original images can be lost if a surface is better sampled in one of the images than it is from the viewpoint of the LDI. Chang et al. [1999] proposed LDI tree to solve the problem caused by different sampling rates in the original images.

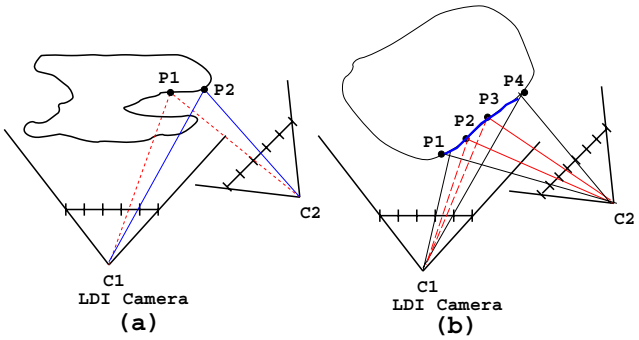

Figure 6: (a) multiple resampling problems of the LDI. When the image taken at camera C2 is warped to the LDI camera, a resampling is needed (P2 is resampled from the image taken at C2). During the rendering stage, these resampled pixels are warped to the output image and are needed to be resampled again. (b) Different sampling rates in the original images. Part of the surface is better sampled at the image taken from C2 than at the image taken from the LDI camera.

We introduce a layered depth O-buffer (LDOB) as an extension of the LDI to overcome these two disadvantages. Figure 6a demonstrates the two resampling steps of the LDI. When we warp the depth images to the LDI camera position, the pixels do not fall on a regular grid. Therefore, resampling is needed at the construction stage of the LDI. During the rendering stage, the LDIs are warped to an image plane. At this time, another resampling is needed. We think that the resampling at the construction stage of the LDI is unneccesary because the LDI is just an intermediate representation and is not used for display. Therefore, this resampling can be delayed to the rendering stage with the help of an O-buffer. When we warp an image to the viewpoint of the LDI we use an O-buffer to record the position of these pixels in the LDI instead of resampling the warped image. Thus, we avoid the two resampling problem.

Figure 6b demonstrates the different sampling rates problem. If the original depth image resolutions are different, then a surface can be better sampled at the original image which has a higher resolution than the LDI image. Even if the resolutions of the original images and LDI are the same, a surface can still be better sampled at some image other than the LDI because of the orientation of the surface.

We propose a two step warping algorithm to preserve all the information in the original images. The LDOB is still constructed by warping the depth images one by one to the LDOB camera position. However, before we warp a depth image to the LDOB, we first try to

Proceedings of the 14th IEEE Visualization Conference (VIS'03)
0-7695-2030-8/03 $ 17.00 © 2003 IEEE

reconstruct this depth image from current LDOB by warping pixels in the LDOB to the image plane of this depth image. Only pixels in the original depth image which cannot be reconstructed from the LDOB are warped and stored into the LDOB. These pixels either represent new surfaces or surfaces which cannot be sampled well in the LDOB.

We first select the viewpoint of one image as the viewpoint of the LDOB and use that image as the initial LDOB. At that time, the LDOB is simply a one layered image with zero offset for each pixel. Then, we sort the depth images based on the angle of the camera of the depth images and the camera of the LDOB. At the first warping step, we warp the LDOB to one original image with the minimum angle and mark the pixels which are sampled in this original image but not well sampled in the LDOB. If no pixels in the LDOB are warped to a pixel in the original image we think this pixel is not well sampled in the LDOB and is marked. In some cases, even though some pixel in the LDOB is warped into a pixel in the original image but their depth or color differences are beyond a threshold, this pixel is still marked as a not-well-sampled pixel. Then, at the second warping step, we warp these marked pixels in the neighboring images to the LDOB and record their positions with the O-buffer and construct the new LDOB. We repeat this process until all images are warped. We use Figure 6b to illustrate our algorithm. We pick up C1 as the LDOB camera position. At the first warping step, we warp pixels in the LDOB to the image plane of camera C2. In this example, there are pixels from the LDOB falling into pixels P_1 and P_4 in the image from C2 and their depth and color differences are less than a threshold. Thus, P_1 and P_4 are already well sampled in the current LDOB but P_2 and P_3 are not. At the second warping step, we warp P_2 and P_3 from camera position C2 to the LDOB. Instead of resampling, we use offsets to record their precise positions in the image plane of the LDOB.

If the original images have dramatically different resolutions, we can combine an LDI tree [Chang et al. 1999] and an LDOB to develop a hierarchical LDOB tree. The LDOB tree has the identical structure as the LDI tree. The only difference is that for each cell in the hierarchy an LDOB instead of an LDI is stored. Compared with the LDI tree, the LDOB tree can avoid resampling and provide better reconstruction by storing more spatial information with samples.

7 Hybrid Rendering with O-Buffers

Volumetric data constructed from CT and MRI scans are widely used in medical virtual systems and they are usually rendered using volume rendering. With the increasing requirement for more realistic simulations, hybrid volume rendering becomes more important. For example, some medical operations involve penetrating the human organ with a medical devices such as a scalpel or implanting a device such as a pace maker. These objects may be modeled by different computer graphics primitives. The human body is usually modeled by volumetric data constructed from the CT or MRI scans. The medical devices can be modeled by polygonal, point-based or image-based models. These models need to be mixed in the same scene in the medical virtual system. Another example for hybrid volume rendering is a polygonal plane flying through a volumetric cloud over an image-based terrain.

Hybrid volume rendering is a challenging problem. All primitives in a scene must be drawn and mixed in topologically depth sorted order because compositing with the over operator for volume rendering is not commutative [Kreeger and Kaufman 1999]. Hybrid volume rendering methods can be categorized into two classes: all primitives can either be rendered and mixed separately or they can be first converted and mixed into a common primitive. As we mentioned before, if we first convert all primitives in a scene into one common primitive, we only need to render one primitive whose rendering can then be highly optimized and can even be implemented in hardware. Handling only one primitive makes hardware unit small and thus more efficient. This approach is used in current graphics boards which convert all geometric models into triangles first so only one primitive is really processed by hardware. Therefore, we believe that converting all primitives into one common primitive first has its advantages and may be a starting point to design hardware for hybrid volume rendering.

The O-buffer provides a unified framework for hybrid rendering because various primitives (e.g., triangle meshes, points, depth images, volumes) can all be converted into O-buffers which provide high spatial precision for samples and avoid unnecessary resampling. If these primitives have no relative motion, they can be premixed into one 3D nonuniform O-buffer. The O-buffer can easily solve the visibility order for these samples. Thus, blending and compositing of different objects can be handled elegantly by rendering the samples in the O-buffer from back to front. Another major advantage of the O-buffer is that when multiple samples from different models occupy the same cell of the 3D space, the redundancy can be reduced by only storing one sample based on some predetermined overlapping rules.

There are other ways to mix triangle meshes, images, points, and volumes. For example, these objects can be separately warped and z-buffers can be used to solve the visibility. One problem with this approach is that it is inefficient to blend multiple transparent objects. These objects have to be sliced into slabs in the depth direction and special attention must be paid to guarantee that slabs from different primitives in the similar depth zone are rendered and mixed together. Another approach is to organize all these objects in an octree-based structure which can be then splatted from back to forth. This approach is similar to our method. However, O-buffers are more compact and can be warped faster than sample lists organized as an octree.

8 Experimental Results

All our experimental results with O-buffers have been generated on a Dell Dimension 8200 desktop with a 2.53GHz Pentium 4 CPU, 1GB of RAM, and an Nvidia GeForce4 graphics board with 64MB memory. Figure 7 shows the image of a uniform O-buffer model which in turn was converted from the Stanford Buddha using the method presented in Section 3.1. The image resolution is 512 $\times$ 512. The original triangle mesh model has 543,652 vertices and 1,087,716 triangular faces. The data has been converted to a 512 $\times$ 512 $\times$ 512 uniform O-buffer. It takes about 23 seconds for the conversion after the data is loaded into memory and 0.05 seconds for rendering by projecting OpenGL points. The conversion can be done once as a preprocessing. The average error measured as the distance from the offset samples to the triangle mesh is 0.08 grid unit. The maximum error is 0.13 grid unit. The number of total offset samples is 529,179.

Our next two experiments demonstrate that the image quality can be improved by storing more spatial information with samples. Figure 8 shows an example of using the O-buffer to preserve the geometry in a depth image of the Stanford dragon. We first generated a depth image of the model. The image resolution is 1024 $\times$ 1024 (see Figure 8a). Figure 8b shows a portion of the image. We constructed a low-resolution 256 $\times$ 256 image by box filtering the original image. Figure 8d shows a portion of the regular low resolution image, where the edges are smoothed out. Then, we convert the depth image to a triangle mesh and generate a low resolution 256 $\times$ 256 uniform O-buffer using the mesh simplification method based on quadric error metrics [Garland and Heckbert 1997; Lindstrom 2000] (see Section 5) . Figure 8c shows a portion of the O-buffer image, where the edges are better preserved.

Proceedings of the 14th IEEE Visualization Conference (VIS'03)
0-7695-2030-8/03 $ 17.00 © 2003 IEEE

Figure 7: An image of an O-buffer model converted from a triangle mesh.

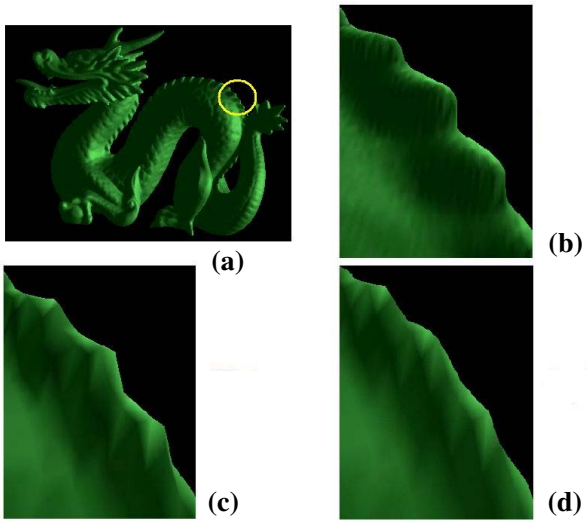

Figure 8: More accurate multiresolution with an O-buffer: (a) A depth image of the dragon; (b) A portion of image (a); (c) An image of a low resolution O-buffer; (d) A low resolution regular image.

Figure 9 demonstrates the advantages of the LDOB. We use a curved checkerboard for the experiment. We rendered seven depth images of the checkerboard from different camera positions around the object. The resolution for each image is 512×512. These seven images have a total of 673,993 non-background pixels. Then, we assemble an LDI and an LDOB, separately. The LDI has 102,362 non-background pixels while the LDOB has 210,855. Both LDI and LDOB can reduce the redundancy of the original images. However, LDOB keeps more samples from original images for surfaces which are not well sampled from the camera of the LDI. Figure 9a shows the image from the viewpoint of the LDI camera. The red quadrilateral region is not well sampled from this viewpoint. Figure 9b shows a portion of the image rendered by warping the LDI back to one of the original images which has a better sampling rate for the red quadrilateral region. In order to avoid holes, a large splat size has to be used and the image is therefore blurred compared to the original image. However, if the LDOB is used, we can keep the samples from the original images. When we warp the LDOB back, we still have enough sampling rate and the final image is still sharp.

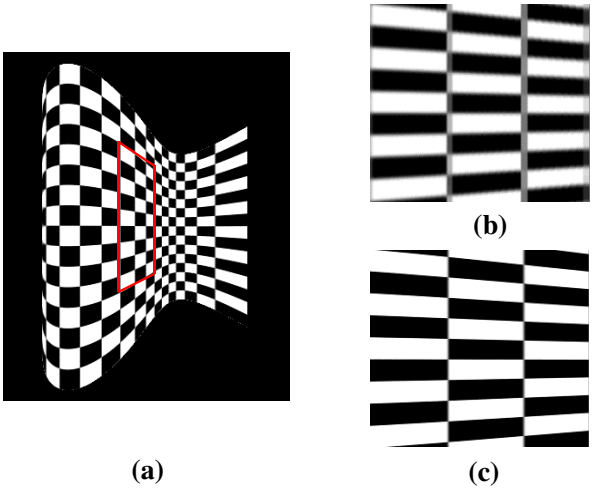

Figure 9: Improving image quality with an LDOB: (a) A curved checkerboard; (b) An image rendered by warping an LDI; (c) An image rendered by warping an LDOB.

Because the O-buffer is used to avoid multiple resampling, the image has almost the same quality as the original image (see Figure 9c). It takes 0.13 seconds to render the LDI and 0.15 seconds to render the LDOB. The LDOB has more samples to warp. However, resampling is faster for the LDOB because the LDI has to use a larger splat size than the LDOB. If we do not use quantization and incremental warping, the time to render the LDOB is 0.19 seconds. Thus, compared with the sample list representation, the O-buffer can gain more than 20 percent speedup for splatting.

We also obtained results for hybrid volume rendering with O-buffers. Figure 10 shows an example which mixes a volumetric CT lobster with an image-based skeleton hand. The resolution of the lobster is $320 \times 320 \times 34$. The image-based human hand consists of six 512×512 depth images which were generated by rendering a geometric model from the stereolithography archive at Clemson university. These images were projected to 3D space and combined with the lobster into a 3D O-buffer. Then, the 3D O-buffer was rendered by volume splatting. The rendering time is 1.2 seconds. It can be seen that these two primitives are blended nicely.

9 Conclusions and Ongoing Work

Our primary contribution in this paper has been the introduction of the O-buffer – a novel uniform framework for modeling and rendering a variety of 2D and 3D sample-based primitives. The O-buffer is a simple primitive, yet more flexible and powerful than a conventional image or volume. It is an efficient representation, supporting a much higher spatial precision for sampling points without increasing the sampling rate or resolution. Image quality is further improved by avoiding multiple resamplings and delaying reconstruction to the final rendering stage. The O-buffer can also be rendered efficiently by exploiting the semi-regular structure of O-buffers and the quantization of offsets.

O-buffers have the potential to greatly improve the modeling power of images and volumes. As a unified representation, the O-buffer will be helpful to fill in the gaps among various sample-based representations and finally, the gap between the geometry-based approach and the sample-based approach. Given the significant role that sample-based rendering is playing in computer graphics, O-buffers will be very useful and have a high potential for both modeling and rendering. It will have an impact on real time ren-

Proceedings of the 14th IEEE Visualization Conference (VIS'03)
0-7695-2030-8/03 $ 17.00 © 2003 IEEE

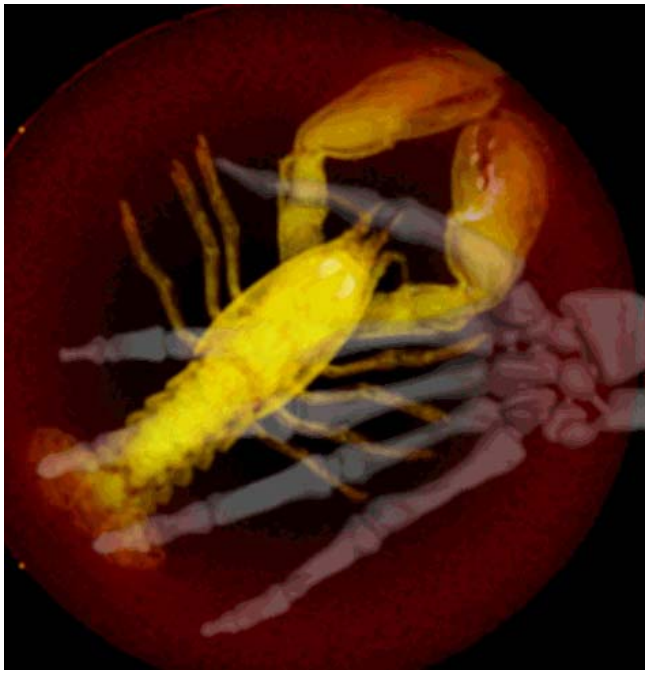

Figure 10: Mixing a volumetric lobster with an image-based hand into one 3D O-buffer which is then rendered by splatting.

dering, level-of-detail management for samples, volume rendering, and volume graphics.

We are currently exploring the use of O-buffers for feature-preserving simplification, the conversion of volume data defined on curvilinear or irregular grids into an O-buffer so that the many efficient manipulation and rendering methods developed for regular volumes may be exploited. We also plan to design a hardware protocol based on O-buffers for hybrid volume rendering.

Acknowledgments

This work is partially supported by ONR grant N000149710402 and NSF grant CCR0306438. The Buddha and Dragon models are courtesy of Stanford University. The human hand model is courtesy of Clemson University. We want to thank Michael Ashikhmin, Manuel Menezes de Oliveira Neto, and Klaus Mueller for their valuable comments on the early draft of this paper. We would like to thank Susan Frank for proofreading the draft.

References

BOTSCH, M., WIRATANAYA, A., AND KOBBELT, L. 2002. Efficient high quality rendering of point sampled geometry. *Proceedings of Eurographics Rendering Workshop 2002*, 53–64.

BRODSKY, D., AND WATSON, B. 2000. Model simplification through refinement. *Proceedings of Graphics Interface 2000*, 221–228.

CARPENTER, L. 1984. The A-buffer, an antialiased hidden surface method. *Computer Graphics (Proceedings of ACM SIGGRAPH 84) 18*, 3, 103–108.

CHANG, C.-F., BISHOP, G., AND LASTRA, A. 1999. LDI tree: A hierarchical representation for image-based rendering. *Proceedings of ACM SIGGRAPH 1999*, 291–298.

CHEN, B., AND NGUYEN, M. X. 2001. POP: a hybrid point and polygon rendering system for large data. *Proceedings of IEEE Visualization 2001*, 45–52.

CIGNONI, P., CONSTANZA, D., MONTANI, C., ROCCHINI, C., AND SCOPIGNO, R. 2000. Simplification of tetrahedral meshes with accurate error evaluation. *Proceedings of IEEE Visualization 2000*, 85–92.

COHEN, J. D., ALIAGA, D. G., AND ZHANG, W. 2001. Hybrid simplification: combining multi-resolution polygon and point rendering. *Proceedings of IEEE Visualization 2001*, 37–44.

FRISKEN, S. F., PERRY, R. N., ROCKWOOD, A. P., AND JONES, T. R. 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. *Proceedings of ACM SIGGRAPH 2000*, 249–254.

GARLAND, M., AND HECKBERT, P. 1997. Surface simplification using quadric error metrics. *Proceedings of ACM SIGGRAPH 1997*, 209–215.

GARLAND, M., AND HECKBERT, P. S. 1998. Simplifying surfaces with color and texture using quadric error metrics. *Proceedings of IEEE Visualization 1998*, 263–270.

GROSSMAN, J. P., AND DALLY, W. J. 1998. Point sample rendering. *Proceedings of Eurographics Rendering Workshop 1998*, 181–192.

HOPPE, H. 1999. New quadric metric for simplifying meshes with appearance attributes. *Proceedings of IEEE Visualization 1999*, 59–66.

KAUFMAN, A. 1987. An algorithm for 3D scan-conversion of polygons. *Proceedings of EUROGRAPHICS 1987*, 197–208.

KREEGER, K., AND KAUFMAN, A. 1999. Hybrid volume and polygon rendering with cube hardware. *Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware 1999*, 15–24.

LINDSTROM, P. 2000. Out-of-core simplification of large polygonal models. *Proceedings of ACM SIGGRAPH 2000*, 259–262.

LISCHINSKI, D., AND RAPPOPORT, A. 1998. Image-based rendering for non-diffuse synthetic scenes. *Proceedings of Eurographics Rendering Workshop 1998*, 301–314.

MAO, X. 1996. Splatting of non rectilinear volumes through stochastic resampling. *IEEE Transactions on Visualization and Computer Graphics 2*, 2, 156–170.

MARK, W. R., MCMILLAN, L., AND BISHOP, G. 1997. Post-rendering 3D warping. *Proceedings of Symposium on Interactive 3D Graphics 1997*, 7–16.

MCMILLAN, L. 1997. An image-based approach to three-dimensional computer graphics. Tech. Rep. TR97-013, Department of Computer Science, University of North Carolina - Chapel Hill.

PFISTER, H., ZWICKER, M., VAN BAAR, J., AND GROSS, M. 2000. Surfels: Surface elements as rendering primitives. *Proceedings of ACM SIGGRAPH 2000*, 335–342.

POPESCU, V., AND LASTRA, A. 1999. High quality 3D image warping by separating visibility from reconstruction. Tech. Rep. TR99-017, Department of Computer Science, University of North Carolina - Chapel Hill.

POPESCU, V., EYLES, J., LASTRA, A., STEINHURST, J., ENGLAND, N., AND NYLAND, L. 2000. The WarpEngine: An architecture for the post-polygonal age. *Proceedings of ACM SIGGRAPH 2000*, 433–442.

QU, H., WAN, M., QIN, J., AND KAUFMAN, A. 2000. Image based rendering with stable frame rates. *Proceedings of IEEE Visualization 2000*, 251–258.

RUSINKIEWICZ, S., AND LEVOY, M. 2000. Qsplat: A multiresolution point rendering system for large meshes. *Proceedings of ACM SIGGRAPH 2000*, 343–352.

SCHILLING, A. 1991. A new simple and efficient antialiasing with subpixel masks. *Computer Graphics (Proceedings of ACM SIGGRAPH 91) 25*, 4, 133–141.

SHADE, J., GORTLER, S. J., HE, L. W., AND SZELISKI, R. 1998. Layered depth images. *Proceedings of SIGGRAPH 1998*, 231–242.

WESTOVER, L. 1990. Footprint evaluation for volume rendering. *Computer Graphics (Proceedings of ACM SIGGRAPH 90) 24*, 4, 367–376.

ZWICKER, M., PFISTER, H., VAN BAAR, J., AND GROSS, M. 2001. Surface splatting. *Proceedings of ACM SIGGRAPH 2001*, 371–378.

Proceedings of the 14th IEEE Visualization Conference (VIS'03)
0-7695-2030-8/03 $ 17.00 © 2003 IEEE