MULTIRESOLUTION RAY TRACING FOR POINT-BASED GEOMETRY

MOHAMED NORDIN BIN ZAKARIA

UNIVERSITI SAINS MALAYSIA
2007

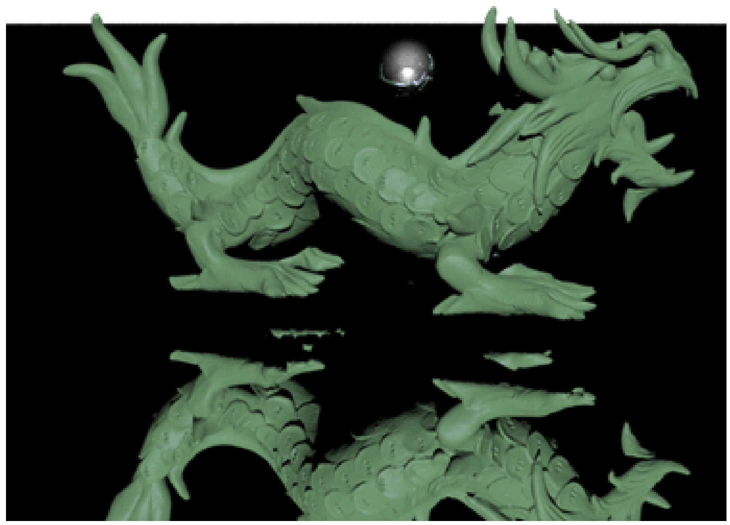
MULTIRESOLUTION RAY TRACING FOR POINT-BASED GEOMETRY

by

MOHAMED NORDIN BIN ZAKARIA

Thesis submitted in fulfillment of the
requirements for the degree
of Doctor of Philosophy

April 2007



Acknowledgements

- To all who has influenced my life directly or indirectly, knowingly or unknowingly…BUT
  - Very special thanks to Dr Bahari Belaton and Dr Abdullah Zawawi for accepting me as a graduate student, and for their advices, superb supervision, and warm friendship.
  - Special thanks to Prof Hans-Peter Seidel for giving me a chance to gain experience in one of the research hubs of Europe.
  - Special thanks to UTP for the support and the facilities.
  - Thanks to my parents for their patience in waiting for the completion of my seemingly endless pursuit of a PhD.
  - Thanks to Eva for helping me to discover myself.
  - Thanks to my little Sohail for having been my little well of joy!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMNS

SJ	Schaufler and Jensen Ray-Surface Intersection Scheme
PSS	Point Set Surface Ray-Surface Intersection Scheme
RFD	Ray-Facing Disk Ray-Surface Intersection Scheme
LoD	Level of Detail
MLS	Moving Least Square
PCA	Principle Component Analysis
BVH	Bounding Volume Hierarchy
NURBS	Non-Uniform Rational B-Spline
GPU	Graphics Processing Unit
PBRT	Physically Based Ray Tracer
CMM	Coordinate Measuring Machine

PENYURIHAN SINAR BERBILANG PELERAIAN UNTUK GEOMETRI BERASASKAN TITIK

ABSTRAK

Tumpuan utama di dalam tesis ini adalah kajian tentang integrasi teknik berbilang peleraian dengan penyurihan sinar di dalam menjanakan imej objek-objek 3D berasas titik. Sejak kebelakangan ini, terdapat keperluan untuk model-model 3D yang semakin meningkat kekompleksan geometrinya. Ini telah menyebabkan penggunaan yang lebih berleluasa teknologi pengimbasan 3D. Teknologi ini berupaya mengimbas sesuatu model fizikal yang kompleks untuk menghasilkan sesuatu set titik yang padat dan tidak berstruktur. Set ini mengandungi banyak maklumat. Walaubagaimanapun, kajian di dalam persepsi manusia menunjukkan tidak semua maklumat berkenaan dapat dilihat atau diproses oleh seseorang. Maka teknik berbilang peleraian menawarkan peluang untuk mengurangkan beban komputasi yang terlibat di dalam melakukan penyurihan sinar bagi set data berkenaan. Sumbangan pertama tesis ini di dalam perkara ini adalah penggunaan struktur data Hierarki Isipadu Kongkongan untuk tujuan penyurihan sinar. Struktur data yang digunakan mengawal tahap perincian yang digunakan di dalam penjanaan imej, menghapuskan permukaan terhadang dan menggunakan kombinasi isipadu berbentuk sfera dan berbentuk kotak untuk mencapai kelajuan penyurihan sinar yang lebih baik. Sumbangan yang kedua tesis ini adalah kajian tentang penggunaan teknik berbilang peleraian bersama-sama dengan struktur data pokok kd yang dioptimakan kosnya. Struktur data

yang terhasil berupaya meningkatkan prestasi penyurihan sinar dengan cara mengawal dengan efisien tahap perincian yang digunakan di dalam penjanaan imej dan menggunakan sesuatu varian yang baru skema persilangan sinar-permukaan. Hasil daripada kedua-dua kajian ini menunjukkan yang teknik-teknik yang dipelopori berupaya menghasilkan imej yang berkualiti dan meningkatkan prestasi penyurihan sinar.

MULTIRESOLUTION RAY TRACING OF POINT-BASED GEOMETRY

ABSTRACT

The primary concern in this thesis is with the incorporation of multiresolution-based optimization into ray tracing algorithms specially tailored for point-based geometry. In recent years, increasing demand for model complexity has led to an increasing use of 3D scanning technologies capable of digitizing a complex physical model into a dense, massive and unstructured point cloud. Despite the dense amount of information contained in this data set, work in human perception study has shown that not all of it will be perceptible by a human viewer. Hence multiresolution technique offers an opportunity to reduce the computational workload involved in ray tracing such data set. In this respect, the first contribution in this thesis is the adaptation and enhancement of a Bounding Volume Hierarchy data structure in order to allow for faster ray tracing. The resulting data structure incorporates an efficient Level-of-Detail control and backface-culling optimization, and uses a mixture of bounding spheres and boxes to enable faster ray tracing. The second contribution in this thesis is an approach for incorporating multiresolution-based optimization into a point-based geometry ray tracer that is already optimized by use of a cost-optimized kd-tree. The resulting data structure incorporates an efficient Level of Detail control and a new variant of ray-surface intersection scheme that improves the ray tracing performance. Both the image quality and the ray tracing performance obtained

point to the effectiveness of the multiresolution techniques introduced in this thesis.

CHAPTER 1
INTRODUCTION

1.0. Overview

In a number of works related to the perceptual aspect of 3D computer graphics system (Reddy 1997, Reddy 2001, Luebke and Hallen 2001, Howlett et al 2004), it has been shown that the details frequently generated by current computer graphics technology are often much more than what users can perceive. Multiresolution rendering is an attempt to exploit this fact to reduce the computational requirement of computer graphics application. In multiresolution rendering, one seek to render an image more efficiently by presenting to the user only what would be perceptible and thus saving computational resources that would otherwise have been wasted. In implementing multiresolution rendering, one must be able to encode and to selectively retrieve different level of details (LoD) of a scene or object during a rendering pass. Intuitively, the idea is to select a high LoD for objects or surfaces whose details are likely to be seen well by the eye and to select a lower LoD for other objects or surfaces.

Of course, multiresolution technique is inherently tied to work in human perception. Much of the perception issue relate to the selection of a LoD given a certain user state and the state of the scene or object being viewed. On the other hand, the focus in this thesis is on the algorithmic aspect of multiresolution technique. Specifically, the author is concerned with studying and designing multiresolution-capable rendering algorithm specialized for ray tracing of point-based geometry.

A point-based geometry is an object that comprises of point samples. In this thesis, it will be interchangably refered to it as a point set. In fact, frequently in this thesis, a ray tracer specialized for point-based geometry shall be frequently refered to as a point set ray tracer, rather than a point-based geometry ray tracer.

The author is interested in point-based representation as for a number of years now, points have been the representation of choice for models of very high geometric complexity or very high appearance granularity. Typically these models are obtained from laser range and optical scanners (Levoy et al 2000), procedurally generated (Stamminger and Drettakis 2001) or sampled from a polygon-based geometry (Grossman and Dally 1998, Pfister et al 2000). With a point-based representation, the surface of a 3D object is described by a set of sample points without further topological information such as triangle mesh connectivity. It has been shown before that the lack of topological information leads to simpler and more efficient rendering (Grossman and Dally 1998, Rusinkiewicz and Levoy 2000), simplification (Pauly et al 2002), level-of-detail control (Rusinkiewicz and Levoy 2000, Chen and Nguyen 2001, Stamminger and Drettakis 2001), and texturing (Pfister et al 2000) for very complex models.

The approach commonly used to directly view a point-based geometry is splatting (Pfister et al 2000, Zwicker et al 2001). In splatting, the basic idea is to iterate through the points in the point set and compute its projection onto the screen. A splatting-based point set viewer, examples of which include QSplat (Rusinkiewicz and Levoy 2000) and Pointshop (Zwicker et al 2002), can typically run at an interactive frame rate on a computer system with recent consumer graphics hardware. While, it is fast and easy to view a point-sampled geometry using splatting, it is nontrivial and expensive to use the technique to create advanced accurate lighting effects such as shadows and self-shadowing, reflection, and global illumination. On the other hand, ray tracing, being based on the simulation of light rays through a 3D environment, can quite easily model such effects.

However, ray tracing tends to be slow. The reason is that while in splat-based rendering, one projects from points onto the screen, in ray tracing, one projects from individual pixels in the screen onto the points. For each frame or image to be rendered, a ray is formed going through each pixel and intersected against the point set. Hence, while in splatting, one processes the list of points once per frame, in ray tracing, one processes it once per pixel. And since, a point-based geometry can contain millions of points and for anti-aliasing more than one ray is casted per pixel, ray tracing of such a dense geometry can take up much computational resources.

Still, animated movies (*Toy Story*, *Over the Hedge*, and virtually all others) and a great many video clips for computer games were all successfully created using methods based on ray tracing. Hence, much effort has been exerted research-wise, to improve ray tracing speed and to bring the ray tracing technique closer to the realms of interactive real-time usage on an ordinary personal computer.

Methods that have been reported in the past on improving the performance of ray tracing include using specialized spatial data structure (Glassner 1989, Havran 2001), exploiting ray coherence (Wald 2004, Reshetov et al 2005), tuning memory and cache performance (Yoon and Manocha 2006), and using parallel computation technology (Wald 2004). A spatial data structure partitions memory space into cells in order to speed up the process of traversing a ray to search for the closest intersection with a surface. Exploitation of ray coherence works because rays close together tends to hit the same portion of a surface, and hence they share parts of the computation involved. Tuning memory and cache performance can improve overall ray tracing performance especially when dealing with large data set, as if memory and cache are not managed wisely, the ensuing cache miss and disk crashing will tend to bog down any ray tracer. Finally, parallel computation is an attractive approach for speeding up ray tracing as the algorithm is quite well-known to be trivial to parallelize.

Only as recently as 2003 has there been attempts to exploit multiresolution methods in ray tracing (Stoll et al 2006, Yoon et al 2006, Christensen 2003). Multiresolution ray tracing methods reported thus far have been tailored for polygonal scenes. However point-based geometry has a different demand as it is geometrically a different representation. Hence, the author's primary motivation in this thesis is to further improve the state-of-the-art in visualization of point-based geometry by investigating a multiresolution approach to its ray tracing.

1.1. Research Scope

It should be highlighted that in this thesis, the primary concern is with the algorithmic aspect of multiresolution ray tracing. As such, only a simple model for LoD selection, the aspect of the system that is most tied up to human perception study, is used. Hence, the author does not perform any human perception study. A survey of existing work in this concern is however provided in Section 2.3.1.

Furthermore, as far as geometric representation is concerned, this thesis is focused on point-based geometry. Hence the methods investigated are specialized for point-based geometry. The author made neither substantive investigation nor strong claim as to whether the methods, results or conclusions to be reported in Chapter 3 and 4 apply for other geometrical representation such as polygonal meshes or NURBS surfaces.

1.2. Research Objectives

The work as presented in this thesis started out with a mission to improve the state of the art in ray tracing of point-based geometry. The general strategy is to improve the performance of existing data structures for ray tracing by employing multiresolution technique. Two data structures are selected. One is the bounding volume hierarchy (BVH), and the other is the kd-tree. The BVH is a simple data structure that is widely implemented in many ray tracers. On the other hand, the kd-tree is widely considered to be the state-of-the-art data structures and is implemented in high-performance ray tracing engines such as the OpenRT (Wald 2005). Hence, improving the performance of these two data structures for ray tracing would have significant impact in the area of ray tracing.

The objectives for the work presented in this thesis can then be succinctly stated as follows:

i) To improve the performance of BVH-based ray tracing of point-based geometry by using multiresolution technique.

ii) To improve the performance of cost-optimized kd-tree based ray tracing of point-based geometry by using multiresolution technique.

1.3. Contributions

Two general contributions are made in this thesis. First, it is shown how multiresolution method can be used to improve the performance of a BVH-based ray tracer. While the incorporation of multiresolution method into a BVH-based ray tracer is by itself a novelty, the author notes the following specific contributions:

- Adaptation of a technique from splat-based rendering to compute the level-of-detail information stored in the nodes of the BVH tree,
- Adaptation of a technique from splat-based rendering to use the level-of-detail information stored in the nodes of the BVH to perform backface culling, and

- The use of non-spherical shapes to bound nodes which is unlikely to project to a pixel area on the screen smaller than a predefined threshold value.

Secondly, a method is presented for integrating multiresolution method into a ray tracer that has already been accelerated by use of a cost-optimized kd-tree. While this integration is by itself a novelty, the author notes the following specific contributions:

- Adaptation of a technique from splat-based rendering to compute the level-of-detail information stored in the nodes of the kd-tree, and
- A new variant of ray-surface intersection scheme that is faster compared to existing methods.

1.4. Thesis Organization

The rest of this thesis is organized as follows: In Chapter 2, a sampling of related background material is provided. In Chapter 3, work on integrating multiresolution capability into a ray tracer based on bounding volume hierarchy is discussed. In Chapter 4, a study on integrating the capability into a ray tracer already accelerated by a cost-optimized kd-tree is presented. Finally in Chapter 5, the overall work is discussed and the thesis concluded.

CHAPTER 2
BACKGROUND

2.0. Introduction

In this chapter, a sampling of background materials relevant to work on multiresolution ray tracing of point-based geometry is provided. In particular, the author presents in this chapter an overview of point-based representation, an overview of the ray tracing algorithm, and that of multiresolution technique.

2.1. Point Based Geometry

The idea of using points as the basis geometric representation dates back even before the age of 3D scanning technologies. Points became popular in particular with the introduction of particle systems. Early works involving this primitive include the modeling of smoke (Csuri et al 1979), clouds (Blinn 1982), fire (Reeves 1983), and trees (Smith 1984). In 1985, Levoy and Whitted (1985) proposed points as a universal modeling primitive and presented algorithms allowing for anti-aliased rendering. The use of points for non-fuzzy models, however, reemerges in recent years, and this is especially attributed to the need to deal with massive data set gathered from 3D scanning devices. These 3D scanning devices use various technologies to digitize a physical object. These technologies are reviewed in the following subsection.

2.1.1. 3D Model Digitization

There exist a variety of approaches to digitize the shape of an object. Whatever the approach, the shape is typically acquired as a set of coordinates corresponding to points on the object's surface. These coordinates measure the distance or depth of the point from a measuring device, and are called range values. The measuring device is accordingly called a rangefinder and the data acquired is called range data. Three dominant classes of acquisition technologies will be briefly discussed: stereo imaging, Coordinate Measuring Machines (CMM) and optical triangulation.

In stereo imaging (Szeliski 1999), the idea is to capture 2D images of objects from different viewpoints and to use the known camera coordinates in each case to reconstruct the shape of the objects from the photographs. Automatic reconstruction addresses issues like finding corresponding objects in different images (correspondence), telling objects apart from each other (segmentation), identification of similar areas (region detection) and identification of boundaries (edge detection).

CMM (Bosch 1995) on the other hand, takes a brute force approach by having mounted, movable touch probes scan the entire surface of an object. The movement of the probe with respect to a reference point is tracked, so that the location of a contact point can be calculated. CMMs are precise and accurate. These have, in fact, led to it being the industry standard for manufacturing applications. However, the machinary needs a human operator, the handling clumsy, and the scanning process is typically slow.

In optical triangulation (Venuvinod et al 2003), a light source, typically laser, projects light onto a surface. A sensor then catches the reflected light rays and determines their direction. As the positions of the light source, the sensor, and the direction of projection are known, the point of intersection of the projected and reflected rays can be found. The intersection point gives the range value for the surface point that the projected ray hit. By translating or rotating the surface through the beam, or by sweeping the beam across the surface, one can then acquire the range data for the entire object.

Whatsoever the method deployed in digitization, one has to note that conceptually, in the same way that pixels form the digital elements of 2D images, the point samples acquired are the atomic units that collectively describes object geometry and appearance. In fact, a point set is, mathematically, a piecewise constant surface approximant. What this means is that the point set approximates a surface in a linear way, and that the quality of the approximation is proportional to the average spacing h between the point samples p_i. Hence, the approximation error of a point set is of the order $O(h)$ (Davis 1975). Consequently, the number of point samples required to cover a surface is proportional to the surface area.

2.1.2. Computing Local Neighborhood

For a number of modeling and rendering operations, information on the local neighborhood of each sample point in a point set is needed. For example, to compute the normal vector for a sample point, one needs to gather information about points in its local neighborhood. There is, however, no connectivity information explicitly stored in a point-sampled geometry. Hence, given a sample point, one cannot directly compute its neighbors.

There two possible approaches in which these local neighborhoods may be constructed; using Euclidean neighborhoods or using k-nearest neighbors. In the first approach, all the point samples within a certain radius around a query point are defined to be its neighbors. The output from this method is however dependent on the sampling density of the surface, as noted by Amenta et al (1998). There may be too many or too few neighbors found within a particular region of a surface. Furthermore, as also noted by Amenta et al (1998), the neighborhood estimate would be erroneous if two separate surfaces or regions of a surface were located spatially close to each other.

On the other hand, if the surface is sufficiently dense, and the sampling satisfies certain sampling criteria, especially adaptation to the local feature size, the second approach - k-nearest neighbors - provides reliable neighborhood information. In fact, Amenta et al (1998) provides a formal proof for the stability of the neighborhood estimate if the sampling criteria is fulfilled.

The k-nearest neighborhood can be computed efficiently by using a hierarchical space partitioning technique such as kd-tree. The kd-tree in this context is a multidimensional search tree for points in k dimensional space. For the purpose of ray tracing, the value of k is 3. One may understand the kd-tree data structure to be an extension of the binary search tree. In a traditional binary search tree, records are defined by only one key. In a kd-tree for point based geometry, records are defined by *3* keys, corresponding to the x,y and z coordinates of point samples within the geometry. Similar to a traditional binary search trees, records are inserted and returned using relational operators namely *less than* (<) and *greater than* (≥) operators. However, in searching through the kd-tree, the key that determines the subtree to use (i.e. left or right) varies with the level in the tree. At level L, key number $L \bmod 3 + 1$ is used, where the root is at level 0. Therefore, the first key (x coordinate) is used at the root, the second key (y coordinate) at level 1, and the third key (z coordinate) at level 2. The search process reverts to the first key at every 3 levels.

2.1.3. Computing Normal Orientation

The normal vector of a point on a surface indicates the orientation of the surface at that point. This vector is needed, for example, when computing the amount of light reflected from the point in consideration. Hence, it is important to consider how one would compute normal vectors for points in a point-based geometry.

The mechanics of computing the normal vector for points or vertices in a triangle mesh is well-known. Given a vertex in a triangle mesh, the first step is to compute the normals of the triangles adjacent to it. The normal of a triangle is computed by taking the cross product of two non-collinear vectors on its plane. The second step is to compute the normal vector for the vertex in concern by taking a weighted average of the normal vectors of the adjacent triangles.

For a point-sampled geometry, there is no edge connectivity information as is the case for a triangle mesh. Instead one first computes the local neighborhood of a point and then one applies Principal Component Analysis based on information in that neighborhood to derive the normal vector for that point. Let p_o be a sample point and $\{p_1, \ldots, p_k\}$ its nearest neighbors. The covariance matrix is given by

$$C := \sum_{i=0}^{k} (p_i - \overline{p})(p_i - \overline{p})^T \in \Re^{3\times 3} \qquad \text{……………….. Equation 2.1}$$

where

$$\overline{p} = \sum_{i=0}^{k} p_i / (k+1) ,$$

The covariance matrix *C* is symmetric and positive semi-definite. A symmetric matrix, A, is a square matrix that satisfies:

$$A = A^T \qquad \text{……………….. Equation 2.2}$$

where A^T is the transpose of A.

A positive semidefinite matrix is a square matrix satisfying certain properties and whose eigenvalues are all nonnegative. For a detailed exposition of this concept, the reader is referred to (Lang 1997). For the purpose of this thesis, it suffices to say that the properties of the matrix C is such that the eigenvector corresponding to its smallest eigenvalue gives an estimate for the normal direction, and hence the normal vector.

Note that this determines the normal vector up to its sign only – the normal vector may be oriented either inward or outward. A consistent orientation over all sample points may be constructed by propagation along a minimum-spanning tree as done in (Hoppe et al. 1992).

2.1.4. Splatting Point Samples

An important task in point-based Computer Graphics is that of rendering a point-based geometry so that it appears to comprise of smooth continuous surfaces rather than a discrete collection of points.

The focus in this section is on rendering via splatting. The simplest possible approach to splat is to simply render the point set as a collection of closely spaced point primitives viewed orthographically. This can be done using a programming library such as OpenGL. Dense sampling is required due to the insufficient object-space approximation power of purely point-based representations.

The earliest systematic account of how one may render points for objects traditionally represented by polygons is that by Levoy and Whitted (1986). Levoy and Whitted delved into detailed discussion on the concept of ‘splatting’. They noted that splatting – the projection of point samples onto the image plane – require that each projected point contributes to more than one pixel.

Not much was heard again about the work of Levoy and Whitted until 1998, when Grossman and Dally again proposed the idea of decomposing 3D models into point samples. Instead of a completely unorganized point cloud, they used a set of depth images that are orthogonally sampled from a given input geometry, as illustrated in Figure 2.1. Each pixel in each depth image is a surface sample containing geometric position and (view independent) surface color. To prevent gaps in images rendered due to the discrete nature of the point samples, Grossman and Dally (1998) proposed the use of a multi-layered depth buffer.

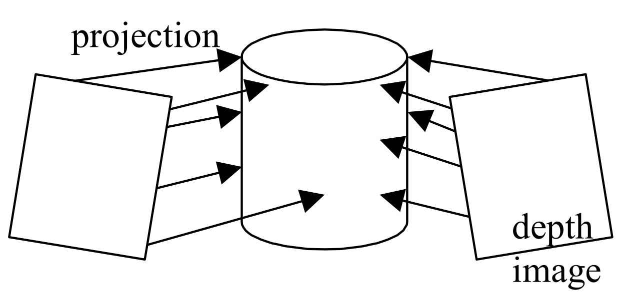


Figure 2.1: Grossman and Dally (1998) Point Samples Projection

Zwicker et al (2001) visited again the concept of splatting. They called points on the surface of the sampled geometry a surface splat. In the basic form, a point *p* has a normal vector *n* and a radius *r*, and could thus be described as an object-space circular disk. To better deal with the curved surfaces, elliptical splats, instead of circular splats, are used. Figure 2.2 illustrates both circular splat and elliptical splat. The two attributes defining an elliptical splat are namely its two tangential axes *u* and *v* and the respective radii. If the two axes coincide with the principal curvature directions of the underlying surface, and the radii are inversely proportional to the minimum and maximum curvatures, then the local approximation attained is optimal.

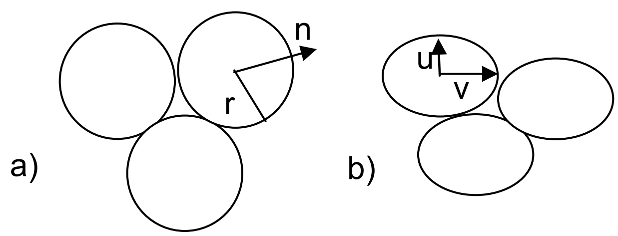


Figure 2.2: a) Circular Splats and b) Elliptical Splats

Kobbelt and Botsch (2004) discussed a number of properties of splat-based surface representation. They drew from differential geometry to note that elliptical splats form the best local approximant to a smooth surface. They further compared between splat-based and triangle mesh representation. A splat-based representation is similar to a triangle mesh representation in that each individual splat is a piecewise linear surface primitive. Hence, a splat-based representation provides the same quadratic approximation order as a triangle mesh representation. Further, just as for a triangle mesh representation, the sampling density for a splat-based representation of a surface can be adjusted according to the surface curvature. This adjustment is such that highly detailed regions are sampled with a higher density of splats, while flat surface regions are sampled more sparsely. A notable difference between a splat-based representation and a triangle mesh representation is that a triangle mesh representation has C^0 continuity while a splat-based representation need not be so. A splat-based representation however, is C^1 continuous. Hence, it is able to approximate a surface and yet at the same time provides the same topological flexibility as pure point clouds. In all, Kobbelt and Botsch (2004) noted that elliptical splat-based representation is a form of surface representation better than triangle meshes.

A primary limitation of splats is in representing sharp features, such as edges or corners in an engineering model. For splat-sampled surfaces, insufficient sampling lead to alias artifacts, and in many cases, such artifacts cannot be removed by simply increasing the sampling density (Kobbelt and Botsch 2001). Kobbelt and Botsch (2001) presented a way to solve the problem by aligning the sampling with the principal curvature directions of the underlying surface. Pauly et al (2003) further showed that if surface splats are to represent a sharp feature, all splats that sample the feature have to be clipped against one clipping line if the feature to be represented is an edge or two clipping lines if the feature to be represented is a corner.

Another approach to the aliasing problem is by using the moving least-squares (MLS) technique proposed by Levin (1998). The MLS technique interpolates a given set of point samples using local higher order polynomials and has been applied to point-based methods by Alexa et al. (2001). An MLS surface is defined by using a projection operator that projects points from a vicinity *B* of the MLS surface onto the surface itself.

MLS surfaces can be used to define a smooth surface from a set of points, and have been shown by Alexa et al (2001) to be versatile as a tool to generate additional sample points on a point sampled surface, e.g., for up- or down-sampling a model, for low-pass filtering it, or for mapping points back onto the initial surface after local restructuring. However, rendering them, for example by up-sampling, is quite involved and does not map to graphics hardware.

A point-based geometry contains a certain amount of noise. For such geometry, a rendering method proposed by Kalaiah and Varshney (2001) may be more suitable than a splat-based approach. They build a hierarchy over a given set of point using Principal Component Analysis (PCA) of the geometry attributes, followed by k-means clustering. Each node in the hierarchy represents a set of points. To render the data set, points in viewable nodes are generated using quasi-random sampling. The viewable nodes are then rendered by splatting the generated points.

2.2. Ray Tracing

The idea of using ray shooting for computing images is as old as 1968, beginning with the work of Arthur Appel (Appel 1968). Appel introduced it as an alternative method for solving the hidden surface problem in rendering solid objects. Ray tracing is now an important technique for the creation of photorealistic images given a synthetic scene or model description. In ray tracing, as illustrated in Figure 2.3, one follows the path of a ray as it travels from the eye point through a pixel in the viewing window to the objects in the scene and on to the light sources illuminating the scene.

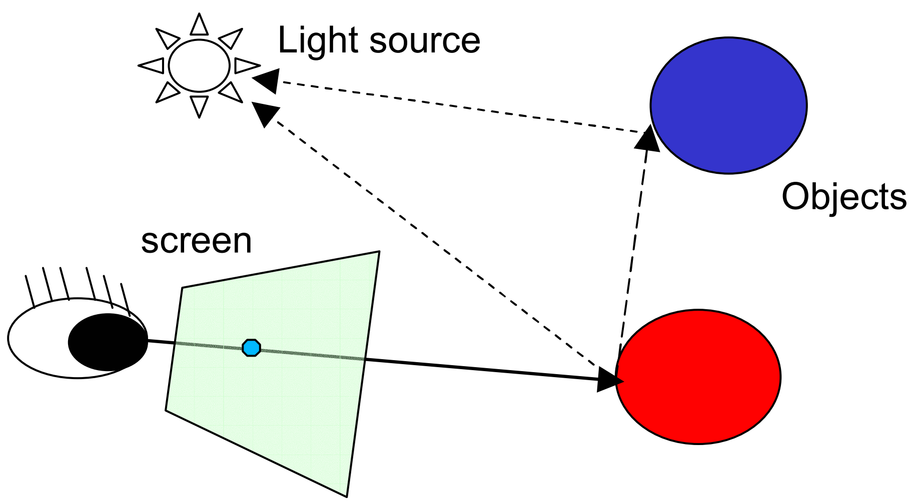


Figure 2.3: The Ray Tracing Concept

Algorithmically, ray tracing takes on the form as shown in the following pseudocode:

```
For each pixel in image,
    Form a ray that passes through the pixel
    Intersect the ray against each object in the scene
    If there is an intersection
        Color the pixel according to intersection surface property
    Else
        Color the pixel with a background color
```

Pseudocode 2.1: The Ray Tracing Algorithm

The key part in the ray tracing algorithm is where one intersects a ray against the objects in a scene. There are two basic problems in doing such intersection. First of all, one needs to know how to compute the intersection between a ray and an object. The way in which this is to be done depends on the geometric primitives involved. It will be explained in the next subsection on how this intersection can be done for a point-based geometry. Secondly, one needs to search for the closest point or cluster of points that is intersected by the ray. One could do so in an exhaustive brute-force manner by simply iterating through the list of points in an input point-based scene and finding which is the closest to the ray origin along the ray direction. However, given that there could be millions of point primitives in a scene, it is important for the search to be done in an efficient manner.

2.2.1. Ray-Surface Intersection

This section considers ray-surface intersection in the context of point-based geometry. A point-based geometry comprises of a set of point samples. There is a fundamental issue in applying ray tracing to a point set. A point, mathematically, has neither volume nor area. A ray is thin. Hence, the chance of a thin ray hitting a mathematical point is practically nil. To deal with this problem in a practical manner, previous works point to a few options:

1) consider each point as defining a small surface area (Schaufler and Jensen 2000)
2) consider using a thick ray, instead of a thin one (Wand et al 2003)
3) use the original points to define a continuous surface (Adamson and Alexa 2003, Wald and Seidel 2005).

Option 1

The first option is the simplest and fastest among existing approaches, and hence is the first approach that the author implemented in his ray tracer. This shall be referred to as the Schaufler-Jensen (*SJ*) approach. Corresponding naturally to the way in which a 3D digitizer scans a physical surface, one associates a disk with each point P_i, $0 \le i \le n$, where $n+1$ is the number of points in the input geometry. The disk has a certain normal, N_i, and radius, r_i, associated with it. The disk is centered at the point P_i at location p_i. The equation of a plane covering the disk is given by:

$$xN_x + yN_y + zN_z + D = 0 \quad \text{……………….. Equation 2.3}$$

where x, y, and z are the free variables of the equation, N_x, N_y and N_z are the components of the normal N_i, and D is a real number which is unique for a given plane.

The parametric form of a ray r is given by:

$$r(t) = o + td, 0 \leq t$$ ……………….. Equation 2.4

where o is the ray's origin, d the direction vector, and t the parameter.

The standard ray-plane intersection calculation computes t as follows:

$$t = \frac{-(N_x o_x + N_y o_y + N_z o_z + D)}{N_x d_x + N_y d_y + N_z d_z}$$ ……………….. Equation 2.5

Substituting t into the parametric form of the ray, one obtains a positional value, I. One then check the distance, s, between I and p_i. If $s < r_i$, the ray does intersect with the disk representing the point P. Otherwise, there is no intersection. Figure 2.4 illustrates an example intersection.

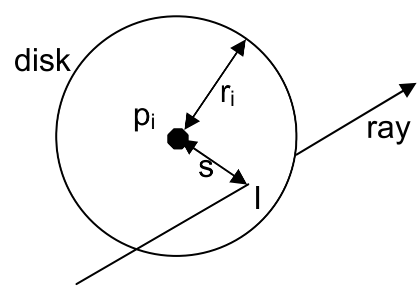


Figure 2.4: Ray-Disk Intersection

Considering points as disk results in a simple and fast ray-surface intersection computation. However, if one intersects with only a single disk, in the rendered image, disks may be seen sticking out, especially at curved area. The cause of this problem is that the area representing each point actually overlaps, as shown in Figure 2.5.

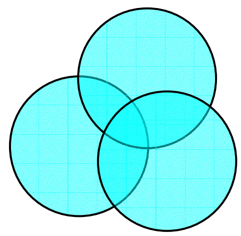


Figure 2.5: Points as Overlapping Disks

To alleviate the problem, one intersects the ray with each of the point primitives. From each intersection, the corresponding attributes (eg. position, normal), *attrib*$_i$, are then interpolated according to the following weighing scheme used by Schaufler and Jensen (2000):

$$attrib = \frac{\sum_i attrib_i * \| I - p_i \|}{\sum_i \| I - p_i \|}$$ ……………….. Equation 2.6

The computed intersection point is slightly dependent on the direction of the incoming ray. But this, in the author's experience and as also reported by Schaufler and Jensen (2000), is not perceptually visible. However in casting shadow rays, this must be taken into account, and a shadow offset must be used. Similar adjustments are also required for reflected and transmitted ray.

Option 2

The second option, reported by Wand et al (2003), is more expensive than the first, as its focus is on anti-aliased ray tracing of polygonal scene rather than fast ray tracing of point-based geometry. Points are used to accelerate the ray tracing process. The approach will only be briefly described here.

To obtain the intersection between a point sample *p* and a cone ray *r*, as described by Wand et al (2003), one first expresses the point in ray coordinates. To do this, the vector, *d*, between *p* and the ray origin is computed. Note that the vector *d* here does not describe the actual ray direction. Instead, it describes the direction from the ray origin to the point *p*. The scalar product between *d* and the ray orthogonal coordinates n_r, u_r and v_r (see Figure 2.6) expresses the point coordinates in ray coordinates.

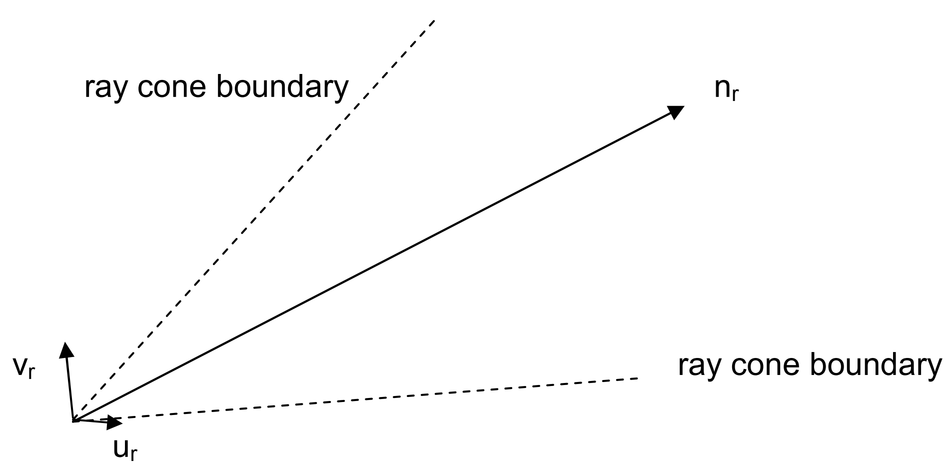


Figure 2.6: Ray Coordinate System