芸術科学会論文誌 Vol.6 No.1 pp.21-36

Point Splatting Based on Translucent Shadow Mapping and Hierarchical Bucket Sorting

Zorig Gunjee, Tadahiro Fujimoto, Norishige Chiba
Faculty of Engineering, Iwate University, Japan
E-mail: zorig@cg.cis.iwate-u.ac.jp, {fujimoto, nchiba}@cis.iwate-u.ac.jp

(a) (b) (c)

Figure 1. Examples of the rendered images of mixed point, polygon and volume data. (a) Composition of the particle model (cloud), the volume data (tree, grass, fountain, road), the polygon model (dragon) and the point clouds (building, basin) in an outdoor scene, (b) mixing the surface model (skull) and volume data (head) in medical imaging, (c) visualizing the volume data (retina), the translucent surface (iris) and the opaque surface (lens) in an artificial eye.

Abstract

In this paper we present our solution for shadow generation, visibility sorting and GPU based shading of translucent point data for point based rendering. Our *Translucent Shadow Mapping Algorithm* uses a spherical coordinate system and solves the distance-based sorting, transparency calculation, shadow mapping, omni-directional mapping and light intensity attenuation problems in one step. We also propose a novel algorithm for visibility sorting of unstructured point data using the *Hierarchical Bucket Sorting* approach. This algorithm uses less memory and produces precise back-to-front ordered slices compared to previous methods. Finally, the shading calculations are performed in the GPU, using the rendering-oriented attributes of the point splats. Furthermore, we demonstrate the efficiency and flexibility of our novel approach of point splatting by showing several rendering results for visualizing mixed 3D data sets.

Keywords: point splatting, translucent shadow mapping, visibility sorting, GPU acceleration, visualizing mixed 3D data

1. Introduction

Rendering is a fundamental component of computer graphics. At the highest level of abstraction, rendering describes the process of converting a description of a three-dimensional scene into an image. In the early years of computer graphics, research in rendering focused on solving fundamental problems such as determining which objects are visible from a given viewpoint. As these problems have been solved (*e.g.,* z-buffer techniques) and as richer and more realistic scene descriptions have become available, modern rendering has grown from the ideas from a broad range of disciplines, including physics and astrophysics, astronomy, biology, psychology and the study of perception, and pure applied mathematics.

Our work addresses the solution of two major problems of rendering complex scenes, which is still challenging due shortage of existing algorithms: (1) translucent shadow generation of point data and (2) effective visualization of unstructured point data with extended rendering oriented properties.

Points are clearly the simplest graphics primitive. In some sense, they generalize pixels and voxels toward irregular samples of geometry and appearance. The

conceptually most significant difference from triangle representation is that points –much like voxels or pixels – carry all the attributes needed for processing and rendering. Until recently, limited programmability has hampered the implementation of point-based rendering algorithms on graphics hardware. However, with the current generation of GPU, it is now possible to control large part of the point rendering process at the interactive rate. Current point primitives store only limited information about their immediate locality, such as their position in 3D space, the normal vector, the bounding ball, and the tangent plane disk. However, our point splatting system uses many other attributes to render different models. Splatting-based rendering techniques are currently the best choice for efficient high-quality rendering of point based geometries.

Translucency is also an important graphics effect that can be used to significantly increase the realism of the rendered scene to enable more effective visual inspection. Rendering of translucent objects is realized by a process called alpha blending in most modern graphics systems. For correct visualization, all objects must be "blended" in back-to-front order. This can be achieved either by sorting the fragments at each pixel in image space or by sorting the points in object space. Our bucket sorting algorithm performs a distance based slicing of all points in object space. It avoids sorting the points in the same slice. This important feature of not sorting all the points saves time, and the slicing operation performs much faster than sorting all points.

The main steps of proposed point splatting system are as follows:

Step 1. The first step in the rendering engine is preprocessing, which may include conversion to a common format (points with rendering-oriented attributes), coordinate transformation to the world coordinate space, data analysis for future processing, subdivision of polygonal meshes to adapt the point densities, and normal vector computation.

Step 2. The rendering system uses the *translucent shadow mapping method* to build a translucent shadow mapping table for each light source.

Step 3. In the final step, the following are carried out: simple object level visibility culling, *hierarchical bucket sorting* for back-to-front alpha blending, advanced shading for rendering-oriented points in the GPU and splatting-related computations.

Our main contributions are as follows:

- We use extended rendering-oriented attributes for the point splats. The shading of each splat is processed differently on GPU, depending on the additional properties of each splat.
- We propose a novel algorithm for visibility sorting using the *Hierarchical Bucket Sorting* approach.
- To solve the translucent shadow-mapping problem, we use a spherical coordinate system and it solves the distance-based sorting, transparency calculation, shadow mapping, omni-directional mapping and light intensity attenuation problems in one step. The proposed algorithm works even in the difficult situation where the light sources are inside the translucent object.

In other words, the proposed point splatting system can render translucent objects with the same efficiency as opaque objects. It can render polygon data, particle models, point clouds and volume data. Moreover, it handles mixed data types well, which is a problematical task for other rendering methods.

In Figure 1 we show several images, in which the mixed data visualization provides a better perception of the relationships between the different components of the images.

In this work we also attempt to demonstrate the benefits of our point splatting system to solve the most difficult problems of rendering mixed data objects such as (1) rendering unstructured point data, (2) shadow generation of translucent point objects and (3) mixed shading of surface and volume data on the GPU.

The rest of paper is organized as follows: examination of related work is described in Section 2, extended attributes for mixed rendering surface and volume points are briefly described in Section 3, the rendering pipeline and major modules of the system, namely translucent shadow mapping, sorting and shading, are presented in Section 4. Section 5 gives some experimental results, followed by the conclusion in Section 6.

2. Related work

The work presented in this paper is closely related to the following sub-domains in computer graphics: point splatting, translucent shadow mapping, visibility sorting, rendering of unstructured grids and mixed surface and volume rendering.

Point splatting. The use of points as display primitives was first proposed by Levoy and Whitted [1]. Currently, many researchers have demonstrated that a splatting approach is doubtless the best choice for high-quality and efficient point-based rendering [2, 3, 4].

Splatting is a popular algorithm for direct volume rendering that was first proposed by Westover [5].

Zwicker et al. [6] introduced EWA (elliptical weighted average) filtering to avoid the aliasing of surface textures and extended his framework to volume splatting. To accelerate volume splatting Chen et al. [7] proposed a hardware-accelerated adaptive EWA volume splatting algorithm. Although research on surface splatting has produced very impressive results, we face numerous open research problems in the translucent point based graphics. In this work we investigate the question of (1) what additional information we have to store with each point, (2) how to effectively generate the shadow of points with different levels of translucency and (3) what kind of sorting method is more optimal for alpha blending the unstructured point clouds.

Translucent shadow mapping. In recent years, both Williams' original Z-buffer shadow mapping algorithm [8] and Crow's shadow volumes [9] have gone through many variations. Hasenfratz et al. [10] made a broader survey of real-time soft shadow algorithms. Unfortunately, most shadow generation methods still do not scale well to scenes with translucent objects and most have difficulties with omni-directional lights.

Several papers propose a method for rendering translucent objects. Dachsbacher et al. [11] use two simultaneous rendering targets to map translucent objects. Recently, many authors have used the multi-pass shadow mapping or the depth peeling [12] method and the ray casting algorithm [13] to generate translucent shadow maps. Even though some of those translucency rendering and shadow generation methods can produce sufficient results, most of them still have challenging problems such as the performance penalty due their multipass nature [12] and the problem of undersampling [13].

Our method differs from those methods in the following ways.

- We implement the spherical mapping techniques on shadow mapping. It allows us to solve the omni directional light problems of the shadow mapping.
- Our algorithm helps to solve distance based sorting, transparency calculation, shadow mapping, light attenuation, collision detection and early ray termination problems.
- Unlike the other shadow mapping tables, our translucent mapping table can stores the full information of mapped points with rendering oriented attributes for physically correct shadows generation.

Visibility sorting. A large volume of research has been devoted to visibility sorting due to its importance in computer graphics. In more recent work, Govindaraju et al. [14] describe an efficient method for visibility sorting by performing image space occlusion computations. Callahan et al. [15] propose a hardware assisted visibility sorting algorithm that operates in both object space and image space. Some authors offer GPU-based parallel sorting of points [16, 17] for correct alpha blended rendering. We introduce the view aligned slicing method based on the hierarchical bucket sorting algorithm, which works in linear computational time. The work most relevant to our method is presented by Yagel et al. [18] and Chopra et al. [19]. Our algorithm uses less memory and produces precise slices compared to the previous methods.

Rendering of unstructured grids. Rendering of unstructured grids has been a topic of much research, and major advances have been made in performing this rendering efficiency. Two major techniques are popular in volume rendering algorithms for unstructured grids: ray-casting and splatting. The ray-casting approaches [20, 21, 22] store the grids in texture memory and performs cell traversal for each ray in the shader program. Splatting methods such as the projected tetrahedral method [23, 24, 25] have become one of most popular methods for rendering unstructured grids. All these approaches must overcome two computational bottlenecks: cell ordering and per cell processing, both of which are not easily solved. Since visibility ordering is essential for volume rendering unstructured grids via cell projection, many techniques have been developed that order the tetrahedra [26, 27]. The most recent work by Callahan et al [15] addresses to depth sorting by computing a partial ordering on the CPU and then using the GPU for help finish the sort. In this paper we are mainly focused on the accurate sorting of points (it is suitable only for point data) and shading each point splats differently on the GPU.

Mixing point clouds, polygons and volume data. Earlier implementations of visualization for mixed 3D data focused primarily on multi-pass hybrid methods. Levoy developed a hybrid ray tracing algorithm [28]. He modified the conventional ray tracer for handling polygon and volume data. Kreeger et al. [29] proposed a rendering approach based on 3D texture mapping. Roettger et al. [30] used 3D texture mapping and 2D texture mapping to combine projected tetrahedral volumes with isosurfaces. More recently, some researchers have developed hybrid rendering methods [31]. All of those works show numerous research problems to render the mixed scenes, such as the most effective way (multipass methods, hybrid methods or methods based on common presentation), illumination (including shadow) models of objects with different physical characteristics, and appropriate data structures for shading the mixed scenes.

In this paper, however, we present a completely different approach to rendering mixed data. We are primarily interested in the mixed rendering of unstructured data and in the improvement of visualization using translucent shadow mapping and GPU based shading. We chose the point model as the most suitable model for shared representation of polygon, volume and point data for rendering mixed objects. Although, much excellent research work has been performed on point splatting such as surface splatting [2, 3, 4] and volume splatting [19, 6]. We did not find appropriate studie on mixed splatting. The point type of representation and the splatting type of rendering meet a number of criteria that are important for mixed rendering of polygon, volume and point models.

- The conversion from a volumetric data or a mesh model to a point cloud in 3D space is conceptually and geometrically simple. In most cases we just replace a vertex or a voxel by a point splat. The important operation in converting the polygonal models into point models is the subdivision of the surface into finer-level triangles. The rendering-oriented attributes, such as the normal vector, the color value and the physical values can come from the original modeling data.
- Rendering-oriented point splats have unique flexibility in that they can form translucent (volume splatting) and opaque (surface splatting) splats. They can depict volume data, point clouds and micro objects with specific material properties.

3. Point attributes for rendering mixed 3D data.

Point-based rendering schemes have evolved as an efficient alternative to triangle-based rendering. We chose the point splatting method for mixed rendering due to (1) the efficiency in rendering complex environments, (2) zero connectivity for efficient streaming for GPU-based stream processing, and (3) productivity in visualizing both surface and volume data.

Current point primitives store only limited information about their immediate locality, such as their position in 3D space, the normal vector, the bounding ball, and the tangent plane disk. However, our point splatting uses many other attributes to render different models.

In our rendering system, we use the following data structures for rendering-oriented point splats.

- **3D geometrical attributes** (position, normal vector, local differential geometry information)
- **Color attributes** (color, opacity, material ID, texture ID, shadow coefficient)
- **Other values** (object ID, size, physical value)

For some attributes we define a separate lookup table and combine the corresponding values during the point rendering of the surface and volume data. We call our primitive a “rendering-oriented particle” because it presents a small point with additional physical values.

4. The rendering pipeline

The proposed point splatting pipeline is illustrated in Figure 2.

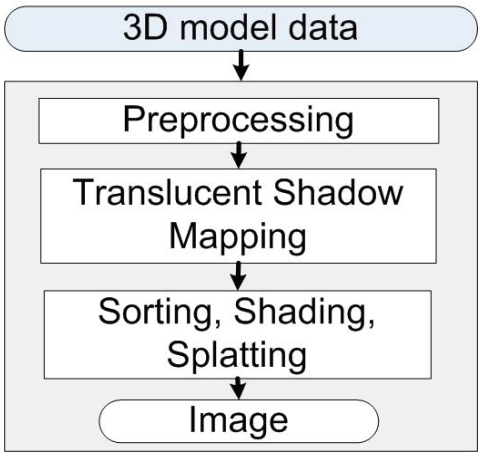


Figure 2. Structure of the extended point splatting system

The first step in the rendering engine is preprocessing, which may include conversion to a common format (points with rendering-oriented attributes), coordinate transformation to the world coordinate space, data analysis for future processing, subdivision of polygonal meshes to adapt the point densities, and normal vector computation.

Converting the polygonal models. Compared with the point cloud model, a polygonal model contains additional topological information that defines the neighboring relationship and order among points. The most important operation in the preprocessing step in rendering the polygonal models is the subdivision of the surface into finer-level triangles, where the triangles can be filled by the effective area of the point splats. To compute the area of the point splat, we use a tangent disk assigned to each point splat. If the area of the triangle is greater than the effective area of the point splats, the algorithm sub-divides the surface into

small patches. During conversion processing it can add transparency to polygon models. The normal vector of each point splat is computed using the polygonal topology of each vertex by smooth interpolation of the original vertices.

Converting the volume data. Computation of the normal vectors for voxel data is performed by using 3D edge detection algorithms such as the 3D Sobel operator. For each voxel, the local gradient vectors serve as the parameters for normal vector computation. All advanced volume rendering methods for classification and interpolation of the physical properties of voxels are used for definition of the color and opacity attributes of the voxels. The voxels with color, opacity and normal vectors are converted to points. The voxel coordinates serve as the position attributes of the point splats.

The rendering system then builds a translucent shadow mapping table for each light source using the translucent mapping method. In the final step, we carry out simple object level visibility culling, hierarchical bucket sorting for back-to-front alpha blending, advanced shading for rendering-oriented points in the GPU and splatting-related computations.

4.1 Translucent shadow mapping

Translucency is important for realistic graphics since many substances in nature are translucent. This paper presents a new method for real-time rendering of translucent shadows for point splatting. The terms translucency and transparency are often used synonymously; however, translucent can be thought of as "seeing through frosted glass", while transparent can be thought of as "seeing through clear glass." In translucent mapping, multiple objects contribute to a pixel's final color.

Because the correct order of objects relative to the light sources is important to calculate the correct color, unsorted depth buffering is insufficient for shadow mapping. The fast rendering of the shadows of transparent and translucent objects, preferably in real time, has been the subject of research over the last few years, but so far this is generally an unsolved problem. To help solve this problem, we use a new approach that is different than those used in previously published algorithms. We use the spherical coordinate system (see Figure 3) as an alternative way to map all visible points in the 2D mapping table. The algorithm uses the point to light source mapping approach. Each translucent shadow mapping table (TSM) contains depth, opacity and visibility sorting information for given light sources. For a given camera position, the final lighting and shadow are generated by composing appropriate subsets of the mapping tables.

This method allows us to render complex translucent objects with varying light and material properties. The translucent mapping algorithm divides large-scale data into a reliable small set of rendering-oriented subsets in the spherical coordinate system (see Figure 4).

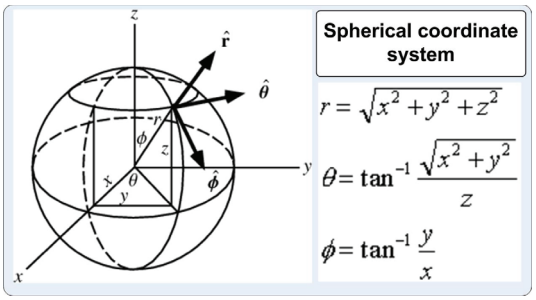


Figure 3. Spherical coordinate system

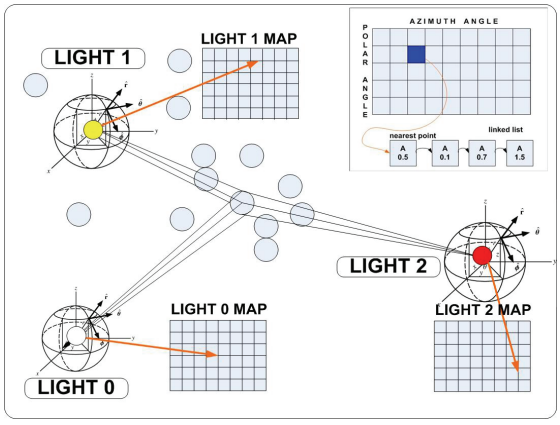


Figure 4. Translucent shadow mapping

The algorithm takes a point set as input, calculates the polar and azimuth angles for each light source, and classifies each point by its direction from each light source. The results are stored in the mapping tables. Each light source has its own mapping table. The rows correspond to the polar angle and the columns to the azimuth angle. The size of the mapping table depends on the desired precision, which is specified by the user. To calculate the index of the mapping table for the point P (x_p, y_p, z_p) in the mapping table for the light source P(x_l, y_l, z_l), we need to translate the point coordinate to the world space of each light source (see Eq. 1).

$$P(x_p, y_p, z_p) \Rightarrow P'(x_p - x_l, y_p - y_l, z_p - z_l) \quad (1)$$

The address of the cell on the destination mapping table is computed as follows (see Eq. 2,3).

$$Index_{col} = \left[\frac{(\phi - pol_{\min}) \cdot 180 \cdot k_{pol}}{\pi \cdot (pol_{\max} - pol_{\min})} \right] \quad (2)$$

$$Index_{row} = \left[\frac{(\theta - az_{\min}) \cdot 180 \cdot k_{az}}{\pi \cdot (az_{\max} - az_{\min})} \right] \quad (3)$$

where θ is the azimuthal angle (denoted λ when referred to as the longitude), ϕ is the polar angle (colatitude, equal to $\phi = 90° - \delta$ here δ is the latitude), pol_{max} and pol_{min} are the maximum and minimum angle formed between the z-axis and the line connecting P' to the center of the spherical coordinate system, az_{max} and az_{min} are the present maximum and minimum angle formed between the x-axis and line connecting the projection of P' onto the x-y plane to the origin of spherical coordinate system, k_{pol} and k_{az} is the number of divisions of the mapping calculation in the polar and azimuthal angles. Each cell in the mapping table contains information about the nearest point in one specific direction. The other points, which are inside the visible region, are mapped as distance-based sorted sub-lists in each direction. During the mapping process, distance-based insertion sorting occurs within the elements of a particular direction. The visible region is defined by Beer's law. Beer's law describes the empirical relationship that relates the absorption of light to the properties of the material through which the light is traveling. In essence, the law states that there is an exponential dependence between the transmission of light through a substance and the concentration of the substance, and also between the transmission and length of the material through which the light travels.

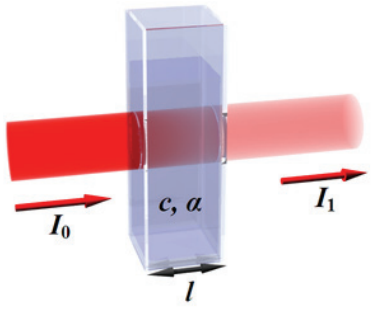


Figure 5. Illustaration of Beer's law. α is the absorption coefficient,*l* is the size of the material, *c* is the density, I_i is the intensity of the incident light, I_o is the intensity of light after passing the object.

We define the optical path as the extension of a ray's passage through an translucent point, scattering as the redirection of direct illumination from a light source (implying single scattering) into the optical path and toward the view point, and extinction as the cumulative effect of both out scattering and absorption. The effect of point properties on the intensity of a light ray can be described by a differential equation (see Eq. 4).

$$dI = \sigma(\vec{x}) + E(\vec{x})d\vec{x} \quad (4)$$

where $\vec{x}$ is the position of the point in three dimensions, $\sigma(\vec{x})$ describes the extinction per unit length, and $E(\vec{x})$ describes the emission and scattering per unit length into the optical path. When b and E are proportional to one another and are functions solely of position x we can define optical depth τ as follows (see Eq. 5)

$$\tau = \int \sigma(\vec{x})dt \quad (5)$$

Beer's Law gives a physical solution to this simple model and gives us the transparency T over the optical path as a function of optical depth (see Figure 5).
There are several ways in which the law can be expressed (see Eqs. 6,7,8):

$$A = \alpha l c \quad (6)$$

$$\alpha = \frac{4\pi k}{\lambda} \quad (7)$$

$$\frac{I_o}{I_i} = e^{-\alpha l c} \quad (8)$$

where A is the measured absorbance, α is the wavelength-dependent absorption coefficient, l is the distance that the light travels through the material, c is the concentration, λ is the wavelength of the light, I_i is the intensity of the incident light, I_o is the intensity of light after passing the object, and k is the extinction coefficient. We use the Eq.6 for our calculation of absorbance. In translucent shadow mapping, ℓ is assumed to be the size of the point splat, and c is assumed to be the opacity coefficient of the point. We assign a high (α=100) absorption coefficient to surface (opaque) points to ensure full absorbance of light at this point and to cast a shadow to the following points in its direction. The absorption coefficient of other translucent points is equal to one.
The relationship between absorbance and transmittance is illustrated in the following diagram (see Figure 6):
So, if all the light passes through a point without any absorption, then the absorbance is zero, and the percent transmittance is 100%. If all the light is absorbed (total absorbance>2.0), then the percent transmittance is zero, and the absorption is infinite.

In translucent shadow mapping, the point nearest to the light source receives full light, and the next point receives reduced light. The intensity of reduced light is calculated by Eq.6. Using this method it is possible to calculate shadow of an arbitrary point. For an arbitrary

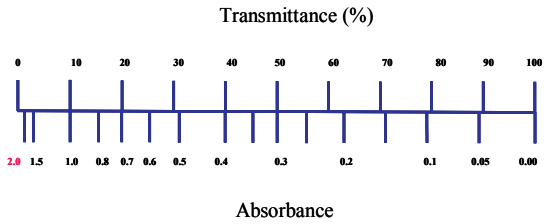


Figure 6. The relationship between absorbance and transmittance

point P_a, the total absorbance A_j of points between the light source and the point P_a is obtained using Eq.9.

$$A_j = \sum_{i \in \Omega j} \alpha_i l_i c_i \qquad (9)$$

Where Ω_j is the set of points which are nearer to the light source than the point P_a in a particular direction j of the ray, α_i is the wavelength-dependent absorption coefficient of each point, l_i is the distance that the light travels through the point i, c_i is the concentration of each point. If A_j >2.0, the point P_a gets a full shadow, otherwise the point P_a gets the reduced light.

The final shadow for each point is calculated as the sum of the light and shadows received from the different light sources. The steps proposed for the translucent mapping algorithm are as follows.

Step 1. An empty mapping table is created for each light source. The 3D positions of the light sources become the centers of the spherical coordinate system.

Step 2. The algorithm reads all the points and assigns corresponding values to each light table. For each point, it calculates the polar angle, the azimuth angle and the distance from the center of the sphere.

Step 3. It finds the related cell in the mapping table and compares the distance value with the first element of the cell.

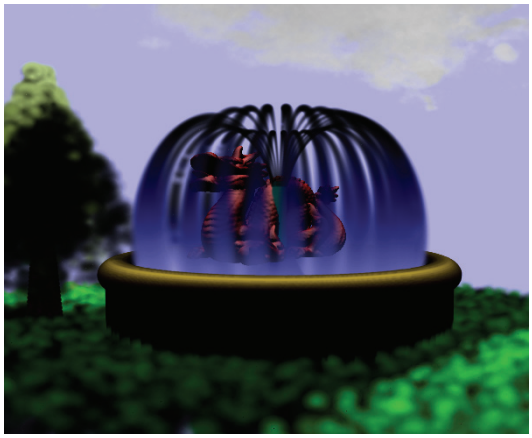
a. If the distance value of the new element is less than that of the head element, this new element becomes the head element of the cell.

b. If the distance value of the new element is greater than that of the head element, the algorithm calculates the absorbance of the leader point and finds its transmittance from the look-up table. If light can pass (see Eq. 9) to this element, it checks the next element on the linked-list of this cell.

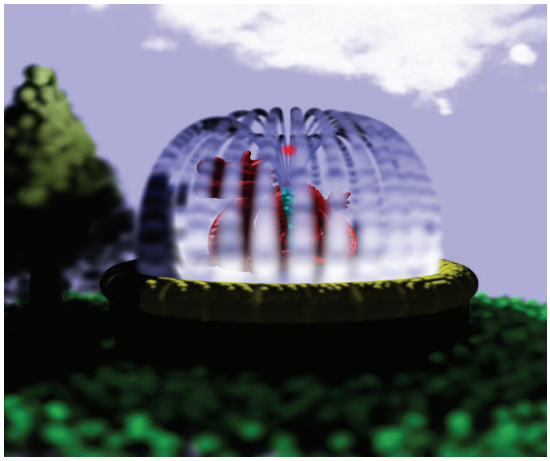
c. Step 3b will continue to process other points of this cell until a new element is inserted in the list, or until full absorbance reaches the visible limit (see Eq. 9) and the percent of the total of transmittance reaches to zero. The points behind of the transmittance region will get the full shadow.

This mapping is repeated for all neighboring cells which cover this point. The point coverage of the mapping table cells is defined by the size of a point and its distance from the center.

Figure 7 illustrates sample images of the shadow generation when light is placed on top of a scene and at the center of the outdoor scene.



(a)



(b)

Figure 7. Shadow generation when (a) the light source is above the clouds, and (b) the light is in the center (the red spot) of the fountain.

In Figure 17, we demonstrate the rendered results of different situations, were translucent and opaque objects located in different order from the light source. (a) shadow of opaque objects, (b) the translucent

object is located between the light and opaque object (c) the opaque object is located between the light and translucent object (d) the light is located inside the translucent object.

4.2 Sorting the points

All direct volume splatting approaches for unstructured points must overcome two computational bottlenecks: point ordering and per splat processing. Sorting the points is necessary for correct blending. Since sorting significantly affects performance, an effective sorting algorithm is an essential part of point splatting.

In this paper we present a sorting algorithm for point models that has a linear O(n) run-time and uses very little of the extra memory. Our algorithm rearranges the points in a back-to-front order from a given viewpoint.

The most popular sorting methods in computer graphics applications are comparison-based sorting methods such as quick sort, merge sort and insertion sort. A point system can be sorted on the GPU using "odd-even merge sort" or "bitonic sort" parallel sorting algorithms. However, the GPU based sorting using the 1024 x1024 texture requires processing in 100-210 rendering passes.

We analyzed distribution-type sorting methods and discovered that the hierarchical bucket sorting method can be more effective than the comparison-based sorting. Bucket sorting is particularly suitable for point rendering, since (1) all the points which are located at the same distance from the viewpoint can be regarded as points with the same priority or as located in one slice and (2) the minimal point size can serve as the most precise thickness of the slice. Actually, our bucket sorting performs a distance based slicing of all points but does not sort all the points. So, it avoids sorting the

points in the same slice. This important feature of not sorting all the points saves time, and the slicing operation is much faster than sorting.

The basic elements used in the hierarchical bucket sorting of points are described in Figure 8. We use two hash tables: one table is used for the first-level approximation and the other table is used for the accurate slicing. Each hash table points to a set of buckets. The first level approximation table is used for slicing the space between the near clipping plane and the far clipping plane into N slices. Each bucket of the accurate buckets is used to partition the approximation stage into K slices. The total number of slices is equal to NxK. We use two sets of buckets to avoid undesirable memory consumption by the empty buckets.

In our implementation the number of buckets (N) in these two bucket sets is the same and is defined by the desirable thickness of the slices, as shown in the following equations (see Eq.10,11).

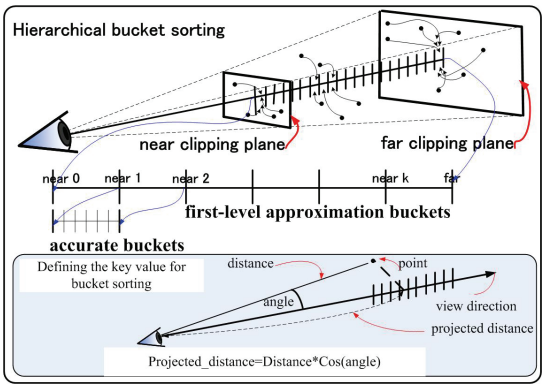


Figure 8. Basic elements of hierarchical bucket sorting

$$N_{slices} = \frac{(P_{far} - P_{near})}{\varepsilon} \tag{10}$$

$$N_{buckets} = \left[\sqrt{N_{slises}}\right] \tag{11}$$

where P_{far} is the farthest point in the view frustum, P_{near} is the nearest point in the view frustum, ε is the thickness of the slices. In a general case, ε can be defined by the *Nyquist Sampling Theorem*. In our rendering, we define ε as the minimum value of the size attribute for all points. When we rendered the

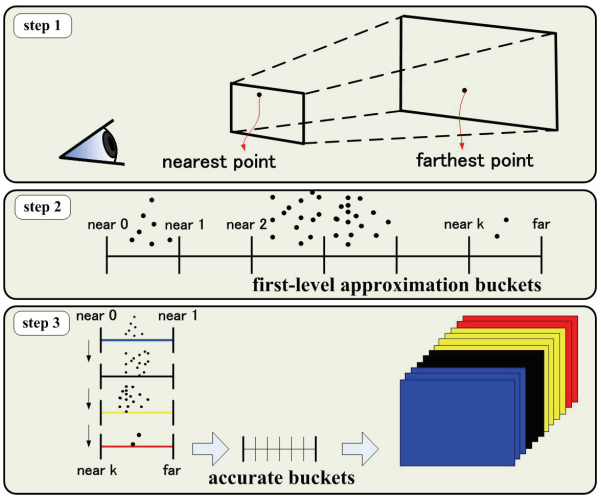


Figure 9. Steps of the hierarchical bucket sorting

outdoor scene with the artificial fountain (Figure 1 (a)), the perspective viewing volume was specified with the near clipping plane equal to 0.01 and the far clipping plane equal to 10. The size of the smallest point in our mixed 3D model data was equal to 0.001. The total number of slices was calculated as (10-0.01)/0.001 and was equal to 9990. To generate the 9990 slices we needed just 200 buckets: 100 buckets for the first-level approximation, and 100 buckets for the accurate slicing. The steps of our hierarchical bucket sorting algorithms are shown in Figure 9.

Step 1. View culling eliminates groups of points outside the view frustum. Culling is achieved by simple view transformation of the point position and checking the point positions against the camera and projection setting. During the visibility culling operation, the algorithm calculates the minimum and maximum distance of the point clouds from the camera.

Step 2. The algorithm traverses all the culled point splats and distributes them into the first-level approximation buckets. The points are assigned into buckets using a simple calculation based on the projected distance from the camera to the point. (see Eq. 12)

$$P_{dist} = Dist_{camera-point} \bullet \cos(angle_{view\_direction\_and\_point}) \quad (12)$$

The $Dist_{camera-point}$ value is calculated as distance between the camera position and the point position. The $\cos(angle_{view\_direction\_and\_point})$ value is calculated using the dot product of the view direction vector and the point vector (from camera to point position) (see Figure 8).

The algorithm divides all the points into $N_{buckets}$ buckets based on the projected distance. The index of the first-level bucket (hash function) for given points is calculated by the following equation (see Eq. 13).

$$Index_{f\_l\_bucket} = \left[\frac{N_{buckets} \cdot (P_{dist} - P_{near})}{(P_{far} - P_{near})} \right] \quad (13)$$

where P_{far} is the farthest point in the view frustum, P_{near} is the nearest point in the view frustum, $N_{buckets}$ is the number of buckets and P_{dist} is the distance value of the point form camera position projected to the view vector.

Step 3. In the last step, we distribute each non-empty bucket of the approximated bucket list into the accurate buckets. We use only additional one hash table with $N_{buckets}$ elements for accurate slicing to sort the buckets of the approximated partitioning.

Algorithmically, the accurate bucketing algorithm works as follows:

```
initialize the hash table for accurate bucketing
read buckets of the first-level approximation buckets in
back-to front order
for each non-empty buckets of first distribution- b1[i]
{
        for each point mapped to bucket b1[i]- p1[j]
                {
                compute projected distance of p1[j];
                define the index of accurate bucket
                using the projected distance as key
                value;
                insert to the bucket as first element;
                }
        read buckets of accurate buckets in back-to
        front order;
        add mapped point to general rendering list;
        clear accurate buckets for processing next
        bucket of first- level approximation;
}
render the general rendering list;
```

First, the algorithm reads all the points of the last bucket of the approximation hash table. Then, it distributes the points of the last bucket into the accurate bucket. All the points of the non-empty buckets of this distribution become the rendering-ordered slice of our point splatting. Then, it continues to read the approximation buckets in back-to-front order and generates the renderable slices.

Hash function for accurate bucketing works similarly to first-level bucketing. We only need to change the range values of the buckets, i.e., instead of P_{near} use $P_{near} + \varepsilon \cdot N_{buckets} \cdot Index_{f\_l\_bucket}$ and instead of P_{far} use $P_{near} + \varepsilon \cdot N_{buckets} \cdot (Index_{f\_l\_bucket} + 1)$, where ε is the predefined thickness of the slices.

Figure 10. Sample image, after alpha blending the first 6100 slices of the 9900 total slices. The light source was located on the front-right side.

The main advantages of our sorting are as follows:

(1) It works in linear time O(n). The algorithm reads all the data sets once and the culled data set twice to produce the sorted slices.

(2) It generates high-quality slicing using a small amount of extra memory. During sorting it eliminates all empty slices.

In Figure 10 we show the rendered image after slicing the parts of the accurate slices.

4.3 Shading on GPU

In the past, Z-buffer graphics were configurable but not programmable. Graphics architectures are now highly programmable. High-level languages for the GPU encapsulate all of the computations for shaders in one piece of code. The point is one of the most suitable renderable objects in the GPU, since it can carry many useful attributes and points are naturally parallel. With rendering-oriented attributes, GPU shader programs can perform light calculations for different surface materials, mix the Phong shading with volume visualization, apply different textures to each point, and compute 2D rotations to the texture coordinates. We have implemented a texture-based point splatting renderer in the GPU. Figure 11 shows the shading pipeline of our rendering engine. The GPU shader requires four components: point position, normal vector, color vector, and attribute vector for each point splat. All rendering-oriented attributes are embedded in the attribute vector. Before rendering we need to bind all uniform parameters and textures (e.g., look-up tables, texture image for splatting) to the shader program. Depending on the varying parameters, the shader performs a series of geometric transformations such as defining the surfel orientations, sizing the texture mapping and normal interpolation. Then, it makes the shading of each point using the results of the transformation operation, color values and additional rendering values in the lookup table. The shading operation generates the different RGB and alpha values according the point splat types. Algorithmically, the point splatting in the vertex shader proceeds as follows:

```
for each point splat p[i]
{
        compute splat orientation;
        define the address of texture mapping;
        project to the screen space;
        shade the splats;
        make shadow generation;
        set texture for splatting;
        make splatting;
}
```

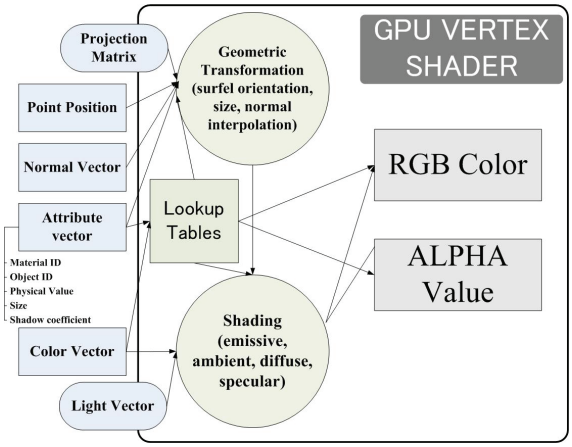


Figure 11. The GPU shader for advanced point splatting

In each step of the shading calculation, the shader gets the specific coefficients from the lookup tables. For example the shading operation uses the material lookup table, distance value and object lookup table to get specific coefficients for the Phong Shading and the shadow generation uses the shadow coefficient and the object lookup table for translucent shadow generation.

In this implementation we did not use EWA filtering. We use several ready-to-use Gaussian kernel textures for splatting. The use of ready-to-use textures reduces the computational time, but requires very careful design of good texture. It is not possible to switch textures on a per-point basis while rendering. Thus, it

is necessary to combine different textures into one 3D texture.

5. Implementation and results

In this study, our goal was to illustrate the basic capability of our point splat rendering. In the future, we plan to optimize all the steps for high performance and high quality-rendering.

The algorithms introduced in this paper are implemented in C++ and Cg shading languages. Images are rendered on a 3 GHz Pentium IV with an NVIDIA GeForce 6600 graphics card. We use OpenGL and its extensions for implementation of the vertex texture, multi target rendering and the Shader 3.0 model of the programmable vertex and fragment processing.

To demonstrate the potential for point rendering of mixed data sets, we selected several simple polygon, volume and point models. For the experiment, we used the Stanford Bunny and Dragon polygon model from the Stanford 3D Scanning Repository and the volume data of a baby, which is available in the public domain; we use the range scanner to take the 3D point cloud of our faculty building; and the particle model of clouds generated by Takeshita [32]. We generated simple volume and surface models of grass, tree and basin using the implicit functions. Using the interactively designed transfer function, we defined the color and opacity values of the cloud model. First, we used our point rendering algorithm to generate simple mixed images (see Figure 12).

Figure 13 illustrates the benefits of the mixing rendering volume data with opaque and translucent surfaces for rendering simple objects such as an artificial eye. Medical imaging applications also can use mixed rendering (see Figure 14). The translucent shadow mapping method is used for self shadow generation and for surface extraction (see Figure 15). Also, we can show the components of mixed rendering and an effective combination for the visualization of outdoor scenes. (see Figure 16). In Figure 16, we used the 3D point cloud and RGB texture data obtained by the range scanner to render the building. Therefore, the image of the building has some strange lighting effect derived from the original scanned RGB data. The all images are rendered to the framebuffer (screen) with resolution 600x500 pixels. By rendering those scenes we attempt to demonstrate the following features of our point splatting system:

- This system can render polygon models (bunny, dragon), volume models (CT scanned volume data of a baby), point clouds (3D point cloud of building obtained by the range scanner), particle models (cloud) and implicit surfaces (basin, fountain, tree, and grass).
- Our system can be used for effectively visualization of the mixed scenes (see Figures 12, 16).
- It can be used in different application areas such as medical applications (see Figure 14), mixing real word objects with artificial objects (see Figure 16), visualization of translucent surfaces (see Figure 12 (b)) and volumetric and surface visualization of implicit surfaces (see Figure 13).
- TSM method can be used for shadow generation (see Figures 7, 17) and for point based isosurface extraction (see Figure 15(a)).
- Translucent and surface graphics increases visual inspection of images (see Figure 14).
- Our point splatting system can render the unstructured point data (point and particle), implicit and polygon models in Figures 12,13,15, and regular grid data (volume models in Figures 14, 15). We rendered regular grid volume data by converting them to the unstructured point data.

The performance of our point splatting, translucent shadow mapping, and hierarchical bucket sorting algorithms is summarized in Table 1. We tested the performance by processing three different types of point clouds all opaque points (model of dragon), all translucent points (model of clouds) and mixed opaque-translucent points. We achieve the best performance when rendering all opaque points. Rendering translucent object gives the worst performance. The result shows that hierarchical bucket sorting and GPU based shading perform at an interactive rate in for million points. These algorithms work in linear time O(n) for both opaque and translucent points. Specially, the GPU based shading algorithm works almost nine times faster than its CPU analogue. It would be more impressive, if we used a more powerful graphics card. But the translucent shadow generation algorithm requires more time for mapping objects with high-level transparency (e.g., cloud). Because it uses insertion sorting for mapping points in each particular direction, the TSM shows the worst $O(n^2)$ time when all points are located along one ray direction from the light source and they are almost transparent. In mixed rendering the TSM shows its best time or near O(n) performance, when opaque elements are located in the front layers to the light source. As shown in Table 1, the TSM time (0.0853/437645) of the opaque dragon model is 1.75 times faster than the TSM time (0.2088/624614) of the translucent cloud model.

Table 1. Performance of advanced point splatting

Model	Dragon (opaque)	Cloud (translucent)	Outdoor scene with fountain
Number of points	437645	624614	1841320
Translucent Shadow Mapping (sec)	0.0853	0.2088	0.3865
Sorting (sec)	0.0522	0.0785	0.1902
Shading (sec)	0.0243	0.0352	0.0876
Rendering	5.2 fps	2.7 fps	1.5 fps

6. Conclusion and future work

The challenge of realistic interactive visualization of complex physical models opens up many new research directions in rendering, classification, and interpolation of the physical properties of points, for adaptive transparency control and for fully photorealistic depictions.

Point rendering has been shown to be an effective method for the display of mixed 3D data. We believe that our work will contribute to future research in the field of computer graphics for several reasons. First, it is the first attempt to build a universal rendering engine to render mixed data sets, volume and polygonal primitives. Second, translucent mapping and bucket sorting can be used with any other rendering type as a high-speed method for accurate mapping of shadows, light, and collision detection. Third, all the methods used in our paper can be utilized by GPU-based algorithms.

In spite of its advantages, point splatting suffers several drawbacks.

(1) The chief problem is associated with the memory requirements of point-type 3D data. To demonstrate the ability of our rendering system we use the thousands of points for each 3D model. The large amount of point data affect to the rendering speed. Also the numerous rendering-oriented attributes absorb a large amount of memory.

(2) The next set of problems is related to image quality. The image quality of point rendering is mostly depends from quality of models. In our demonstration, we mixed the high quality models (dragon, bunny, cloud, buildings) and the simple models (grass, tree, basin). Those results show us that we need to do more research on the image space EWA filtering and Phong splatting [3] for point geometry with modifications suitable for a transparent object. We will also need to use advanced lighting technologies, such as radiosity for global illumination, without losing the advantages of point rendering.

(3) Other issues are related to the optimization of the preprocessing steps. In general, all preprocessing algorithms are implemented on the CPU by many researchers, but most of them are not optimized for the GPU implementation.

For the future improvement, we are working on the following research areas.

- Research on effective point conversation techniques for polygon and unstructured grid data.
- Improvement of quality and performance of TSM
- More research on additional rendering-oriented attributes for realistic point visualization and the compression methods of attributes. We also should investigate the question of what additional information we have to store with each point to accelerate spatial search.
- GPU implementation of visibility sorting and TSM
- Implementation of advanced illumination techniques within TSM

Acknowledgement

This work was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research (B) 16300021.

References

[1] M. Levoy: “Display of Surfaces from Volume Data,” IEEE Computer Graphics and Applications, Vol. 8, No. 3, May, 1988.

[2] M. Botsch, A. Hornung, M. Zwicker, L. Kobbelt: “High-Quality Surface Splatting on Today's GPUs,” Eurographics Symposium on Point-Based Graphics 2005.

[3] M. Botsch, M. Spernat, L. Kobbelt: “Phong Splatting,” Symposium on Point-Based Graphics 2004, pp. 25-32, 2004.

[4] M. Zwicker, H. Pfister, J. van Baar, M. Gross: “Surface Splatting,” SIGGRAPH 2001.

[5] L. Westover: “Interactive Volume Rendering,” Proceedings Volume Visualization, pp. 9-16. May 1989.

[6] M. Zwicker, H. Pfister, J. van Baar, M. Gross: “EWA Volume Splatting,” In IEEE Visualization (VIS), pp. 29-36, October, 2001.

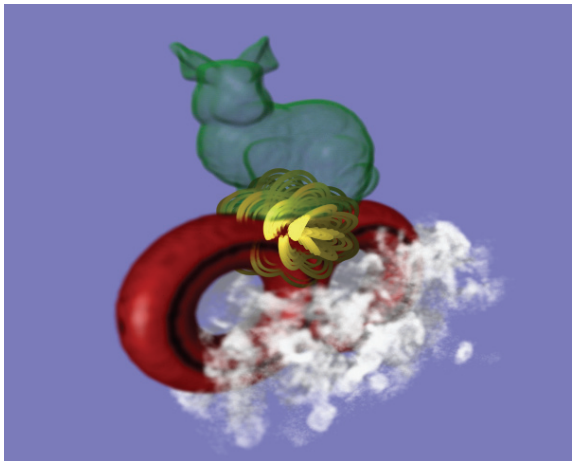
[7] W. Chen, L. Ren, M. Zwicker, and H. Pfister: "Hardware-Accelerated Adaptive EWA Volume Splatting," in Proceedings of IEEE Visualization 2004, pp. 67-74, 2004.

[8] L. Williams: “Casting Curved Shadows on Curved Surface,” Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques, pp. 270-274, August 23-25, 1978.

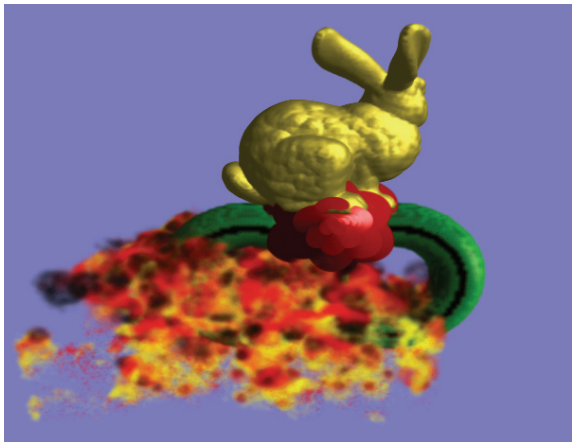
[9] F. Crow: “Shadow Algorithms for Computer Graphics,” Computer Graphics, 11(2), pp. 442-448, Summer 1977.

[10] M. Hasenfratz, M. Lapierre, N. Holzschuch. and F. Sillion: “A Survey of Real-Time Soft Shadows Algorithms,” Eurographics 2003.
[11] C. Dachsbacher, M. Stamminger: “Translucent Shadow Maps,” Eurographics Symposium on Rendering 2003.
[12] C. Everett: “Order-independent Transparency,” NVIDIA, 2001.
[13] T. Locovic, E. Veach: “Deep Shadow Maps,” Proceedings of ACM SIGGRAPH 2000.
[14] N. Govindaraju, M. Hensen, M. Lin, and D. Manocha: “Interactive Visibility Ordering and Transparency Computations among Geometric Primitives in Complex Environments,” Symposium on Interactive 3D Graphics, 2005.
[15] S. P. Callahan, M. Ikits, J. Luiz Dihl Comba, C. T. Silva: “Hardware-Assisted Visibility Sorting for Unstructured Volume Rendering,” IEEE Trans. Vis. Comput. Graph. 11(3): pp. 285-295 (2005).
[16] P. Kipfer and R. Westermann: “Improved {GPU} Sorting,” GPUGems 2, Chapter 46, 2005, Addison-Wesley, pp 733-746.
[17] P. Kipfer, M. Segal, R. Westermann: “UberFlow: A GPU-Based Particle Engine,” Graphics Hardware 2004.
[18] R. Yagel, D. Reed, A. Law, P.-W. Shih, and N. Shareef: “Hardware Assisted Volume Rendering of Unstructured Grids by Incremental Slicing,” Proc. 1996 Volume Visualization Symp., pp. 55-62, Oct. 1996.
[19]. C. Prashant, J. Meyer: “Incremental Slicing Revisited: “Accelerated Volume Rendering of Unstructured Meshes,” Proceedings of IASTED Visualization, Imaging, and Image Processing 2002, Ma'laga, Spain, Sept. 9-12, 2002.
[20] P. Bunyk, A. Kaufman, and C. Silva, "Simple, fast, and robust ray casting of irregular grids," in Proceedings of the Dagstuhl'97 - Scientific Visualization Conference, pp. 30--36, 1997.
[21] M. P. Garrity, ``Raytracing Irregular Volume Data", Computer Graphics, 24: pp. 35-40, 1990.
[22] M. Weiler, M. Kraus, M. Merz, T. Ertl, “Hardware-Based Ray Casting for Tetrahedral Meshes.”, pp. 333-340 IEEE Visualization 2003
[23] D. King, C. Wittenbrink, H. Wolters, “An Architecture for Interactive Tetrahedral Volume Rendering”, pp. 101-110, International Workshop on Volume Graphics 2001.
[24] M. Kraus, W. Qiao, D. S. Ebert. “Projecting Tetrahedra without Rendering Artifacts.” in Proceedings IEEE Visualization 2004, pp. 27-34, 2004
[25] P. Shirley and A. Tuchman. ”A polygonal approximation to direct scalar volume rendering.” in 1990 Workshop on Volume Visualization, pp 63--70, San Diego, CA, December 1990
[26]. J. Comba, J. Klosowski, N. Max, J. Mitchell, C. Silva, and P Williams. “Fast polyhedral cell sorting for interactive rendering of unstructured grids.” Computer Graphics Forum, Vol 18, No 3, pp 369--376, September 1999.
[27]. R. Cook, N. Max, C. Silva, and P. Williams. “Image-Space Visibility Ordering for Cell Projection Volume Rendering of Unstructured Data.” IEEE Transactions on Visualization and Computer Graphics, Vol .10, No. 4, 2004
[28] M. Levoy: “A Hybrid Ray Tracer for Rendering Polygon and Volume Data,” IEEE Computer Graphics and Applications, vol. 10, no. 2, pp. 33-40, 1990.
[29] K. Kreeger and A. Kaufman: “Mixing Translucent Polygons with Volumes,” IEEE Visualization 1999.
[30] S. Roettger, M. Kraus, and T. Ertl: “Hardware-Accelerated Volume and Isosurface Rendering Based on Cell-Projection,” In Proc. Visualization '00, pp. 109-116. IEEE, 2000.
[31]. M. Ferre, A. Puig, D. Tost: “Using a Classification Tree to Speed up Rendering of Hybrid Surface and Volumes Models,” WSCG 2004: pp. 105-112.
[32] D. Takeshita, T. Fujimoto and N. Chiba; “Recursive Particle Generator for Animating Plume Fluid,” Proceeding of IWAIT, 2005.

(a)

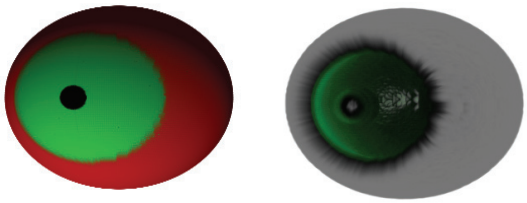


(b)



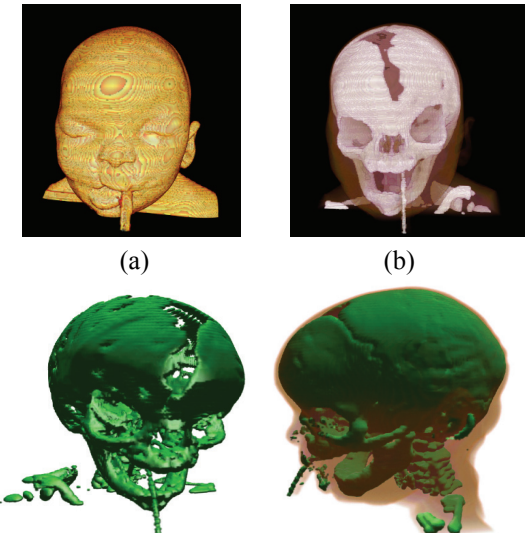
(c)

Figure 12. Visualization of mixed volume, point and polygon data. (a) mixed rendering (b) mixed object with different level of translucency (c) shading with different material values



(a) (b)

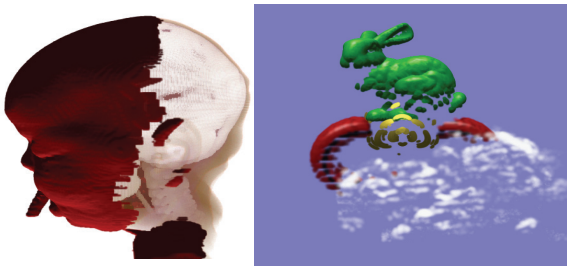
Figure 13. Rendering of simple artificial eye. (a) surface rendering (b) mixed rendering



(a) (b)

(c) (d)

Figure 14.Comparison of volume and surface rendering of medical data (a) volume rendering, (b) translucent volume rendering (c) surface extraction (d) mixed rendering



(a) (b)

Figure 15. Sample usage of the translucent shadow mapping table (a) translucent shadow mapping for surface extraction (b) translucent shadow mapping for generating self shadow

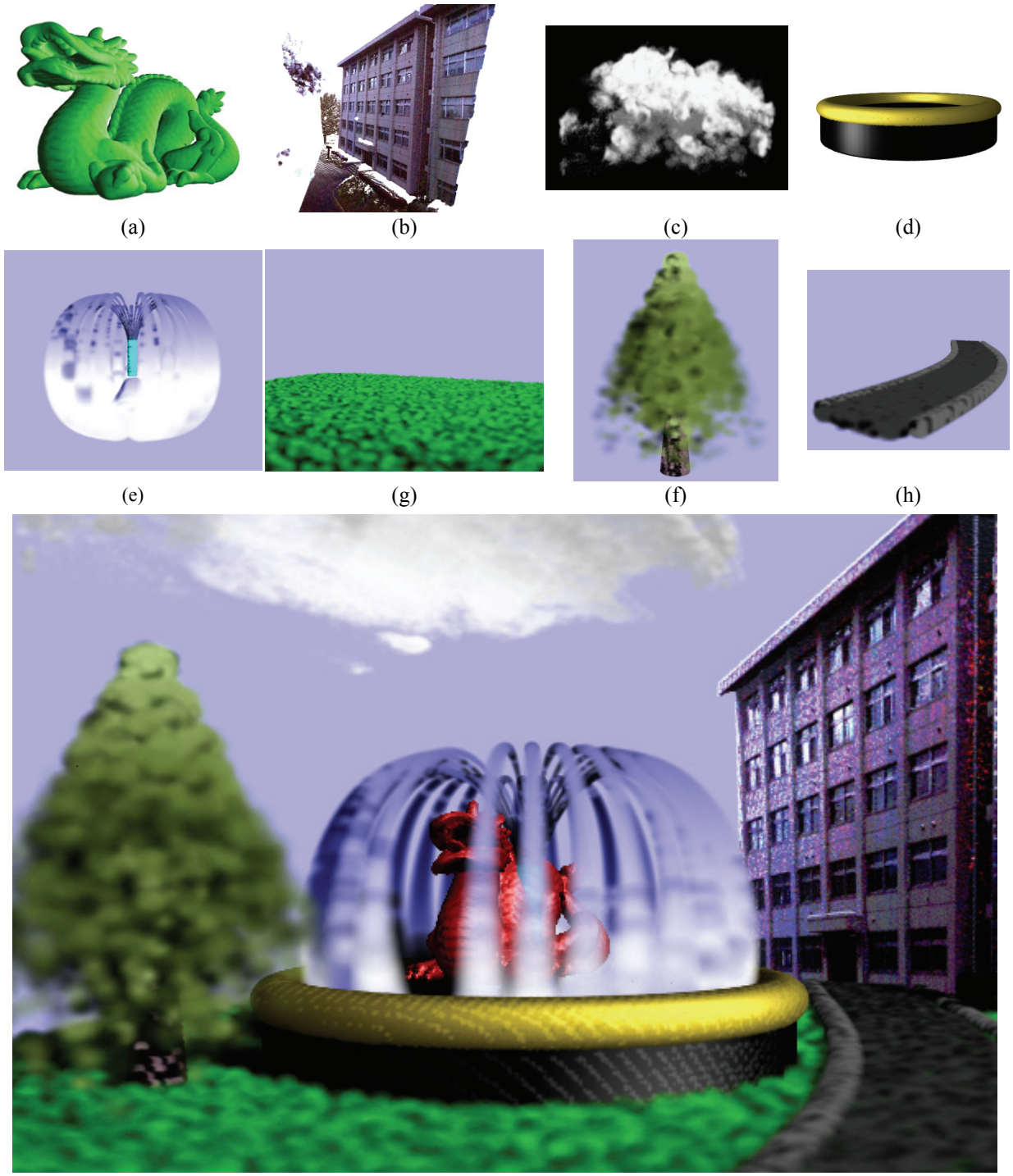
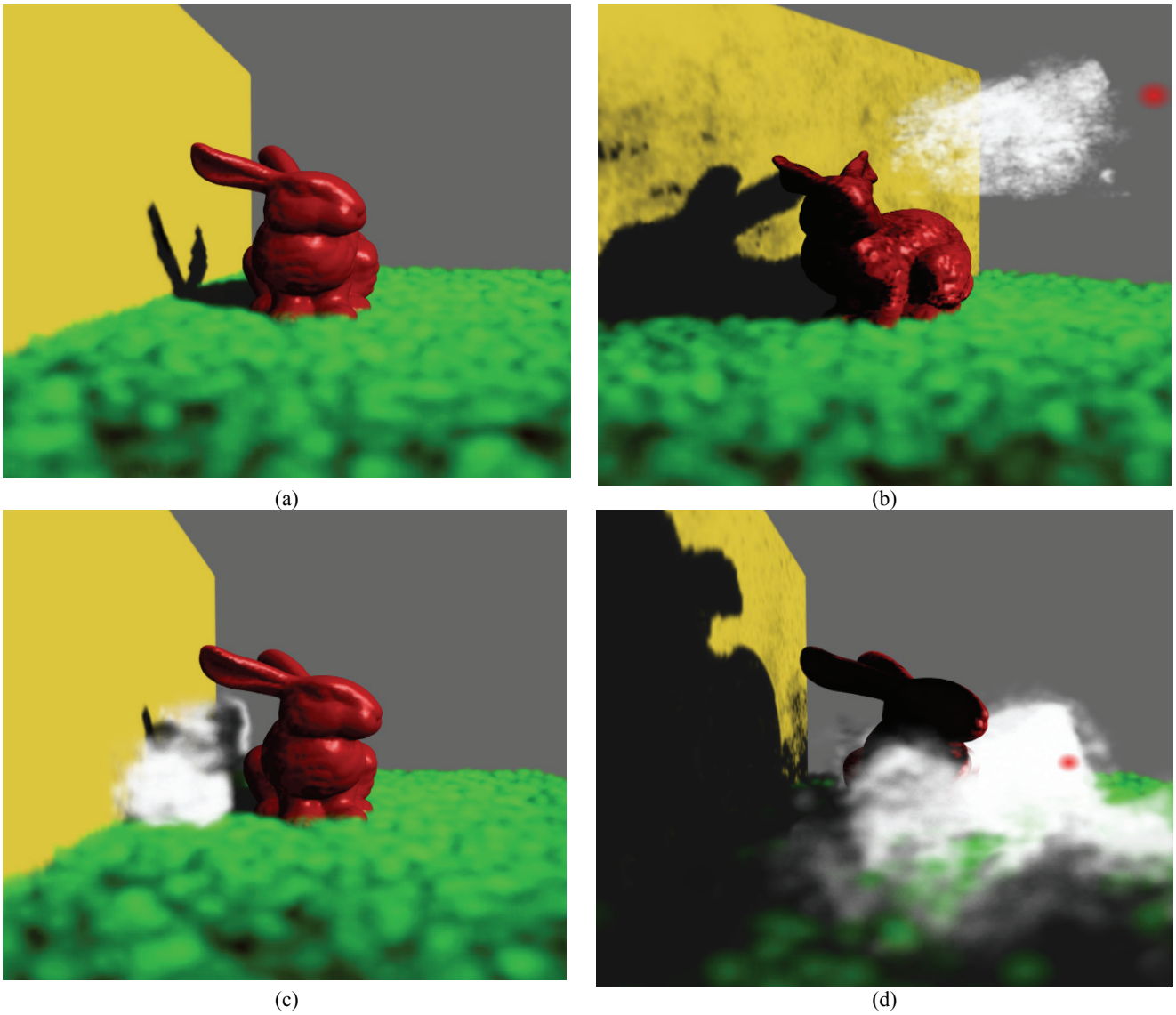


Figure 16. Mixed rendering of artificial outdoor scene. (a) polygon model of dragon (b) 3D point cloud taken by range scanner (c) particle model of cloud (d) point based surface model generated by algebraic equation (e-h) simple volume models generated by implicit functions – fountain, grass, tree and road (h) mixed rendering of point, polygon and volume data.



(a) (b)

(c) (d)

Figure 17. Different situations were translucent and opaque objects located in different order from the light source. (a) shadow of opaque objects, (b) the translucent object is located between the light and opaque object (c) the opaque object is located between the light and translucent objects (d) the light is located inside the translucent object. The location of the light source is marked by red spot.