Efficient Ray Tracing of Many Light Sources

Xiaomei Wang

A thesis submitted to the Faculty of Graduate Studies
in partial fulfillment of the requirements
for the degree of

Master of Science

Graduate Programme in Computer Science
York University
Toronto, Ontario

June 1999

Efficient Ray Tracing for Many Light Sources

by

Xiaomei Wang

a thesis submitted to the Faculty of Graduate Studies of York University in partial fulfillment of the requirements for the degree of

Master of Science

1999

Efficient Ray Tracing of Many Light Sources

Xiaomei Wang

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Program of Computer Science
York University

ABSTRACT

Standard ray tracing algorithm slow down unacceptably when a large number of light sources are in the scene because the shadow determination process is $O(n)$, where n is the number of light sources. This thesis puts forward a new *Light Extent Volumes* approach to efficiently ray tracing scenes with many light sources. By building a hierarchical tree of light extent volumes one can approach approximately logarithmic complexity for typical scenes in determining which light sources contribute significant irradiance to the intersection point. This allows tens of thousands of light sources in a scene to be rendered in reasonable time. The relative performance of the algorithm improves as the number of light sources increases in the scene. It achieves significant speedup over other existing approaches, up to 150 times faster. Moreover, the algorithm requires minimal memory overhead for shadow testing acceleration. Another important feature is its simplicity of implementation. In addition, the approach is orthogonal to most other global illumination techniques and can be added to existing direct light calculation and optimizations. The *Light Extent Volumes* approach is a practical algorithm for efficiently ray tracing scenes with many light sources.

Acknowledgements

Many thanks to my supervisor, Professor John Amanatides, for his help, patience, knowledge advice, and his valuable time which lead to the completion of this thesis.

Thanks also to the committee members for their insightful suggestions and advice regarding to this thesis.

I would also extend my thanks to my friends Claude, Oleg, Kenneth, Johnathon, Zusheng, Chang, Biao, Linqi, Xiaoyan, Ying, Chris, lila, Laura, and Razvan for their fun, help, and support during these two years.

Special thanks to my husband Shen Zhou for his love, understanding, support and encouragement. Further thanks to my parents, sisters and two angelic nephews, Kangning and Kaiwen. Without their encouragement and love, I would not be able to achieve this goal.

Table of Contents

Chapter 1

Introduction

1.1 Introduction to Ray Tracing

Ray tracing is a powerful 3D image-rendering technique that simulates the interaction of light with 3D objects at each intersection point within the environment. It was first developed by Appel [APPE68] for visible surface determination. Whitted [WHIT80] then extended ray tracing to handle specular reflection and refraction.

For a typical ray tracer, given a viewpoint and a view plane which is divided into a grid and each element in the grid represents a pixel of the resulting image, a ray is shot from the viwepoint, through a pixel, and into the scene, as shown in Figure 1.1. The first object that the ray intersects is the object visible in that pixel of the view plane. When a ray intersects with an object, the intensity and color of the intersection point are assigned to the pixel. Refraction and reflection are modeled by recursively shooting refraction and reflection ray until a bounce limit is exceeded or no more objects are encountered, as shown in Figure 1.2. This scheme produces high quality images.

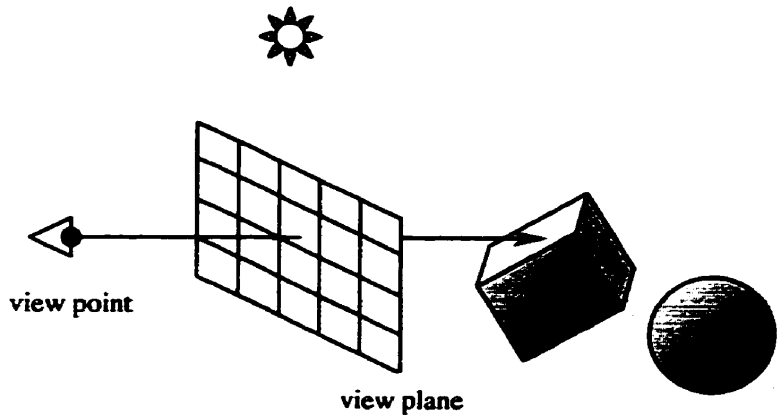


Figure 1.1 example of simple ray tracing

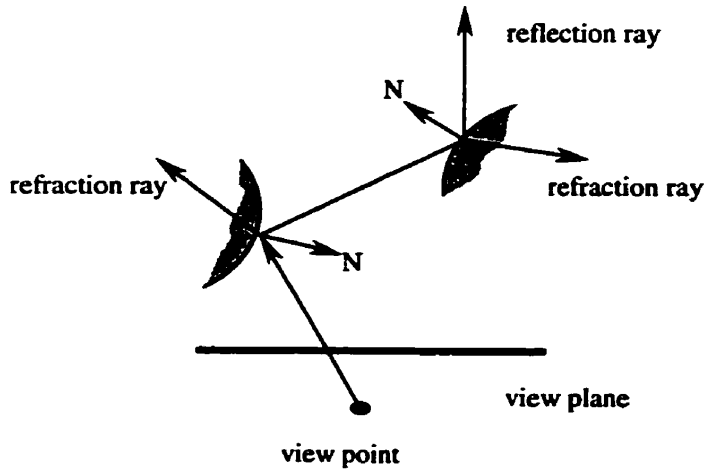


Figure 1.2 ray tracing modeled by reflection and refraction

1.2 Problem of Ray Tracing Scenes with Many Light Sources

For most existing commercial rendering systems (for animations, film special effects, postproduction, advertising, etc.), ray tracing remains the rendering algorithm of choice. In such environments, scenes containing a large number of geometric primitives as well as a large number of light sources are common. Unfortunately, standard ray tracing algorithms slow down unacceptably when large numbers of light sources are in the scene because the shadow determination process is $O(n)$, where n is the number of light sources. We need to send a shadow ray from each intersection point towards every light source to determine whether the intersection point is visible to that light source. If there are more than a few light sources, shadow determination quickly becomes the dominant computation even though there is a high probability that most light sources have negligible influence to most of the scene. What is needed is an approach whose complexity is better so that we can render scenes with tens of thousands of light sources in reasonable time.

Researchers have been looking at ways of solving this problem and there are two distinct strategies to consider: (1) *fewer ray-object intersection approaches* which reduce the number of objects that have to be intersected with a ray by only concentrating on objects close to the shadow ray, including Bounding Volume Hierarchies [WHIT80, RUBI80, WEGH84, KAY86, STUE94] and Space Subdivision [GLAS84, KAPL85, JANS86, FUJI86, AMAN87, EOKS89, WOO90a]; (2) *shadow ray reduction methods* which shoot shadow rays only to the most significant light

sources and then estimate the visibility of the others [WARD91, HOUL93, SILL94, SMIT94, STAM95, SHIR96, PAQU98].

1.3 Goal

The objective of this thesis is to develop a new approach to efficiently ray tracing scenes with many light sources. It focuses on the target that given an intersection point during the ray tracing process, quickly determine all the light sources which make great contribution to this intersection point. By building a hierarchical tree of light extent volumes one can approach approximately $O(\log n)$ complexity in determining which light sources contribute significant irradiance to the intersection point, where n is the number of light sources. This allows hundreds or even tens of thousands of light sources in a scene to be rendered in reasonable time.

Chapter 2

Background

2.1 Overview

In this chapter, ray tracing topics related to this thesis are discussed. Topics include illumination model, recursive ray tracing, intersection culling techniques, and the techniques of ray tracing multiple lights.

2.2 Illumination Model

The role of the illumination model is to determine how much light is reflected to the viewer from a visible point on a surface as a function of light source direction and strength, viewer position, surface orientation, and surface properties. In this section, Phong's [PHON75] illumination model is introduced because it is the most popular illumination model in the computer graphics field. Phong's model includes three factors: diffuse reflection, specular reflection, and ambient light. Based on Phong's model, an improved point-light-source model is further provided to simulate some of the directionality of the lights, such as sharply delineated spotlights.

2.2.1 Lambertian Diffuse Reflection

Lambertian diffuse reflection is the simplest type of reflection where a ray of light, after an amount of absorption, is scattered back into the environment with equal intensity in equal directions (Figure 2.1 a). The amount of energy reflected per unit area is proportional to the cosine of the angle between the normal to the surface at that point, and the direction to the light source (Figure 2.1 b).

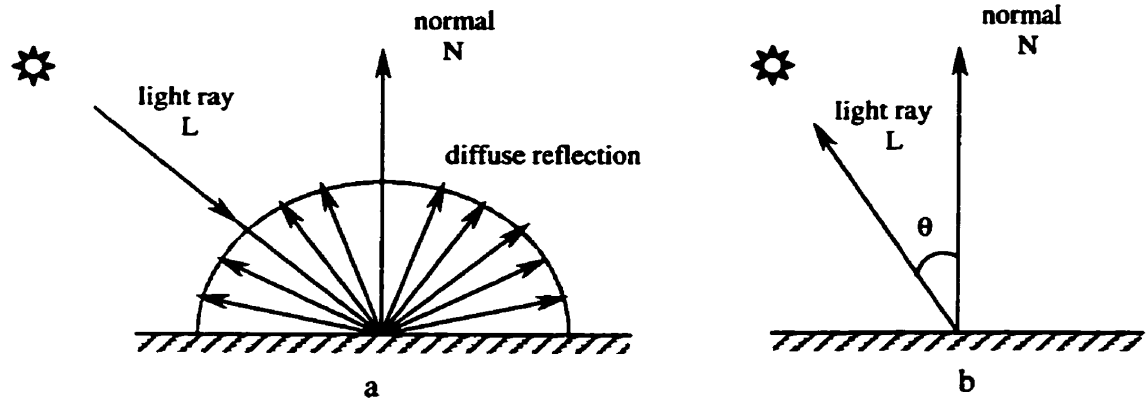


Figure 2.1 Diffuse reflection – light is scattered in all directions from a point on the surface

At this stage in the development of the model, we should consider the role of distance by adding an attenuation term that reduces light intensity as a function of the distance of the surface from the light source. This ensures that surfaces of the same color, but at different distances from the light sources, are not assigned the same intensity. The physical choice for this attenuation term is $1/d^2$.

Lambertian diffuse reflection can be written as:

$$I_d = (I_i / d^2) K_d \cos\theta \quad (1)$$

Where

I_i is the intensity of the light source;

d is the distance between the intersection point and the light source;

K_d is a constant of reflection dependent on the surface material;

θ is the angle of reflection between the incident light direction and the surface normal.

Lambertian diffuse surfaces appear dull and do not produce regular reflection. It is important to note that the intensity the viewer sees reflected off of a diffuse surface is independent of the viewer's position.

2.2.2 Specular Reflection

Specular reflection is due to the shininess of a surface. Unlike diffuse reflection, it is highly dependent upon the position of the viewer and the light source at each point on the specular surface. In Phong's model, specular reflection is scattered about the mirror direction when a surface is not mirror like but still shiny (Figure 2.2 a).

In this model the intensity of the reflection is proportional to the cosine (raised to some power) of the angle between the mirror direction and viewer direction (Figure 2.2 b).

$$I_s = (I_i / d^2) K_s (\cos\alpha)^n \quad (2)$$

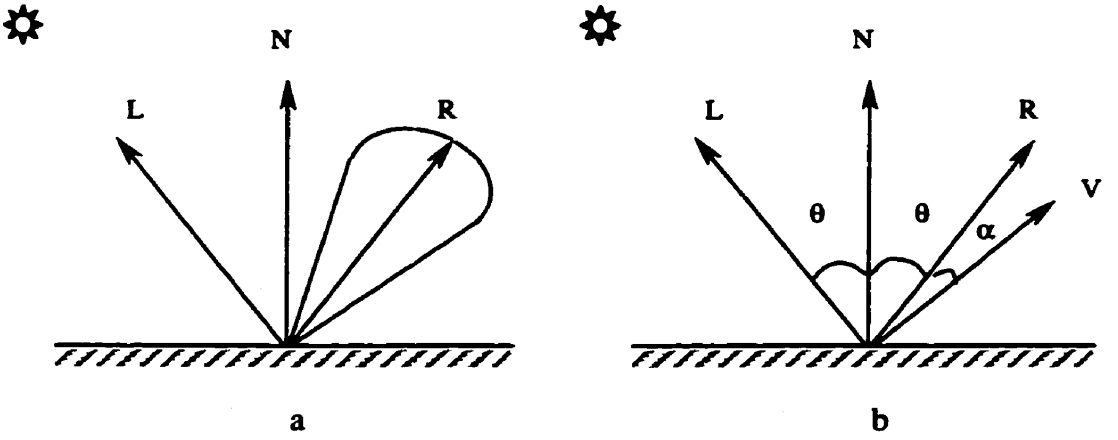


Figure 2.2 Specular reflection – light is concentrated around the mirror direction R

Where

I_i is the intensity of the light source;

K_s is the constant representing the fraction of energy reflected specularily, which is a function of the surface properties;

α is the angle between the mirror direction and the viewer direction;

n is the specular exponent, also a function of the surface properties.

Values of n typically vary from 0 to several hundred, depending on the surface material being simulated. Shinier surfaces have larger values of n. A value of $n = 0$ will result in a Lambertian diffuse reflectance, and a value of $n = infinity$ will result in a mirrored surface.

2.2.3 Ambient Light

Ambient light is the result of multiple reflections from many surfaces in the environment, and is incident on a surface from all directions. Without ambient light, objects in shadow would be completely black. Since it's generally very expensive to directly compute the ambient light, the ambient component is often modeled as a constant term for a particular object by using a constant ambient reflection coefficient as shown below:

$$I = I_a K_a \tag{3}$$

Where

I_a is the intensity of the ambient light;

K_a is the *ambient-reflection coefficient* which determines the amount of ambient light reflected from an object's surface.

2.2.4 Phong Model

Combining ambient light, diffuse reflection and specular reflection, intensity from Phong's model is given by

$$I = I_a K_a + (I_i / d^2) [K_d \cos\theta + K_s (\cos\alpha)^n] \tag{4}$$

In Phong's model, the global term (ambient) is modeled as a constant, and the diffuse and specular terms are modeled as local components. The overall effect of the lack of interaction between objects in a scene is that they appear plastic like. Also, the lack of shadows means not only that objects do not cast a shadow on other objects, but self-shadowing within an object is omitted. These can be solved by the technique of *Recursive Ray Tracing*.

2.2.5 Improved Point-Light-Source Model

Real light sources do not radiate equally in all directions. Warn [WARN83] has developed easily implemented lighting controls to model some of the directionality of the lights. A directed light is modeled mathematically as the light emitted by a single point specular reflecting surface illuminated by a hypothetical point light source, as shown in Figure 2.3. Think of the point labelled "LIGHT" in Figure 2.3 as a surface which reflects light onto the object. The normal orientation of this single point surface is controlled by the light direction vector. A hypothetical point light located along this vector illuminates the reflector surface which, in turn, reflects light onto the object.

We can use the Phong illumination equation to compute the intensity of the reflected light at a point on the object. If we further assume that the reflector has a diffuse coefficient of 0 and a specular coefficient of 1, then the light's intensity at a

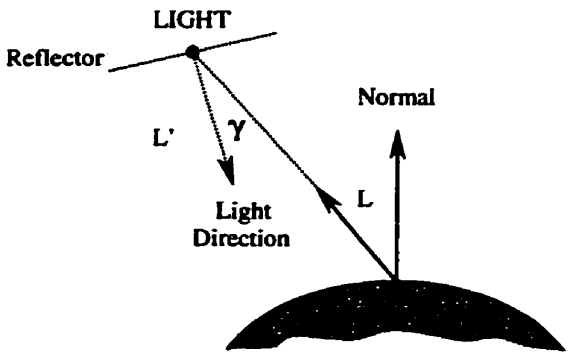


Figure 2.3 Warn's lighting model. A light is modeled as the specular reflection from a single point illuminated by a point light source.

point on the object is

$$I_{L'} \cos^{P}\gamma \tag{5}$$

Where $I_{L'}$ is the intensity of the hypothetical point light source, γ is the angle between $-L$ and the hypothetical surface's normal, L', and P is the reflector's specular exponent. The exponent P provides control over the concentration of the light. By increasing the value of P, the light becomes more concentrated around the primary direction. This can be used to simulate the effect of a spotlight. The light can be aimed by adjusting the orientation of the light direction vector.

Equation (5) can thus be substitued for the light-source intensity I_i in the formulation of Equation (4). Then the intensity of a directed light at a point on the object based on Phong's model is given by

$$I = I_a K_a + (I_{L'} \cos^{P}\gamma / d^2)\,[\,K_d \cos\theta + K_s (\cos\alpha)^n\,] \tag{6}$$

In Warn's method, a sharply delineated spotlight is modeled as a variable sized cone surrounding the light direction. As shown in Figure 2.4, a cone with a generating angle of δ may be used to restrict the light source's effect by evaluating the illumination model only when $\gamma < \delta$.

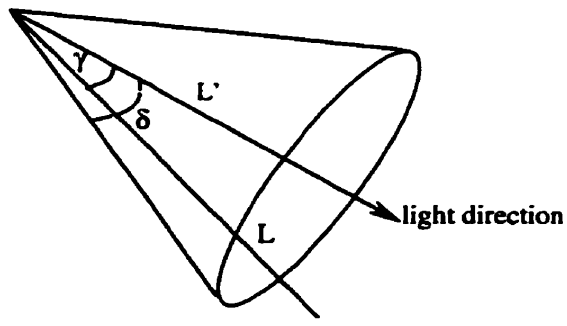


Figure 2.4 The intensity distribution of a spotlight is restricted with cone.

2.3 Recursive Ray Tracing

The illumination model described in the Section 2.2 is for simple ray tracing which only computes pixel values at the closest intersection of a ray from view point with objects. Recursive ray tracing extends to handle shadows, reflection, and refraction (Figure 2.5).

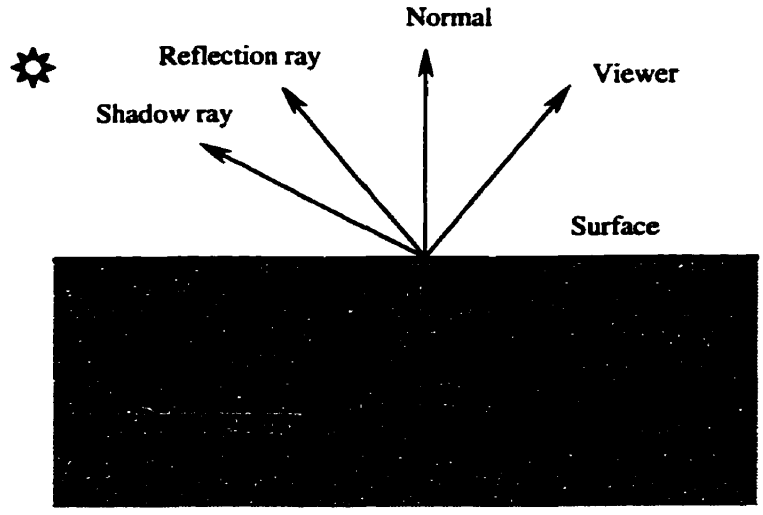


Figure 2.5 reflection, refraction and shadow are spawned from a point of intersection

To calculate shadows, an additional ray from the point of intersection is shot to each of the light sources. If the shadow ray strikes an object before reaching the light source, the point of intersection is in shadow.

The illumination model developed by Whitted [WHIT80] and Kay [KAY79] fundamentally extends ray tracing to include specular reflection and refractive transparency. Each of these reflection and refraction rays then in turn recursively

spawn shadow, reflection and refraction rays. The illumination model by Equation (6) can be extended to:

$$I = I_a K_a + \sum_{i=1}^{m} (S_i I_L \cdot \cos^r \gamma / d_i^2)[K_d \cos\theta + K_s (\cos\alpha)^n] + K_r I_r + K_t I_t \qquad (7)$$

Where

m is the number of light sources

S_i is the visibility factor of the light source

K_r is the reflection coefficient

I_r is the intensity of the reflected ray

K_t is the transmission coefficient

I_t is the intensity of the refracted transmitted ray

Values for I_r and I_t are determined by recursively evaluating Equation (7) at the closest surface that the reflected and transmitted rays intersect. A maximum depth can be used to limit the times of recursion in a very shiny environment, or the recursion is stopped when the ray doesn't intersect with any object in the scene.

2.4 Intersection Culling Techniques

In ray tracing, most of the computational time goes to computing intersections between rays and objects. So right from the start, ray tracers [WHIT80] included schemes for reducing linear time complexity so that they could handle complex scenes in reasonable time. These schemes try to quickly determine candidate objects which have a high probability of intersecting the ray, and generally come into two flavors: *Hierarchical Bounding Volumes* [WHIT80, RUBI80, WEGH84, KAY86] and

Spacial Subdivision [GLAS84, KAPL85, JANS86, FUJI86, AMAN87, EOKS89, WOOA90].

2.4.1 Hierarchical Bounding Volumes

The most fundamental and ubiquitous tool for ray tracing acceleration is the *bounding volume*. Bounding volumes provide a particularly attractive way to decrease the amount of time spent on intersection calculations. Each volume contains a given object and permits a simpler ray intersection check than the object. Only if a ray intersects the bounding volume does the object itself need to be checked for intersection. If the ray misses most objects, intersection of the bounding volume reduces computation times significantly. Whitted [WHIT80] initially used spheres as bounding volumes since they are the simplest shapes to test for intersection.

Though bounding volumes substitute simple intersection checks for more costly ones, they don't reduce the number of checks. Theoretically it may reduce the computation by a constant factor, but cannot improve upon the linear time complexity. To alleviate this problem, Rubin and Whitted [RUBI80] firstly used *hierarchical bounding volumes* in ray tracing to attain a theoretical time complexity which is logarithmic (expected case) in the number of objects instead of linear.

When constructing the hierarchy structure tree, each object is bounded in a volume (e.g. cube, sphere) whose geometric attribute is much simpler than the object itself. Furthermore, close objects are grouped together to form another bounding volumes. By enclosing a number of bounding volumes within a larger bounding volume it was possible to eliminate many objects from further consideration with a

single intersection check. A child volume is guaranteed not to do the intersection test if its parent does not. Thus, if intersection tests begin with the root, many branches of the hierarchy may be trivially rejected. For example, in Figure 2.6 (a), the ray first hits volume A and then one of its children, volume B. Since the ray does not hit volume C, the further intersection test for volume C can be avoided immediately. Figure 2.6 (b) shows the corresponding bounding volume hierarchy tree.

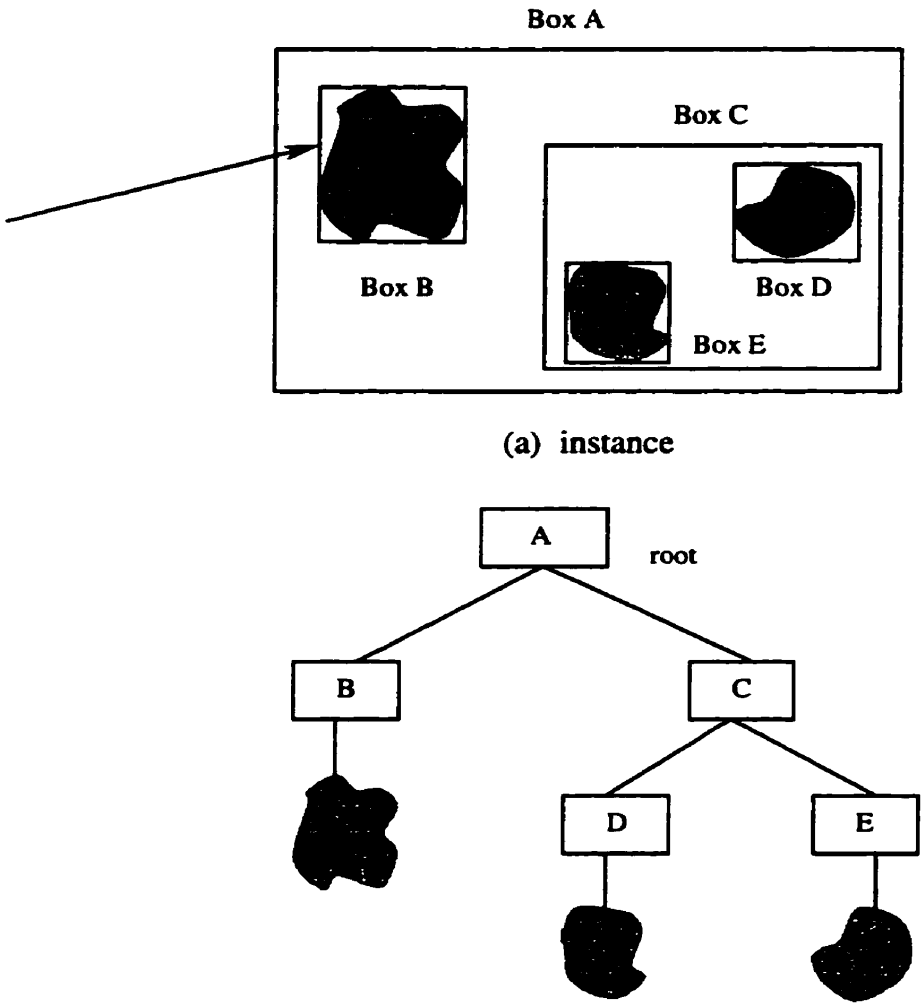


Figure 2.6 example of hierarchical bounding volumes

2.4.2 Spatial Subdivision

Spatial subdivision scheme works by partitioning a volume bounding the environment into voxels. A fundamental difference between bounding volume hierarchies and spatial subdivisions is that the former selects volumes based on given sets of objects, whereas the latter selects sets of objects based on given volumes.

The basic spatial subdivision technique [GLAS84] is built on the basis of the voxel traversal grid structure (Figure 2.7). Space encompassing all objects is placed in a grid of cubes called voxels. Each voxel contains a list of all objects which reside in that voxel. Each ray traverses the grid in order and tests for intersection only with objects residing in the voxel traversed, until an intersection is found or the ray has completely traversed the grid. Performance is improved because less objects are handled in each ray intersection computation.

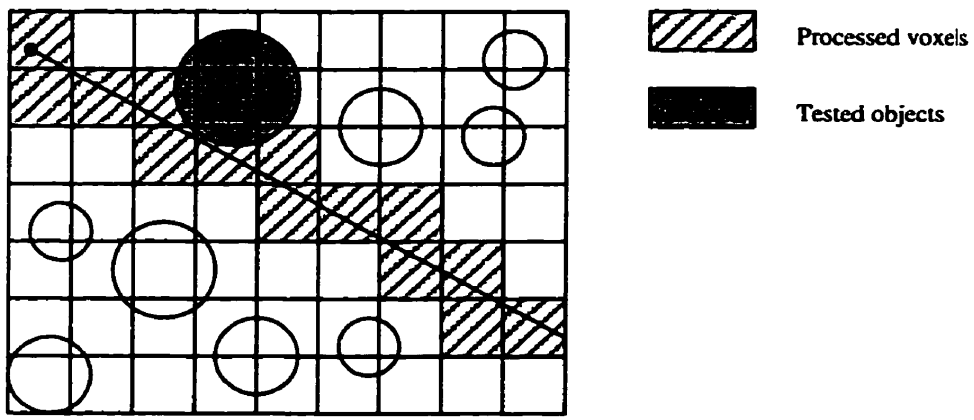


Figure 2.7 A 2D analogy of uniform spacial subdivision

2.5 Ray Tracing Multiple Lights

When ray tracing complex models such as large building, the resulting scene typically contains hundreds or even thousands of light sources. In rendering such scenes, shadow determination quickly becomes the dominant computation even though there is a high possibility that most light sources have negligible influence to most of the scene. There is an extensive literature on the research dedicated to speeding up shadow calculations using spatial coherence and subdivision [WOO90b]. Most of these approaches however are highly dependent on the number of light sources, and are thus unsuitable for scenes with many light sources [BERG86, WARD91, HOUL93, SILL94, SMIT94, STAM95, SHIR96, PAQU98].

In this section, three recently emerged approaches of rendering scenes with thousands or even more light sources are discussed, including *Adaptive Shadow Testing* [WARD91], *Monte Carlo Direct Lighting* [SHIR96], and *Light Hierarchy* [PAQU98]. These techniques based on the assumption that in such scenes at most a few hundred light sources (and usually at most tens) will contribute significantly to the radiance at any particular point. So they employed different ways to identify important light sources from other negligible lights, shoot shadow rays only at the most important light sources and then estimate the visibility of the others.

2.5.1 Adaptive Shadow Testing

Ward presented *Adaptive Shadow Testing* method [WARD91] which performs well for a moderate number of light sources, and is the most suitable algorithm to date for the treatment of scenes with many light sources.

This approach first calculates the potential contribution of each light source at every point to shade (without considering visibility), and uses this estimation to generate a sorted list of light sources. The ordered list is then traversed and thus shadow testing on the sources with highest potential contributions is computed first. If the sum of the potential contributions of the remaining light sources is below some threshold, the traversal stops. The algorithm can be written as the following steps:

1) Compute potential contributions from all light sources in front of the point

2) Sort the contributions in descending order

3) Compute ***r(i)***, the sum of the next N^C contributions smaller than source ***i***, where ***N*** is the number of light sources and ***C*** is the certainty

4) Initialize the sum (***S***), hits(***V***), and tests(***W***) to 0.
   For *each contribution in the sorted list* do
      If ***S t > r(i)*** then go to step (5);
      Increment the test counter, ***W***;
      Increment the test counter for light source ***i***, ***W(i)***;
      If *source **i** is visible from this point* then
         Increment the hit counter, ***V***;
         Increment the hit counter for source ***i***, ***V(i)***;
         Add contribution for source ***i*** to ***S***;
      End if
   End for
5) For *each untested contribution* do
      Multiply contribution by ***V/W*** and ***V(i)/W(i)***;
      Add weighted contribution to ***S***;
   End for

6) Return ***S***

Figure 2.8 Pseudo code of Adaptive Shadow Testing

2.5.2 Monte Carlo Direct Lighting

Shirley *et al.*[SHIR96] introduced Monte Carlo techniques [HAMM64, SHRE66, HALT70, YAKO77] for direct lighting calculations by using a probability intensity function over all the light sources. Since the mixture probability intensity function is the sum of the products of mixing weights and individual probability functions, in the linear method, the calculation of mixture probability density function requires querying every light in the scene. This might be too slow with thousands or millions lights. Based on the assumption that in such scenes at most a few hundred lights will contribute significantly to the radiance at any particular point, the lights are then divided into two subsets: one is the set of bright (important) lights, and the other is the set of dim (less important) lights. This selection is performed as a preprocess, and is based on an approach similar to the sphere of influence. A sampling probability is then assigned to each bright light source, and a unique probability is assigned to all the dim light sources. If a large number of rays are shot per pixel, this method can be very effective.

The difficult part of this method is deciding which lights are important for a particular point. As pointed out by Kok and Jansen [KOK91], a light that is responsible for a large fraction of the radiance of a point is likely to be responsible for a large fraction of the radiance of its neighboring points. A spatial subdivision scheme is then used to precompute the list of important lights for each cell in the spatial subdivision structure. For a particular cell, a light is put in the candidate list if it might contribute more than a threshold average spectral radiance to a diffuse

surface within the cell. To characterize important versus unimportant lights, an axis-aligned influence box is associated with each light that includes all points that might include that light in its important light list. When deciding whether a light is important to a cell, just check whether the cell and the influence box overlap, and if so, then the light is treated as an important light source.

2.5.3 Light Hierarchy

Paquette [PAQU98] introduced a new data structure in the form of a light hierarchy for efficiently ray-tracing scenes with many light sources. An octree structure is constructed with the point light sources in a scene. Each node keeps an approximate representation of the light sources it contains by means of a *virtual light source*. Figure 2.9 shows an example of such a point light hierarchy.

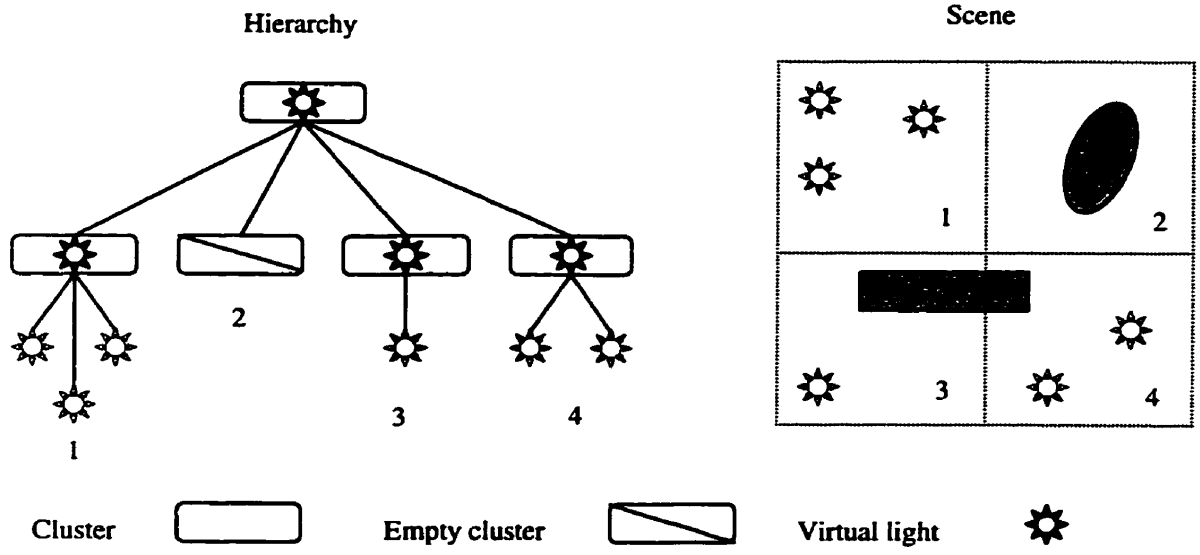


Figure 2.9 example of light hierarchy

Once the light hierarchy is built, error bounds committed with the virtual light approximations are developed to shade a point, both for the cases of diffuse and specular reflections. These bounds are then used to guide a hierarchical shading algorithm. If the current level of the light hierarchy provides shading of sufficient quality when the associated error bound is below the desired threshold, the approximation is used, thus avoiding the cost of shading for all the light sources contained below this level. Otherwise the descent into the light hierarchy continues.

2.5.4 Comparison of Ray Tracing Multiple Lights Approaches

Three approaches of rendering the scene with thousands or even more of light sources have been described in this chapter. Ward's Adaptive Shadow Testing approach performs well for a moderate number of light source, but since its complexity is $O(n)$, as the number of light sources increases, the cost of sorting the contributions of all these light sources can make this method impractical for the rendering. Compared to Adaptive Shadow Testing, Paquette's Light Hierarchy approach is an important improvement for scenes with a high number of light sources. However, this method is designed only for scenes without occlusion which are uncommon in the daily life. Monte Carlo Direct Lighting approach can be very effective if a large number of rays are shot per pixel. Unfortunately, as with all Monte Carlo approaches, noise due to insufficient sampling can appear in the rendered images. Moreover, since the unimportant light source to be sampled is chosen randomly, an unsuitable partitioning into unimportant and important light sources can greatly increase the amount of noise.

2.6 Solid Modeling

There are many ways to model 3D objects. In this section, we provide a summary of two of them, *Constructive Solid Geometry* and *Instancing*.

2.6.1 Constructive Solid Geometry

Constructive Solid Geometry (CSG) is a method of creating complicated objects by performing Boolean set operations on more primitive objects, such as sphere, cube, and cylinder and so on. Typically an object is stored as a CSG tree with simple primitives at the leaves and operators at the internal nodes. Some interior nodes represent Boolean set operators, whereas others perform affine transformation, such as translation, rotation and scaling. A complicated object can be defined by a CSG tree, as shown in Figure 2.10.

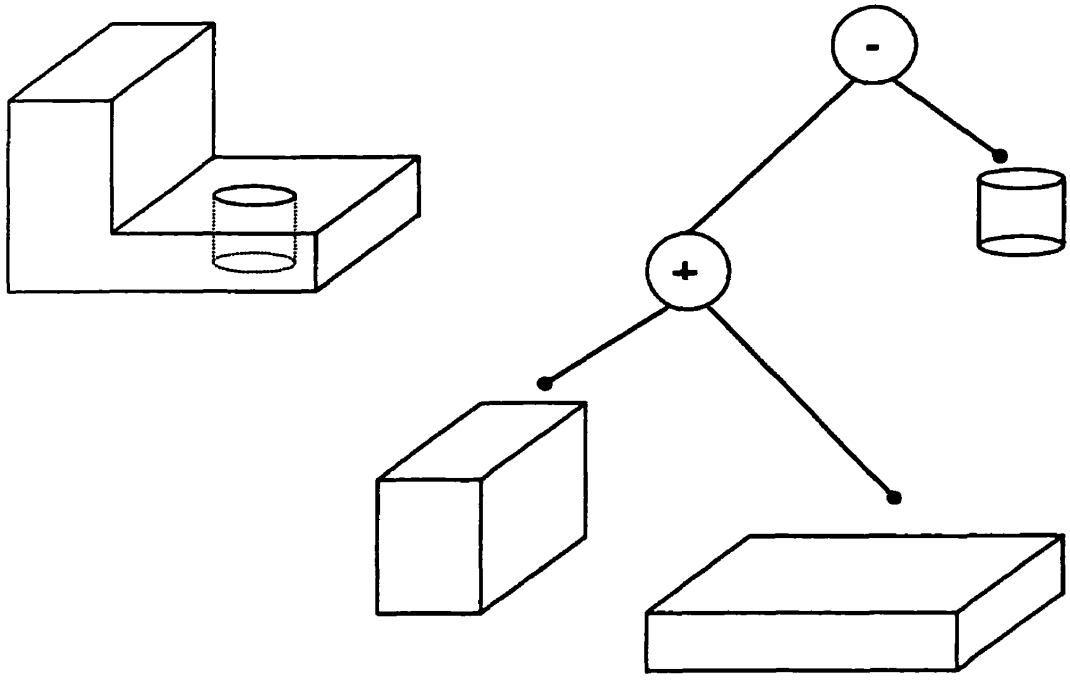


Figure 2.10 an object defined by a CSG tree

2.6.2 Instancing

In some complicated models, the same objects may occur repeatedly only with a few different characteristics, such as transformation attributes. When building up a hierarchy of the model, we may only create one master copy of the object. Copies of other objects can be represented by simply using some instance nodes referring to the master object with their own transformation. Thus, once the master object is instanced in the model, the same object is automatically replicated many times. So instancing saves a lot in both space and time for generating the primitive hierarchy. Figure 2.11 shows a model hierarchy with instancing.

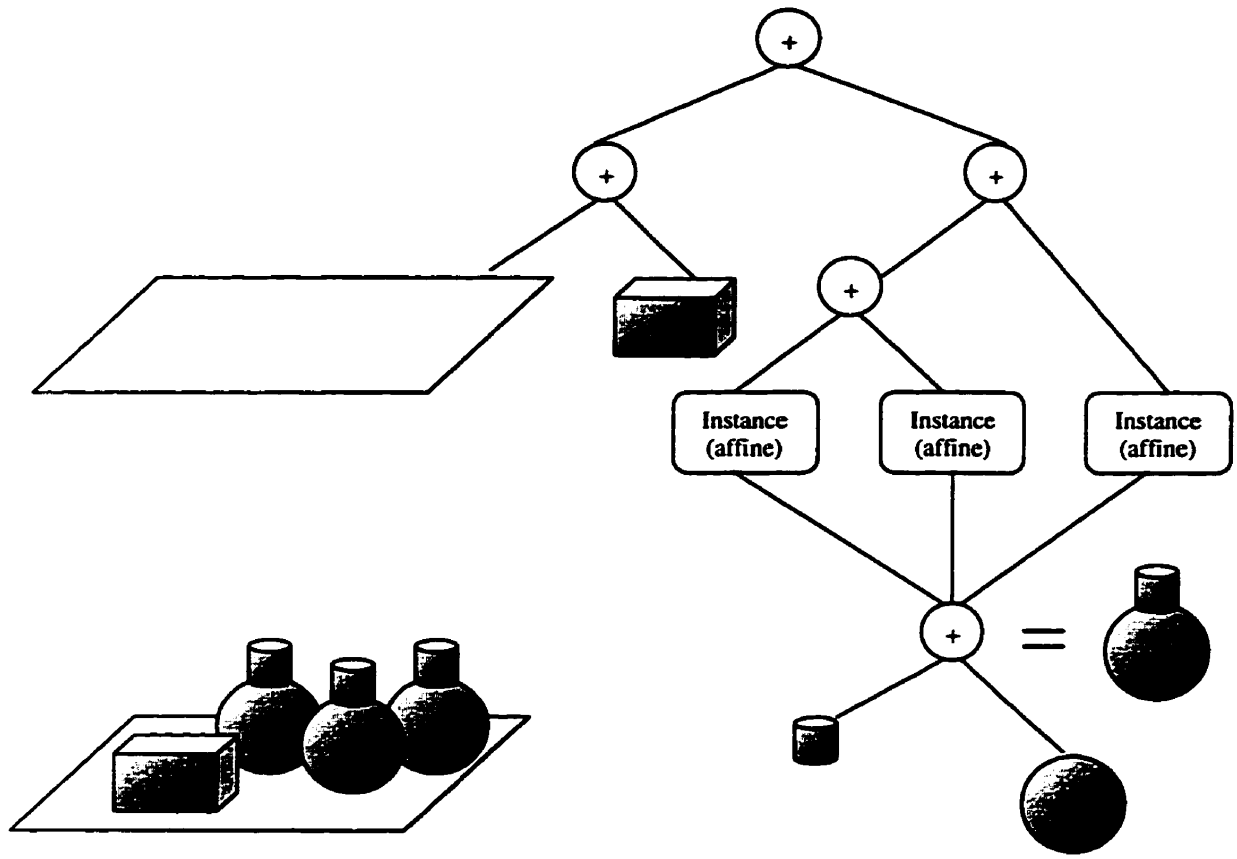


Figure 2.11 a model hierarchy with instancing

Chapter 3

Efficient Ray Tracing with Many Light Sources

3.1 Overview

Standard ray tracing algorithms slow down unacceptably when large numbers of light sources are in the scene because the shadow determination process is $O(n)$, where n is the number of light sources. By building a hierarchical tree of light extent volumes one can approach $O(\log n)$ expected-case complexity in determining which light sources contribute significant irradiance to the intersection point. This allows tens of thousands of light sources in a scene to be rendered in reasonable time.

3.2 Motivation

In scenes with multiple light sources, typically only a few will create strong shadows in any part of the scene. These will generally be the sources with the high concentration of light in that section due to source brightness, direction and proximity.

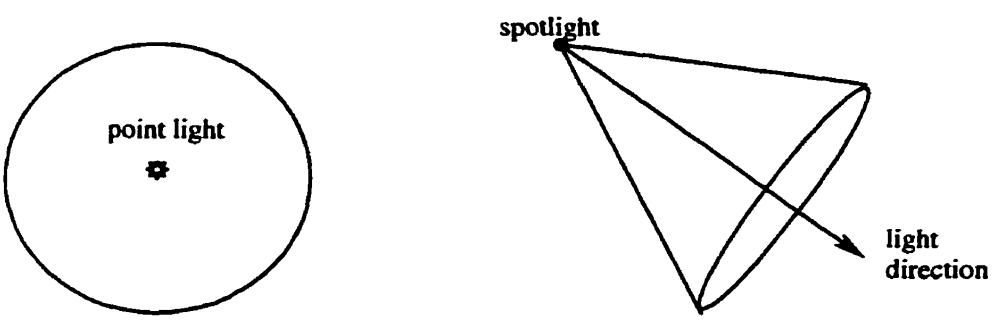
This observation leads to a simple optimization: we can perform shadow testing only on the light sources with high contributions, and quit testing for those unimportant lights whose contributions are below some threshold.

So all that we need is a way to quickly decide which light sources are important enough for a particular point on the object surface to shoot shadow rays to.

3.3 Light Extent Volumes

Given a light source, the *light extent volume* is the volume in space where the irradiance from the light source is above a given level *tolerance*. Here the tolerance is a specified light intensity threshold that is the minimum influence considered to the scene for rendering results.

As shown in Figure 3.1, for a point light source that radiates uniformly in all directions, its light extent volume would be a sphere whose radius is determined by tolerance. For a spotlight, it would be a cone. A light only has effect on objects that lie within its light extent volume. If the object is outside the volume, the intensity contribution is assumed to be zero, and no further evaluation of the illumination model is required for that light. Of course, this only makes sense when relatively few



(a)
light extent volume of a point light source

(b)
light extent volume of a spotlight source

Figure 3.1 light extent volume models

lights illuminate any individual surface. If there are a large number of rays from light sources in the vicinity of some surfaces, then the combined irradiance may be significant even though individually they all may be below the tolerance. Fortunately, this situation is very rare and can easily be checked for.

3.4 Hierarchy of Light Extent Volumes

As mentioned in the overview, a hierarchical data structure of light extent volumes representing the light sources in the scene is required to achieve the goal of efficient treatment of scenes with many light sources.

The leaves of the light hierarchy are point light sources or spotlight sources. The inner nodes could be affine transformation nodes used to specify the properties of a light source and light bounding volumes of sub-trees, coupled with Union set operators used to combine all the other nodes for describing the complicated light situation. In Figure 3.2, we show an example of such a light hierarchy.

3.5 Shadow Testing with the Light Hierarchical Structure

Visibility testing is the most time-consuming part of a global illumination calculation, and the visibility of light sources is particularly important since they determine the initial lighting distribution. If we could assume that all of the light sources were visible at every point, the calculation would reduce to a few simple operations. Unfortunately, it is almost always necessary to check for occlusions from light sources. And if there are multiple light sources in the scene, shadow determination quickly becomes the dominant computation of renderings.

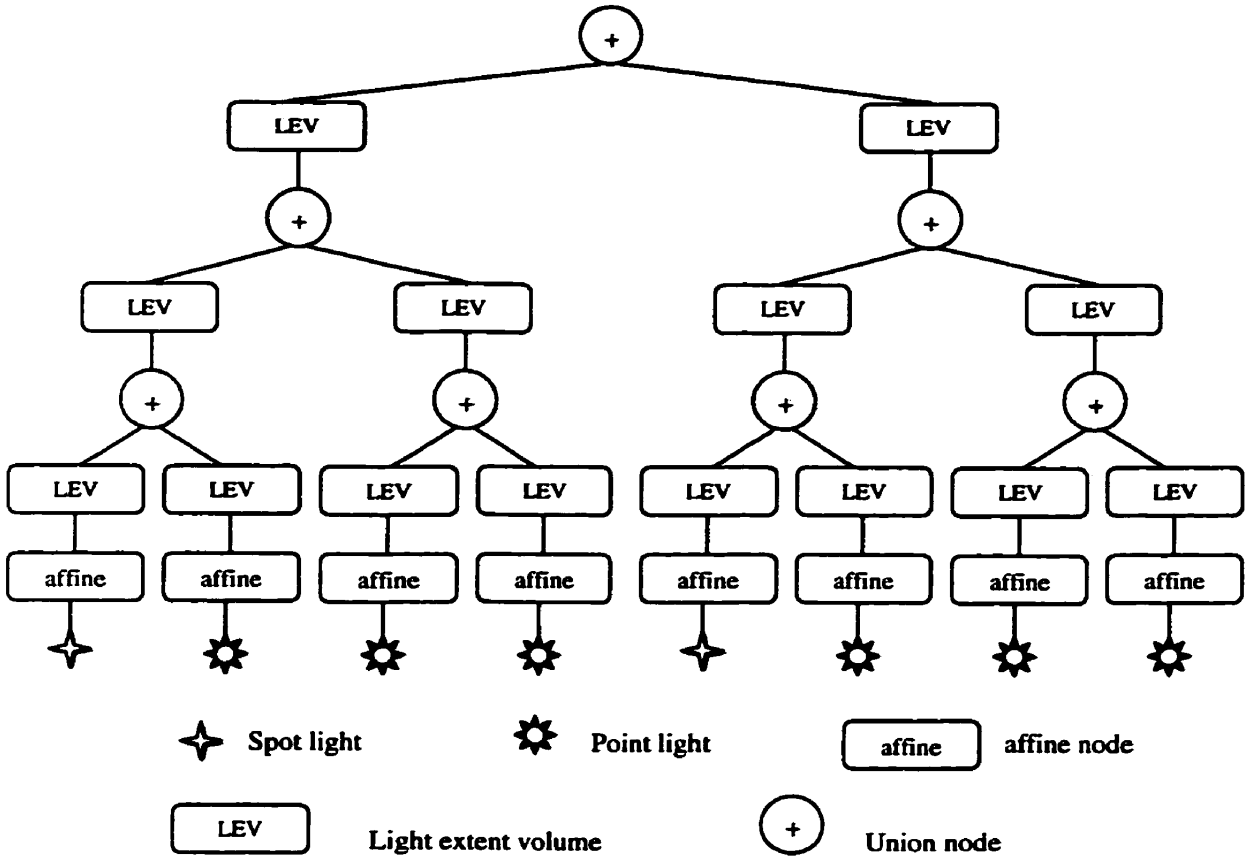


Figure 3.2 a light hierarchy example

In this section we develop an algorithm allowing us to use the hierarchy of light extent volumes to do quick shadow testing. The algorithm requires very little storage, and produces no visible artifacts. Furthermore, the users can control the accuracy and reliability of the technique, adapting it to suit their requirements. If the user specifies the tolerance of the light extent volume of zero, the algorithm degenerates to the original case, providing straightforward validation and comparison of results.

3.5.1 Basic Procedure

The basic procedure behind the idea of shadow testing with the light hierarchy consists of two steps. The first step is to pick up the important light sources from all the light sources in the scene at each intersection point. And then, since the unimportant light sources have negligible influence to the scene, only those important light sources are sent shadow rays to in the recursive ray tracing for calculating the light intensity of that point.

The most difficult part of this approach is how to characterize important light sources versus unimportant light sources. In the following sections, we will provide algorithms to efficiently identify important light sources from those whose influence to the scene are negligible in light hierarchies.

3.5.2 Algorithm of Identifying Important Light Sources

Figure 3.3 is an outline of the algorithm designed to identify important light sources from negligible sources in the scene.

The recursive function *CreateImportantLightList* firstly takes the position of intersection point, *point*, and the root node of the light hierarchy, *node*, as the input. Each leaf node of the light hierarchy is a point light source or a spotlight source, while interior nodes could be affine transformation nodes used to define the properties of light sources, light extent volumes bounding subtrees or union set operators combining branches of the light hierarchy. The boolean function *isInsideLightExtentVolume* called from within *CreateImportantLightList* is responsible for checking whether the given intersection point is inside the light extent

```
LightList *CreateImportantLightList( point, node )
begin
        switch (node.type) {
        case Point light:
        case Spotlight:
              return current light source
        case Affine:
              newPoint = inverseAffine(point, node.affine)
             lightlist = CreateImportantLightList (newPoint, node.child)
             return applyAffine(lightList, node.affine)
        case light extent volume:
              if isInsideLightExtentVolume (point, node.volume) then
                    return CreateImportantLightList (point, node.child)
              else return NULL
        case Union:
              leftList = CreateImportantLightList ( point, node.leftChild )
             rightList = CreateImportantLightList ( point, node.rightChild )
             add leftList and rightList into ImportantLightList
             return ImportantLightList
        }
end
```

Figure 3.2 algorithm designed to identify important light sources from unimportant light sources in light hierarchies without instancing

volume. When the test fails, the branch bounded by this light extent volume is immediately discarded. All the light extent volumes in the light hierarchy are evaluated in the local coordinate, whereas the intersection point is located in the world coordinate. In order to correctly check whether the intersection point is inside a light

extent volume, every time when we meet an affine transformation node during the traversal, we need to transform the intersection point with its affine transformation information. The procedure *inverseAffine* is called to inverse the affine transformation on the intersection point if we meet the affine transformation node during the traversal. This procedure is designed to guarantee that the intersection point and the light extent volume are in the same coordinate when checking whether the intersection point is inside that light extent volume. Besides, the procedure *applyAffine* is designed to calculate a light's attributes according to the affine transformations stored in its parent nodes.

To determine the important light sources, the function checks to see whether the intersection point is inside the light sources' extent volumes by traversing the hierarchy of light extent volumes from top to bottom. It examines the light extent volume bounding a certain branch of the light hierarchy tree. If the intersection point does not locate inside that light bounding volume, the rest of the light hierarchy can be ignored. Otherwise, it will keep on examining the child branches until the intersection point is not inside the light extent volume or it reach the leaves of the light hierarchy which contains the light sources. The sources whose light extent volumes surround the intersection point are the important lights we are seeking. All the important light sources for that intersection point are stored in a linked list. The list of the important light sources is then used in a recursive ray tracing algorithm, and only those sources are important enough to send shadow rays to.

By specifying the tolerance of light extent volumes, the users can control the accuracy of the technique to suit their requirements. By making tolerance sufficiently small, light sources' extent volumes have reasonable regions of influence to the scene. So our ray tracing results will be very close to those of the full calculation without the associated cost.

3.5.3 Optimization of the Light Hierarchy Structure by Instancing

Scenes with a large number of light sources are becoming increasingly important, such as outdoor scenes, urban settings, and opera and theater applications. In those complicated scenes, the same submodel containing light information may occur repeatedly with different transformation attributes. So we can use the instancing property to create compact representation of the scene. By instancing the light hierarchy only needs to create one master copy of each new light subtree that can subsequently be instanced instead of making multiple copies of the same subtree. Instancing can quickly create a large complicated light hierarchy from the one simple light subtree without using the same amount of memory.

A light hierarchy with instancing has two kinds of nodes referring to light information: light nodes and instance nodes. Light nodes indicate point light sources or spotlight sources, while instance nodes represent light combinations found in master object instances. And the master copy of an instanced object is symbolized by a master node that contains a light extent volume bounding the whole subtree. By the technique of light subtree instancing, the light hierarchy in Figure 3.2 can be modified

to the tree shown in Figure 3.4. It's obvious that instancing saves a lot in both space and time for generating the light hierarchy tree.

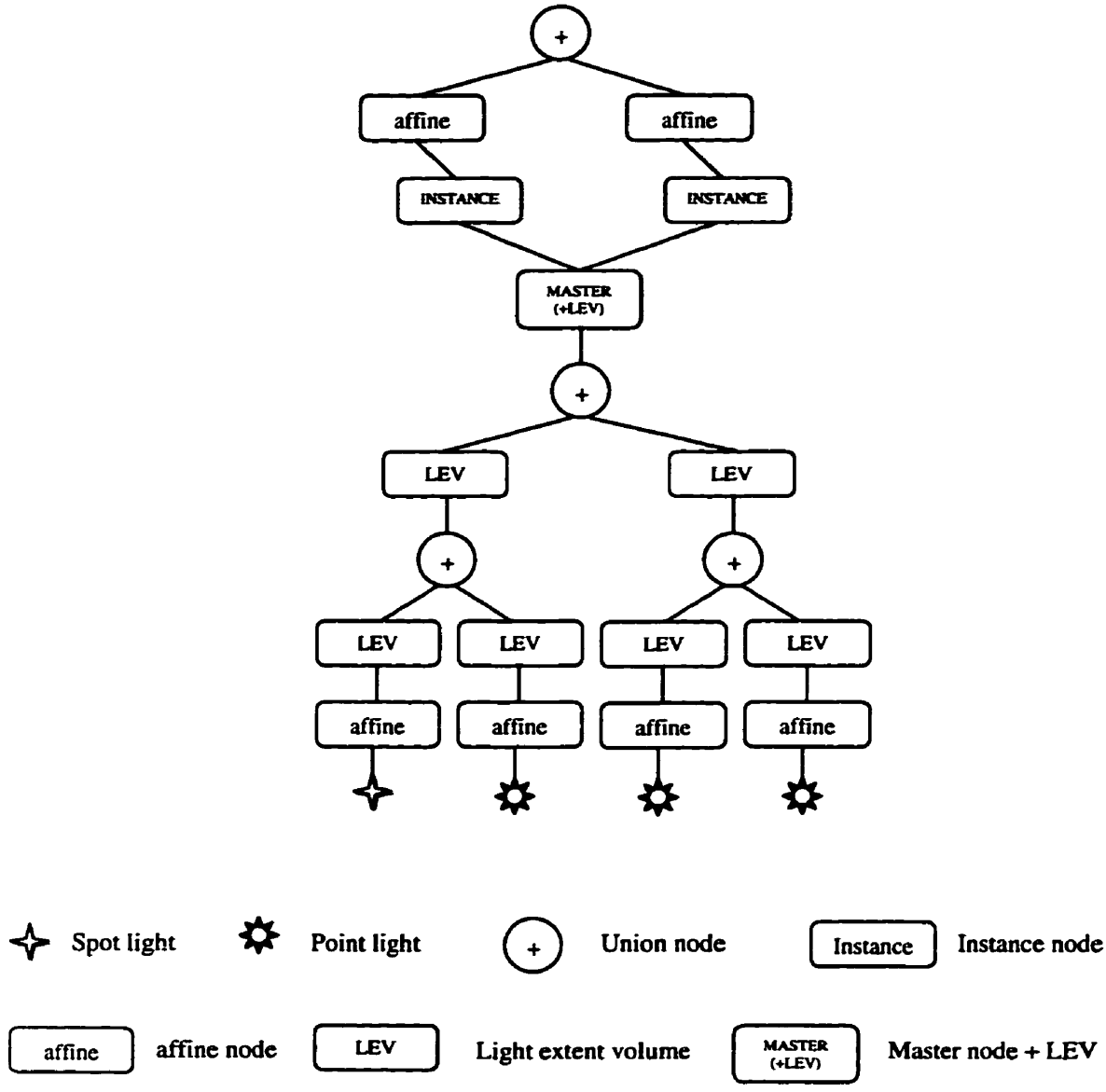


Figure 3.4 light hierarchy with instancing where only one master copy of light subtree is created

3.5.4 Algorithm for Light Hierarchies with instancing

In light hierarchies with instancing, it is possible that light sources can be in master objects. Once these master objects are instanced in the scene, the same light subtrees are potentially replicated many times. So we need to modify our algorithm to handle this. The approach to efficiently pick up important light sources from light hierarchies with instancing is shown in Figure 3.5.

```
LightList *CreateImportantLightList( point, node )
begin
        switch (node.type) {
        case point light:
        case spotlight:
                return current light source
        case affine transformation node:
                newPoint = inverseAffine(point, node.affine)
                lightList = CreateImportantLightList(newPoint, node.child)
                return applyAffine(lightList, node.affine)
        case instance node:
                return CreateImportantLightList (point, node.child)
        case master node:
        case light extent volume:
                if isInsideLightExtentVolume (point, node.volume) then
                        return CreateImportantLightList (point, node.child)
                else return NULL
        case Union:
                leftList = CreateImportantLightList ( point, node.leftChild )
                rightList = CreateImportantLightList ( point, node.rightChild )
                add leftList and rightList into ImportantLightList
                return ImportantLightList
        }
end
```

Figure 3.5 algorithm designed to identify important light sources from unimportant light sources in light hierarchies with instancing

Chapter 4

Implementation

4.1 Introduction

A distribution ray tracer originally written by John Amanatides and Don Mitchell was extended to include our new approach called *Light Extent Volumes* (LEV) and tested on a SUN ULTRA10 workstation to compare its performance to other ray-tracing multiple light approaches, including the traditional ray tracing and Ward's Adaptive Shadow Testing. This ray tracer takes a scene described in Scheme, a variation of Lisp, as input. Its implementation is basically divided into two parts as preprocessing and rendering. In order to understand the implementation well we first discuss the structures for modeling.

4.2 Structures for Modeling

Structures for modeling are the foundation of the implementation. In this ray tracer, a scene is composed of three parts: a view camera, light sources and solid objects. In each scene, there only exists a single view camera. The information of light sources is stored in a light hierarchy tree. Objects are represented by an object

CSG tree. A novelty of our ray tracer is that instancing is used in both the light hierarchy and the object CSG tree by only creating one master copy of an instanced object instead of making multiple copies of previously defined objects.

4.2.1 CSG Nodes in the Object Tree

In the implementation, the information of each scene is originally specified and stored in a CSG tree. In the CSG tree different types of CSG nodes are designed to aid in describing any possible complicated scenes, such as object nodes, instance nodes, master nodes, affine transformation nodes, set operation nodes, shading nodes and miscellaneous nodes.

4.2.2 CSG Nodes in the Light Hierarchy

There are seven types of CSG nodes used for the construction of the light hierarchy, including Point light node, Spotlight node, Instance node, Master node, Light Extent Volume node, Affine node, and Union node.

- **Point Light Node and Spotlight Node**

  A point light source or a spotlight source is firstly defined at the origin with an intensity value of 1. Light decay is modeled as $1/r^2$. It can be transformed like any other object in the scene. By using a scale transformation, a light source can be defined with any light intensity value.

  A spotlight is by default modeled as a cone with a generating angle of ninety degrees surrounding a light direction up along the z-axis. Each spotlight node can provide control over the concentration of the light by specifying its *dropoff*

exponent. After increasing the value of dropoff, the light becomes more concentrated around the light direction. By using a scale transformation, the generating angle of the spotlight cone can be varied. Moreover, after a rotate transformation the spotlight can be oriented in any desired direction.

- **Instance Node and Master Node**

  In many complicated scenes, the same objects occur repeatedly with different affine attributes. So when generating the light hierarchy or object CSG tree, the ray tracer only creates the master copy of a new object which can subsequently be instanced. By instancing one can quickly create a large complicated or repetitive scene from one simple object (eg. Instancing one seat to create a theatre auditorium).

  In the light hierarchy a master object is symbolized by a Master node as its subtree's root. Each Master node contains a light extent volume bounding the current master object that can accelerate light extent volume computation in the preprocessing stage and traversal in light hierarchy in the rendering step. Once the master object is created, all instances are represented by instance nodes referring to the master copy with their own transformations.

- **Light Extent Volume Node**

  It defines a light extent volume bounding the indicated sub-light-hierarchy. In order to achieve traversal acceleration, light extent volumes inside the sub-light-hierarchies become necessary. Light extent volume nodes allow us to bound any sub-light-hierarchies as desired.

- **Affine Transformation Nodes**

  Since all light sources firstly rest at the origin, several basic affine transformations become necessary to represent the light sources at different positions with different physical attributes. The affine transformations include translation, rotation, and scaling. A translation node is used to move the light instances along x, y, and z axes. A rotation node can rotate the light direction of a spotlight object about a certain axis by the indicated angle, and make it orient to any desired direction. A scaling node is designed to scale a light object about the origin along x, y and z axes to change its light intensity or modify a spotlight's cone size.

  Moreover, in light hierarchies with instancing, affine transformation nodes working with instance nodes are essential to represent different instanced objects.

- **Union Set Operation Node**

  It performs the union operation on the indicated sub-light-hierarchy trees by setting two sub-light-hierarchies as the union node's left and right child respectively. The result is a new sub-light-hierarchy tree that can be further manipulated.

With the aid of the above CSG nodes, we can create light hierarchies that may describe any possible light situations in scenes.

4.3 Preprocessing

In the preprocessing stage, the distribution ray tracer reads the input data from Scheme script files. The data include the camera's position and orientation,

information of light sources, solid objects' locations, orientations and surface properties, resolution of image and output device.

As the information is being read, an original CSG tree is generated, which is a composite tree, including the information of camera, light sources and solid objects in the scene.

Key steps of preprocessing are as following:

1) Pull out the camera node from the original CSG tree
2) Pull out light nodes from the original CSG tree and generate a light hierarchy

- Light Node Extraction

When generating the light hierarchy, point light nodes and spotlight nodes with their affine transformation nodes are directly extracted from the CSG tree; however, instance nodes with their affine information are copied from the CSG tree. Each instance node created in the light hierarchy is set to link to its directly instanced subtree whose root is a master node. Besides, the light extent volume nodes are pulled out from the tree and added into the light hierarchy for traversal acceleration. In addition, union nodes are copied from the CSG tree into the light hierarchy to combine different branches.

- Light Extent Volume Computation

Once a primitive light hierarchy is constructed, light extent volumes in the tree are then computed. If the light hierarchy has instancing property, we firstly compute light extent volumes in all master sub-light-hierarchies and then those in

the top-level. There is a linked list designed to keep track of all the master objects in the hierarchy so that we can quickly access each master subtree.

For each master subtree, the computation of light extent volumes begins with a depth-first traversal of the sub-light-hierarchy. We keep on traversing the children of the node until we reach a leaf that contains a point light source or a spotlight source. Then an initial light extent volume of a sphere or cone resting at the origin is calculated based on the information about the light's intensity and the tolerance of light extent volume. When tracing back from the bottom to top, we need to modify that light extent volume based on the affine transformation nodes and union set operation nodes we meet. In case of an affine transformation node, we transform a half-computed extent volume by scaling, rotating, or moving as indicated by the affine node. For a union operation node, a new light extent volume is computed by merging two volumes bounding the child branches of the node. Light extent volume nodes and the light extent volume in the master node are set as volumes bounding their sub-light-hierarchies.

During the traversal for light extent volume computation, if we meet instance nodes, we only need to check the light extent volume stored in their child, a master node, and do not need to traverse the instanced subtree again and again. It will save us a lot in time for the computation.

3) Generate an object CSG tree

After pulling out camera node and light nodes, the ray tracer firstly repairs the CSG tree, then computes the object bounding boxes of the tree, and finally

optimizes the tree links for shader. After all these are done, an object CSG tree is generated.

After preprocessing, the information in the original CSG tree goes into three parts: a view camera, a light hierarchy and an object CSG tree. Figure 4.1 shows the graphical representation of preprocessing stage.

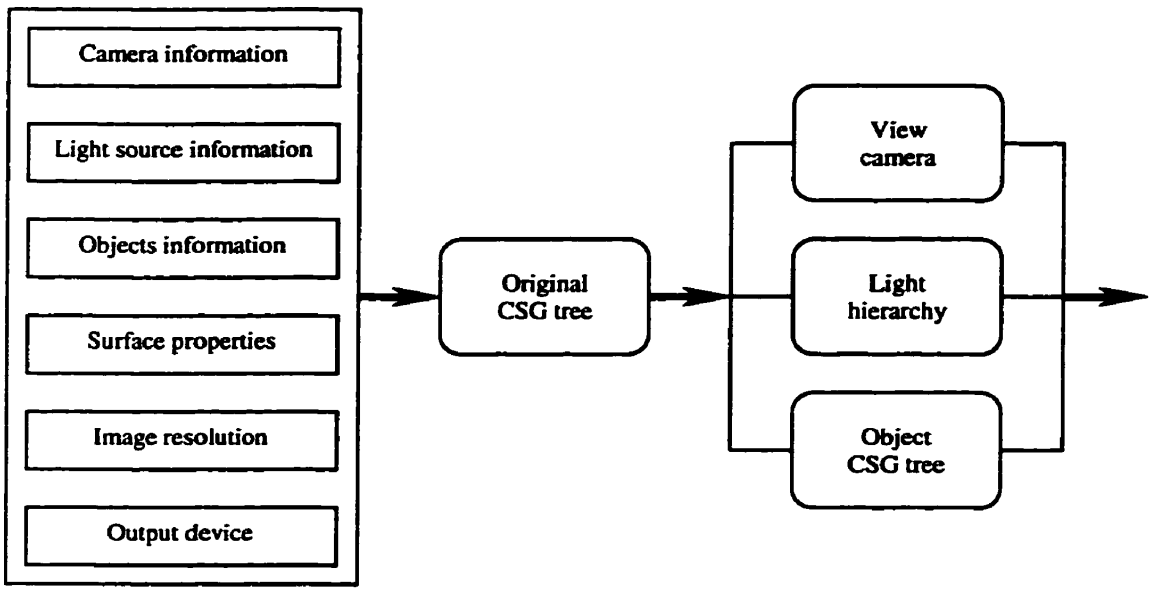
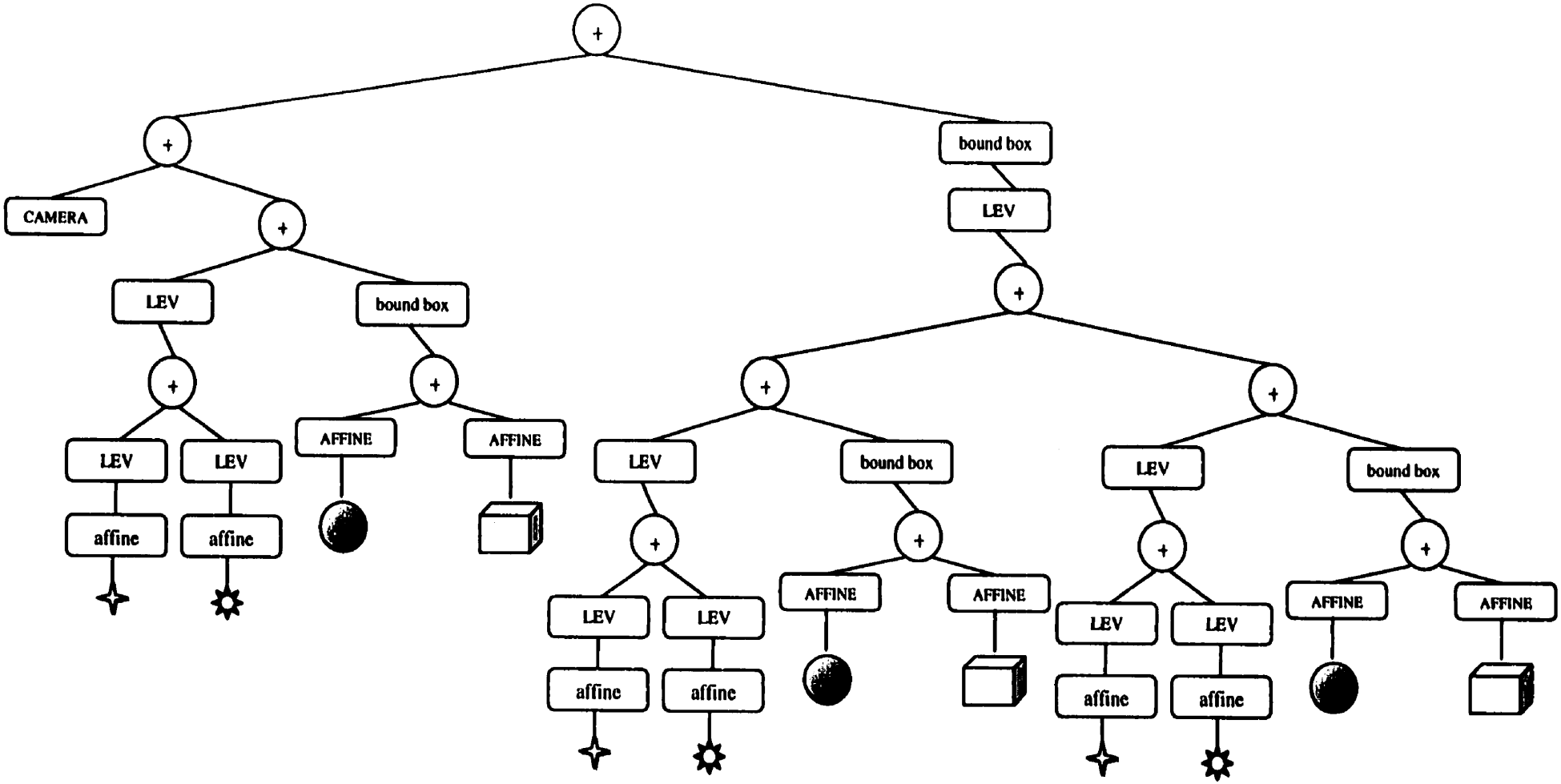
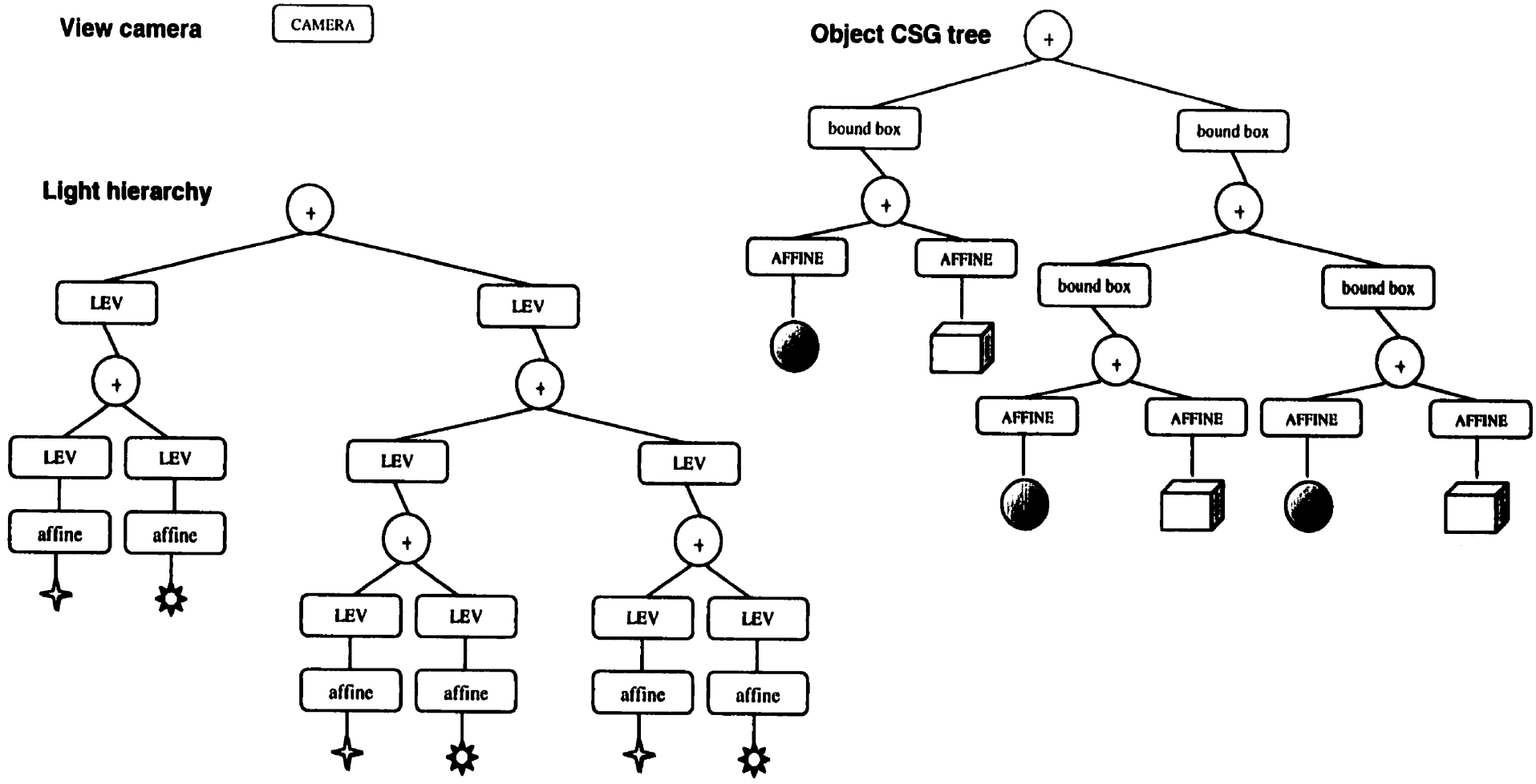


Figure 4.1 preprocessing pattern

In the following, we construct a simple model in two different ways: one is without instancing, and the other is with instancing. Each example illustrates the original CSG trees of the scene model and the generation of the light hierarchy and the object CSG tree after preprocessing, as shown in Figure 4.2 and Figure 4.3 respectively.

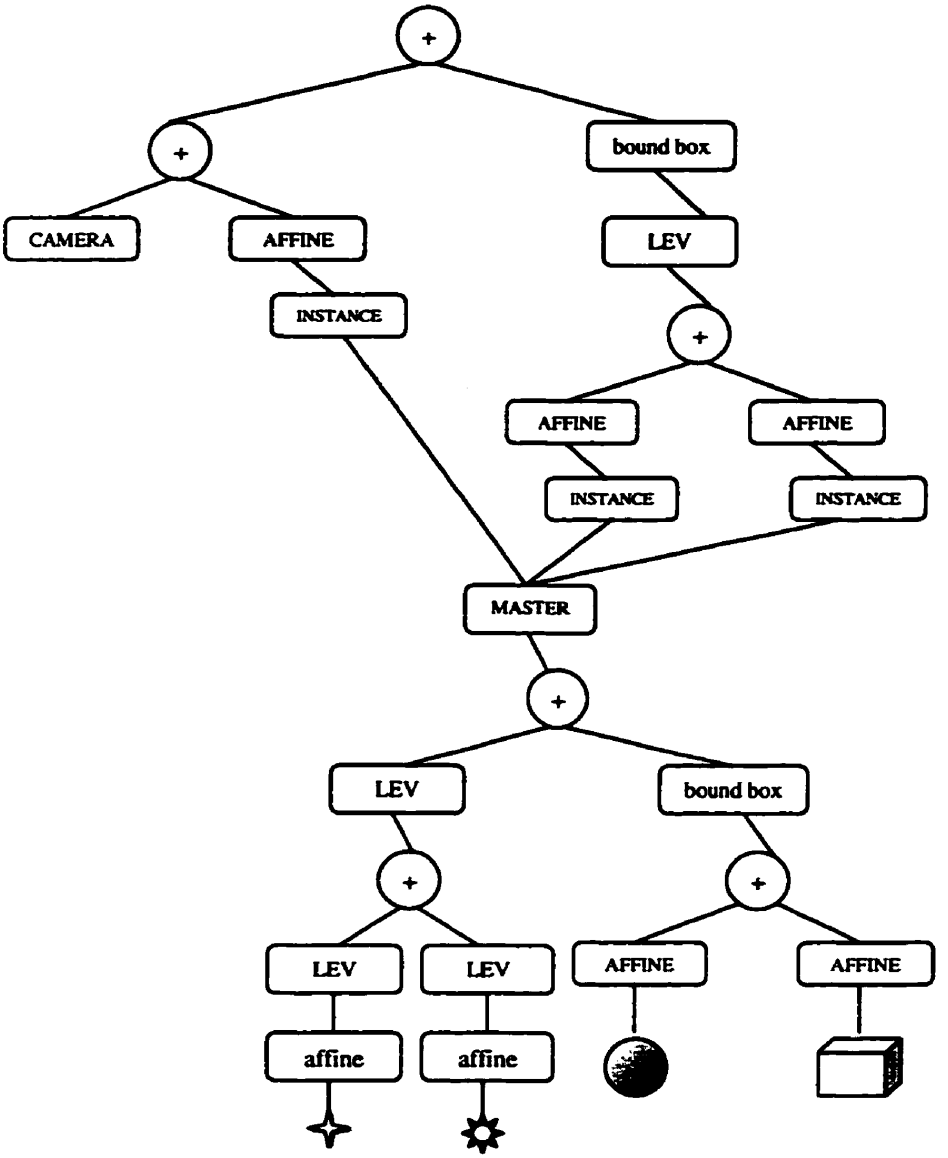


(a) an original CSG tree of a simple scene model constructed with instancing

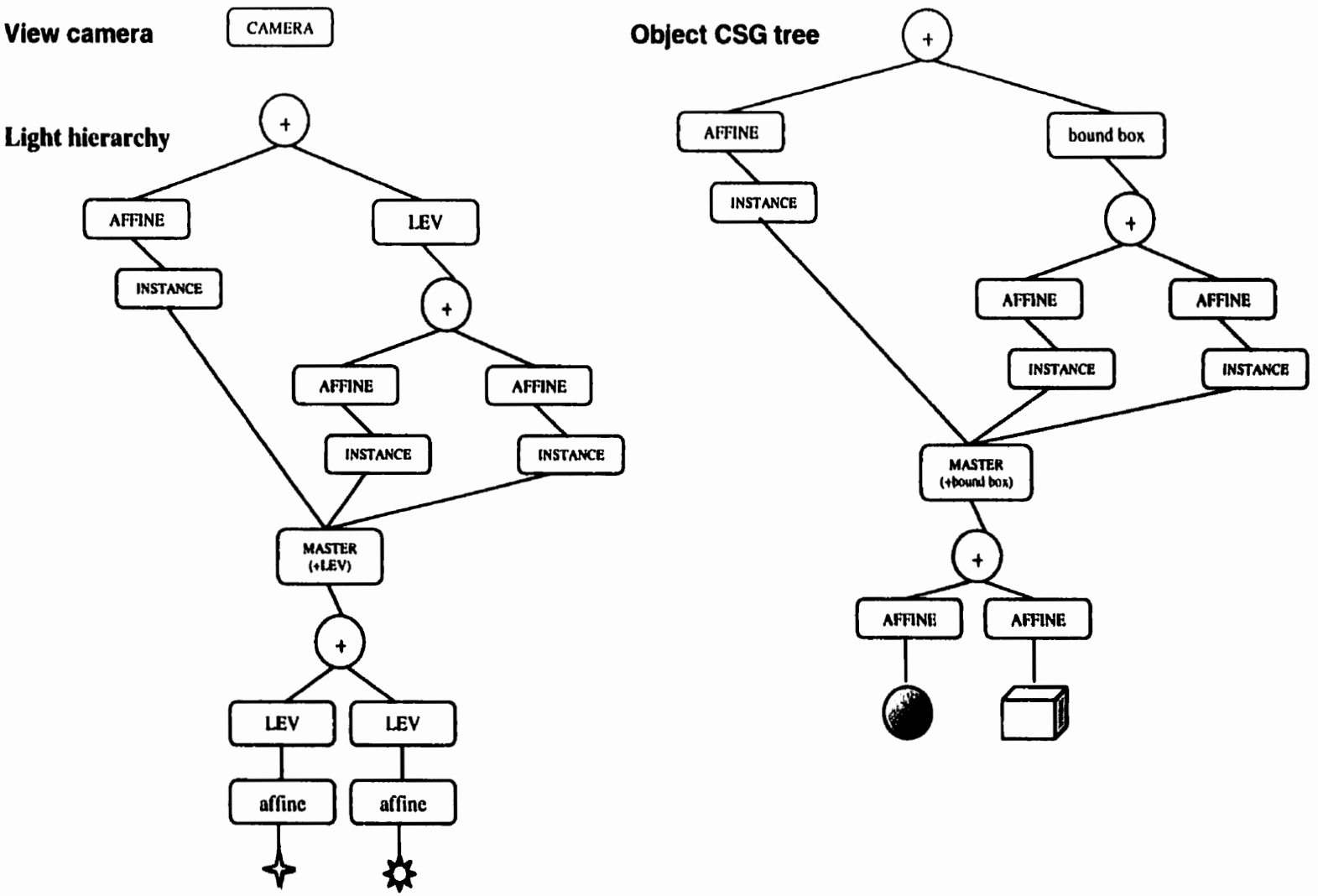


(b) scene model after preprocessing

Figure 4.2 preprocessing on a scene model constructed without instancing



(a) an original CSG tree of a simple scene model constructed with instancing



(b) scene model after preprocessing

Figure 4.3 preprocessing on a scene model constructed with instancing

4.4 Rendering

The ray tracer starts to trace individual pixels after the preprocessing computation is done. Rays are shot from view camera into the scene and each ray passes through a pixel. With the hierarchical bounding volume technique, we first examine the bounding volume in the root node of the object CSG tree. If the ray does not intersect with that bounding volume, the rest of the CSG hierarchy tree can be ignored. Otherwise, we will keep on examining the children of the node until the ray misses or we reach the leaves of the hierarchy tree which contain the objects. Those objects stored in those leaves are then tested with the ray.

Once the intersection point is determined, this ray tracer uses the hierarchical tree of light extent volumes generated in the preprocess step to quickly find out all the light sources that are important enough to do the shadow tests. Beginning with the root, we take the intersection point to check whether the intersection point is located in a light bounding volume. If it is, then go down the light hierarchy tree; otherwise the rest of the branches of the light hierarchy may be trivially rejected. When we reach the leaves of the light hierarchy, light sources stored in those leaves are selected as important light sources.

After the list of important light sources at an intersection point is generated, a recursive ray-tracing algorithm with the global illumination model is applied to calculate

the light intensity of this intersection point. If the surface is reflective or refractive, a reflection or refraction ray is shot from the intersection point. The shading of each pixel is based on the Phong illumination model as well as refraction or reflection. The final value of the pixel is output in PPM (Portable Pixmap) format.

4.5 Simplicity and Compatibility of the LEV algorithm

An important feature of the LEV method is its simplicity of implementation. The algorithm that provides a great reduction in shadow testing time can be written in less than a page of C code; the shading function itself is unchanged. Furthermore, the approach is compatible to most other global illumination techniques, and can be added to existing direct light calculations and optimizations. When the algorithm is added to an existing ray tracer, the only modification made is to generate a light hierarchy of all light sources in the scene in the step of preprocessing, and then in the rendering stage at each intersection point we only send shadow rays to important light sources that can be identified by traversing the light hierarchy.

4.6 Implementation of Ward's AST method

In order to compare the performance of the LEV algorithm with other approaches, we also implemented Ward's AST method in our distribution ray tracer. Its pseudo code is shown in Figure 2.8. First, by traversing the light hierarchy constructed in the preprocessing stage, a linked list is generated to store all the light sources in the scene.

And then, the light list is passed to the shade module. In the shade module, first compute potential contributions from all light sources at a certain intersection point, and then sort the contributions in descending order. Only significant light sources appeared in the first part of the sorted light list are selected to do visibility test, while negligible light sources in the rest of the list simply make estimated contributions to the scene. The whole point of the algorithm is to minimize the number of light sources that must be tested for visibility.

Chapter 5

Testing and Results

5.1 Overview

This chapter lists a set of test cases which allows us to evaluate our new approach of Light Extent Volumes in several different aspects.

5.2 Testing Aspects

In this thesis different types of testing were performed to examine the new approach of the hierarchical tree of light extent volumes and compare it with other ray-tracing multiple light approaches, including the traditional ray tracing and Ward's Adaptive Shadow Testing.

Testing includes the following aspects:

1. Compare Light Extent Volumes with the other two approaches which include the traditional ray tracing and Ward's Adaptive Shadow Testing.
   - Compare the performance of the Light Extent Volumes method with the other two existing approaches.

- Compare extra memory usage of different approaches that require additional data structures for shadow testing acceleration.

2. Examine some characteristics of the Light Extent Volumes method.
   - Check the algorithm's running time behavior
   - Check the fraction of light sources needed for shadow testing when the algorithm is applied with different values of tolerance.
   - Check the changes of the fraction of light sources for shadow testing when the algorithm is tested with increasing number of light sources in the scene.
   - Check the average and maximum pixel error corresponding to the different tolerance, as compared with a fully tested source calculation.

A distribution ray tracer was implemented and tested on a SUN ULTRA10 300MHz 256Mb workstation. When comparing the performance of different approaches, the timing is for the whole process that includes the preprocessing and rendering stages of an approach, and it was evaluated by the system /bin/time. Each test was repeated three times and the average value was used as the final data.

5.3 Test Scene Description

Following are the test scene models used in the testing.

- **Scene Model 1**

The test scene model is a square building with recursively instanced rooms. In each room there is a light in the ceiling shining on two robots riding unicycles. A complete view of the building with 64 light sources is shown as Figure 5.1.

Scene models with more light sources are generated by increasing the model's recursive level. Note that scene models with 256, 1024, 4096, and 16,384 light sources are shown in Appendix A.

Figure 5.1 Test Model 1: a complete view of the building with 64 light sources (image resolution: 320 * 240)

- **Scene Model 2**

The scene model is a peano-curve maze with recursively instanced robots riding unicycles where each robot is shone by a streetlight. Figure 5.2 illustrates the top view of the peano-curve maze at recursive level 1 with 45 light sources. We increase light sources in the scene model by enhancing the recursive level of the peano-curve maze. The scene models with more light sources are shown in Appendix A.

Figure 5.2 Test Model 2: a top view of the peano-curve maze with 45 light sources (image resolution: 320 * 240)

- **Scene Model 3**

The scene model is a performance hall where there are multiple spotlights in the ceiling shining on three robots riding unicycles. For the purpose of comparison, each time we increase the number of spotlights in the scene by replacing a spotlight with four smaller ones. The scene model with 64 spotlight sources is shown as Figure 5.3.

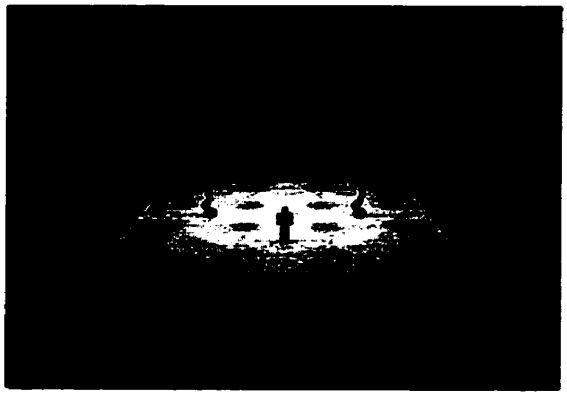


Figure 5.3 Test Model 3: the performance hall where there are multiple spotlight sources in the ceiling shining on three robots riding unicycles (image resolution: 320 * 240)

It's worthy to mention that every object in Scene Model 1 and Scene Model 2 is illuminated by relatively a few of the candidate light sources, while in Scene Model 3 every object is shone by most of the light sources in the scene. Since the LEV algorithm assumes that in such scenes relatively few lights will illuminate any individual surface, we expect that our algorithm will work well for Scene Model 1 and Scene Model 2 and fail in Scene Model 3.

5.4 Test cases

According to the test aspects discussed in Section 5.2, we designed two sets of test cases to evaluate our new approach of Light Extent Volumes (LEV). The set of the first two tests compares our new approach with the traditional ray tracing (RT) and Ward's Adaptive Shadow Testing (AST). The second set of test cases examines some characteristics of our approach of the light hierarchical tree of light extent volumes. Note that all the output images in the following tests are rendered at the resolution of 320*240, and the data collected from the tests are listed in Appendix B.

5.4.1 Compare LEV with Traditional RT and AST

In the following test cases we compare the LEV method with the traditional RT and Ward's AST in such aspects as the performance and the extra memory space required for quick shadow testing.

5.4.1.1 Test 1.1 : Performance Comparison

We examine the performance of the LEV method by comparing it with the traditional RT and Ward's AST. We use variations of three scene models introduced

in Section 5.3, where scene model 1 contains a number of light sources ranging from 64 to 16,384, for scene model 2 the number of light sources is from 45 to 12,285, and scene model 3 contains light sources from 64 to 16,384. In each test, the tolerance we used is 0.01 light intensity. Since the actual error will be smaller than the required tolerance, the test runs result in images that are expected to visually indistinguishable from those computed with the traditional ray tracing.

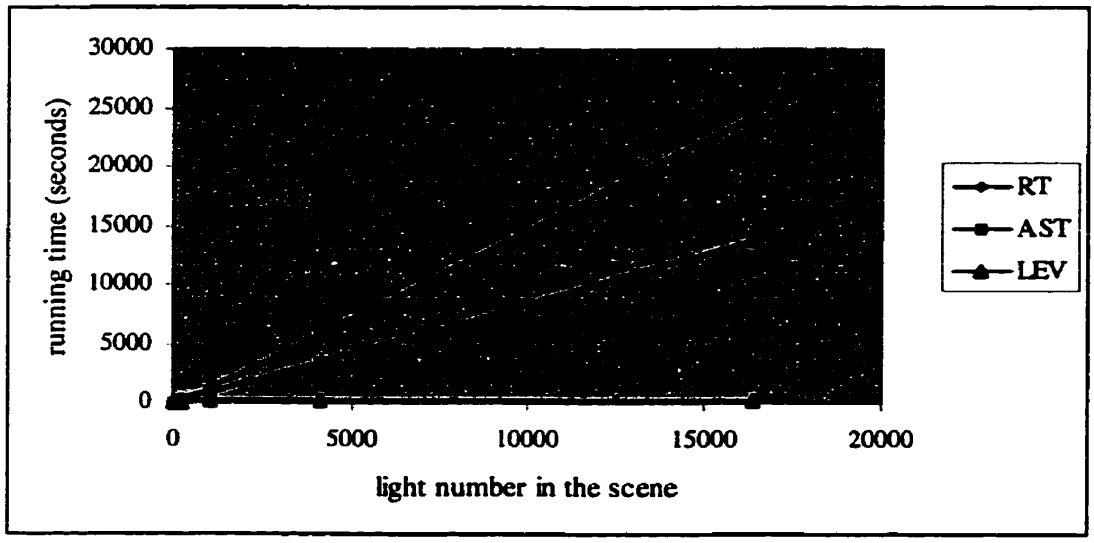
Tests were performed on three scene models. The result from scene model 1 is presented in Figure 5.4 (a) and (b), and the result from scene model 2 is in (c) and (d). The result tables in Figure 5.4 (a) and (c) show the cost of traditional ray tracing (RT), the cost of Ward's Adaptive Shadow Testing (AST), the cost of the Light Extent Volumes method (LEV), the cost percentage of Ward's method over the ray tracing method (AST/RT), the cost percentage of our method over the ray tracing method (LEV/AST), and the speedup achieved by our method over the ray tracing method (LEV/RT) respectively. Figure 5.4 (b) and (d) are the graphic representations of the performance comparison of three approaches based on scene model 1 and scene model 2 respectively. Besides, result images from scene model 1 and scene model 2 are visually indistinguishable from the original images.

The result we got from scene model 3 is far from those from scene model 1 and scene model 2. At the tolerance of 0.01, we tested the models with different light sources. There are two situations: either the running time by our LEV method is close to that by the traditional RT, or the method has great speedup, but the result image is

obviously different from the original one. We will examine more about scene model3 to explain this result in a later test.

LIGHTS IN THE SCENE	RT	AST	LEV	AST/RT	LEV/RT	LEV/RT SPEED UP
64 lights	98.86s	42.12s	46.38s	42.61%	46.91%	2.13
256 lights	373.77s	224.48s	78.96s	60.06%	21.13%	4.73
1,024 lights	1268.58s	812.23s	101.29s	64.03%	7.98%	12.52
4,096 lights	5807.60s	3380.42s	134.56s	58.21%	2.32%	43.16
16,384 lights	24289.28s	13665.89s	184.14s	56.26%	0.76%	131.91

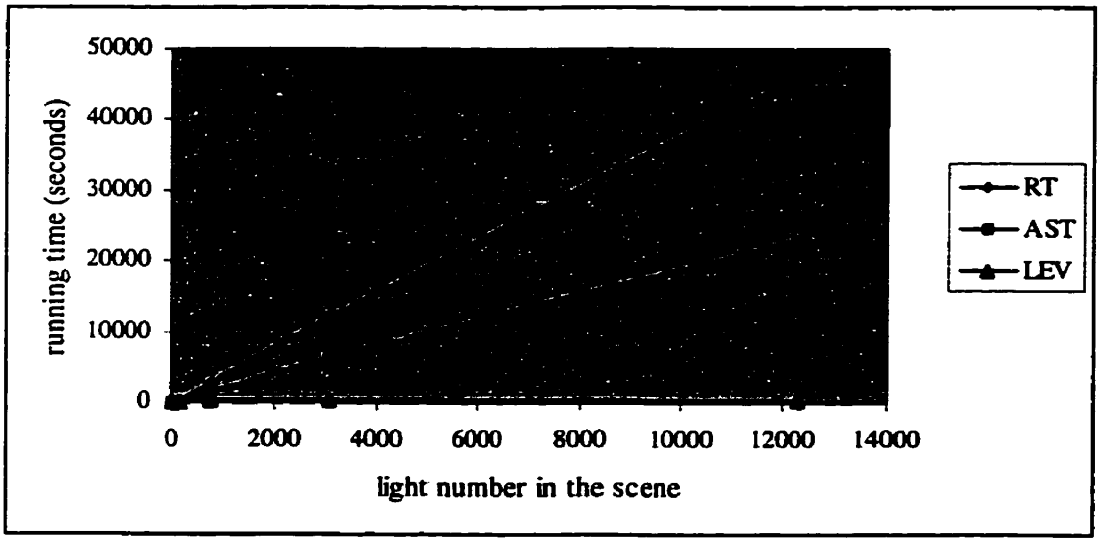
(a) result table from scene model 1 for performance comparison of three approaches. All timings are in seconds on a SUN ULTRA10 300MHz workstation.



(b) graphic representation of performance comparison on Scene Model 1

LIGHTS IN THE SCENE	RT	AST	LEV	AST/RT	LEV/RT	LEV/RT SPEED UP
45 lights	99.58s	39.66s	59.65s	39.83%	59.90%	1.67
189 lights	248.09s	132.53s	62.76s	53.42%	25.30%	3.95
765 lights	2569.58s	1359.05s	218.19s	52.89%	8.49%	11.78
3,069 lights	11989.90s	6155.61s	287.73s	51.34%	2.40%	41.47
12,285lights	46025.75s	22953.04s	296.19s	49.87%	0.64%	155.39

(c) result table from scene model 2 for performance comparison of two approaches. All timings are in seconds on a SUN ULTRA10 300MHz workstation.



(d) graphic representation of performance comparison on scene model 2

Figure 5.4 performance comparison of traditional ray tracing (RT), adaptive shadow testing (AST) and light extent volumes (LEV).

The performance comparison results based on scene model 1 and scene model 2 show that compared with the traditional RT, the LEV approach achieves significant speedups (up to 150 times), and for a high number of light sources, our approach consistently faster than Ward's AST method. The reason is that different from the LEV method, both traditional RT and the AST method behave linearly in rendering time. So for a high number of light sources, their rendering costs become critical.

5.4.1.2 Test 1.2 : Memory Overhead Comparison

Among three approaches, both AST and LEV require extra data structures for shadow testing acceleration. This case is to examine the extra memory space usage of those two methods. The comparison is based on the analysis of Scene Model 1.

In Ward's AST approach the storage overhead is a few additional words per light source for keeping track of test and hit counts. It's about 24 bytes per light source (2 double, 2 int).

The extra memory space usage caused by the LEV method is a hierarchy of CSG nodes that excludes light nodes and their directly related affine nodes used to further define lights' attributes. If the light hierarchy of scene model 1 is generated without instancing property, the memory overhead per light source is about 136 bytes, including a union node (16 bytes) and two light extent volume nodes(60 bytes each), where light extent volumes are fully used in the light hierarchy. However, our light hierarchy is constructed by instancing. So the hierarchy is quite compact. Since the tree keeps only one master copy of the instanced light subtree, compared with the

increase of the number of light sources in the scene model, the size of the light hierarchy grows very slowly.

Figure 5.5 shows the comparison of extra memory space used by AST and LEV. The memory overhead of AST is far less than that of LEV by the light hierarchy without instancing. However, when we use the light hierarchy with instancing, at a small number of light sources the storage overhead of AST is less than that of LEV, while for a high number of light sources the LEV method consistently has less storage overhead. So by constructing the light hierarchy properly, the space overhead of the algorithm can be minimal.

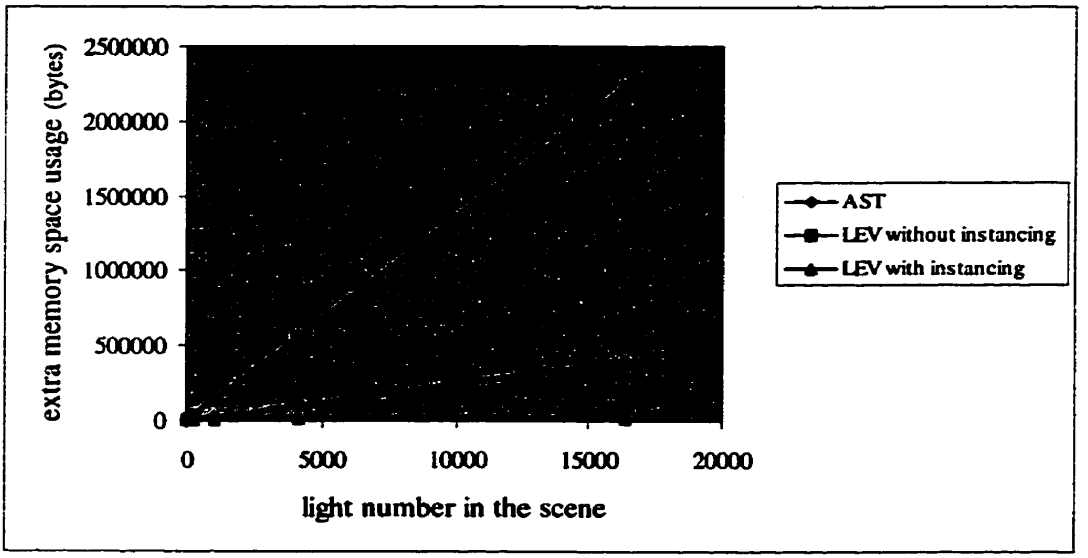


Figure 5.5 comparison of extra memory space used by Adaptive Shadow Testing (AST) and Light Extent Volumes (LEV) for shadow testing

5.4.2 Examine Characteristics of Light Extent Volumes

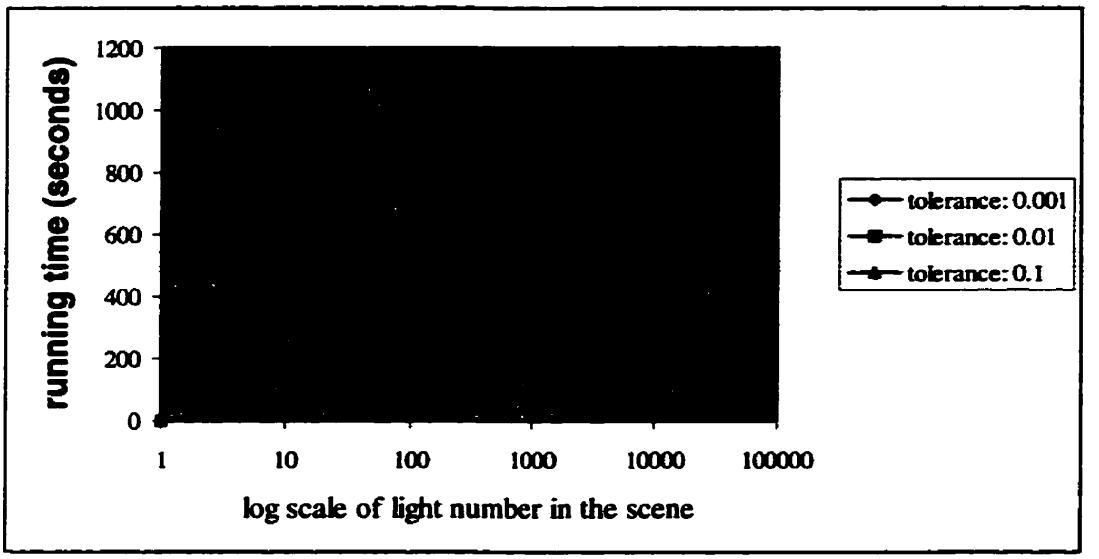
In this section, a set of test cases is designed to examine some characteristics of the LEV approach. Firstly, we checked the algorithm's running time behavior. And then, we checked the fraction of light sources tested for visibility when the algorithm is applied with different values of tolerance. In order to see what happens to the calculation as we increase the number of light sources in the scene, we then used variations of scene models containing more light sources and repeated the tests. At last we evaluate the quality of result images by means of average and maximum pixel errors.

5.4.2.1 Test 2.1 : Running Time Behavior

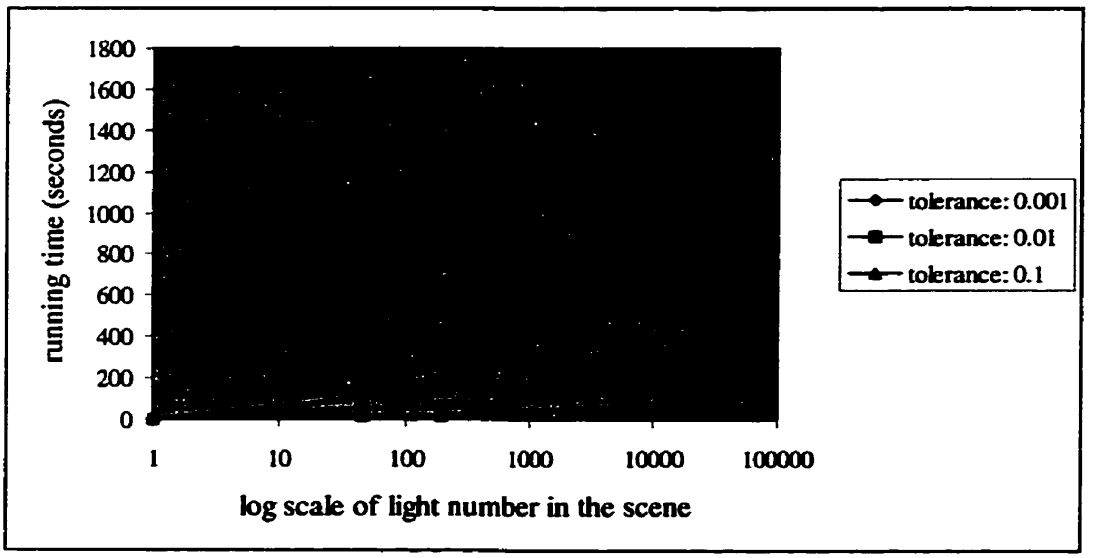
The purpose of the LEV method is to approach approximately *O(log n)* complexity for typical scenes in determining which light sources contribute significant irradiance to the intersection point. This computation complexity allows hundreds or even tens of thousands of light sources in a scene to be rendered in reasonable time.

We checked the algorithm's running time behavior by testing it with different number of light sources at different values of tolerance on scene model 1 and scene model 2. In Figure 5.6 (a) and (b), the horizontal axes of the graphs show the logarithmic scale of the number of light sources in the scene. Since the running time has the linear relation with the logarithmic scale of the light number in the scene, it demonstrates that when rendering scenes where every object is directly shone by a

relatively few of the candidate light sources, our algorithm has approximately logarithmic behavior in execution time.



(a) result based on Scene Model 1



(b) result based on Scene Model 2

Figure 5.6 run time behavior of the Light Extent Volumes method

5.4.2.2 Test 2.2 : Fraction of Light Sources for Shadow Testing

The highlight of our LEV method is that only those light sources that have significant contribution to the scene are considered for shadow testing. So the fraction of light sources tested for visibility to each intersection point is an important value because it directly determines the performance of the algorithm.

Figure 5.7 shows the fraction of light sources for shadow testing when the algorithm is applied with different values of tolerance of light extent volume and tested on three scene models. The horizontal axis of the graph shows the different values of tolerance. The vertical axis shows the ratio of average light sources tested per intersection point compared to all the candidate light sources in the scene. Note that with a target accuracy of zero (tolerance), all of the candidate light sources are tested. However, with a none-zero accuracy, the number of light sources tested for visibility varies with different intersection points. So we use the average light number tested per intersection point for that tolerance. The average value is computed by the following formulation as:

$$\frac{\Sigma(\text{the number of light sources tested at each intersection point})}{\text{the number of intersection points}}$$

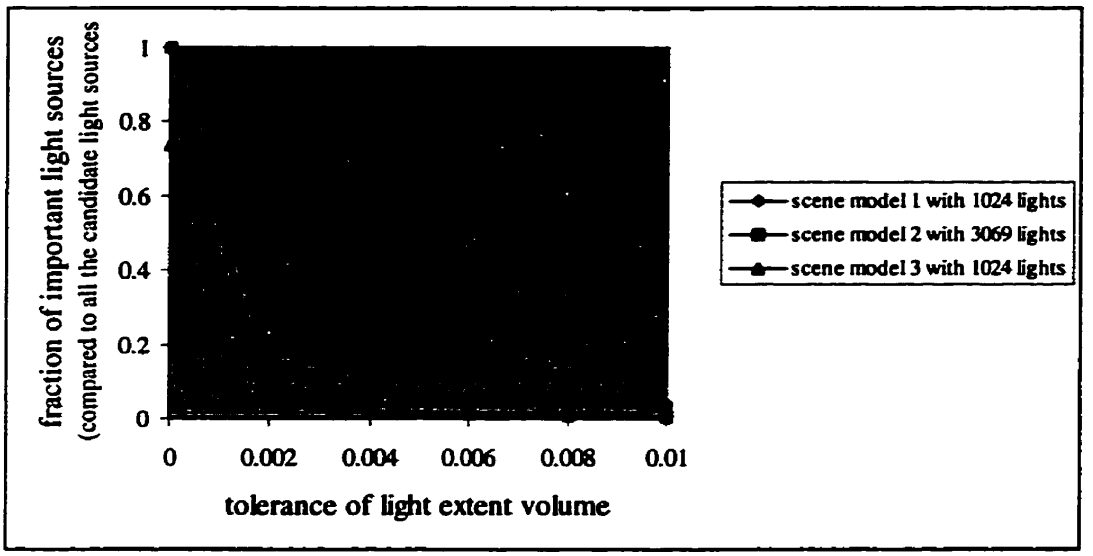


Figure 5.7 fraction of light sources for shadow testing

The fraction curves of scene model 1 and scene model 2 show that the average number of light sources tested per intersection point decreases smoothly with higher tolerance of light extent volume, while the curve of scene model 3 is quite different from those two.

In order to further check scene model 3, we did some extra tests on it. In model 3 most of light sources are important sources to any object surface. As shown in Figure 5.8, some parts of the curves keep consistent, that is, once the light extent volume is big enough, most light sources are regarded as important sources. Since most light sources in the scene are tested for visibility, the rendering time is almost linear to the number of light sources. And then the algorithm's performance degenerates to the traditional ray tracing, but on the other side the output image quality is satisfying. However, when the fraction of light sources is low (most of

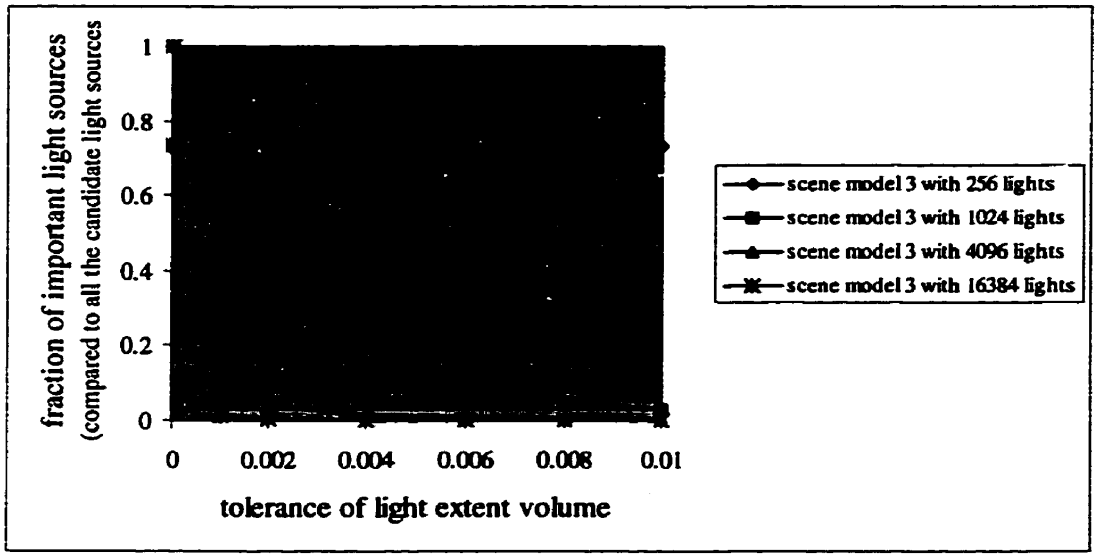
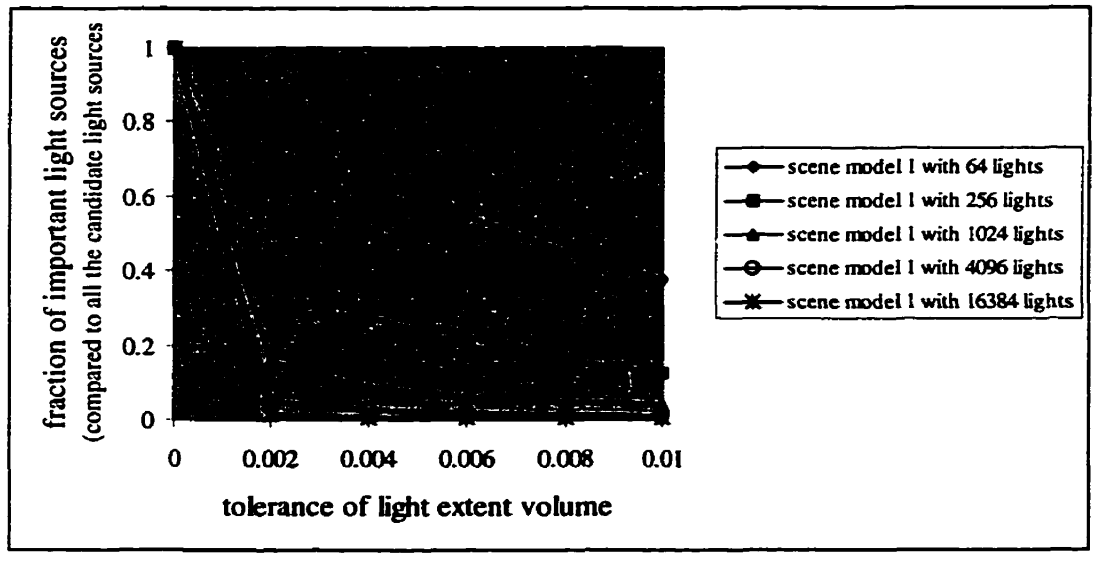


Figure 5.8 test result of fraction of light sources for shadow testing on Scene Model 3

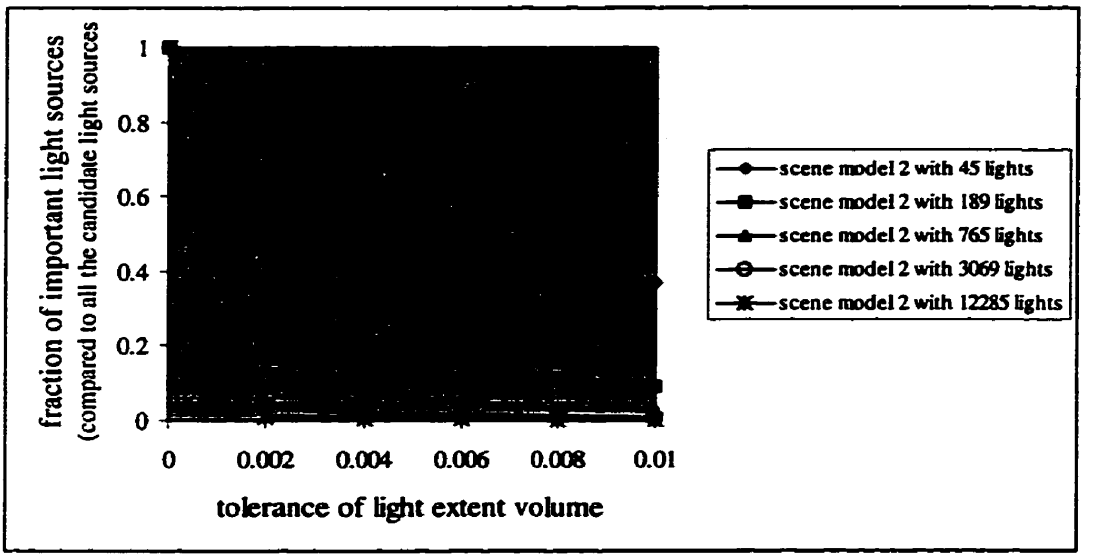
light sources are not regarded as important sources), though the algorithm can do quick shadow testing, there will be a great loss at the result image quality because many light sources that actually sum up to make significant contribution to the scene are ignored in the rendering. This test case proves that the LEV method does not work well for scenes like model 3 where every object is shone by most of the light sources in the scene. So the following tests will be performed only on scene model 1 and scene model 2.

5.4.2.3 Test 2.3 : Fraction of Lights Tested When Increasing Light Sources

In order to see what happens to the above calculation as we increase the number of light sources, we used scene model 1 and scene model 2 with more light sources and repeated the tests. The resulting fractions of shadow tests for the modified scene model 1 and scene model 2 are shown in Figure 5.9 (a) and (b) respectively. In both cases, the most noticeable difference is that the overall drop in the fractions of sources tested, which indicates that the algorithm's performance improves as light sources are added to the scenes. The running time still takes longer of course.



(a) result based on Scene Model 1



(b) result based on Scene Model 2

Figure 5.9 fraction of important light sources when increasing light sources

5.4.2.4 Test 2.4 : Average and Maximum Pixel Errors of Rendering Images

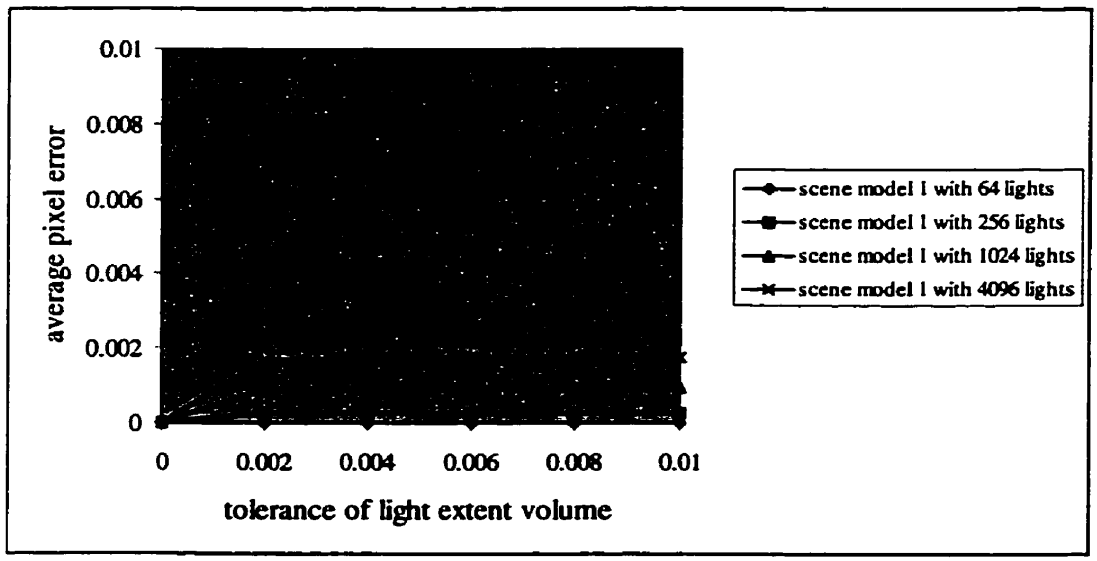
Since only important light sources are selected for shadow testing, what concerns us most is then whether the image quality is satisfying. In this test case, we use *pixel error* to evaluate the quality of an image generated by the LEV method, as compared with a fully tested source calculation. The output image from the distribution ray tracer characterizes each pixel value into three channels as R, G, B. Thus, the *pixel error* can be defined as following:

Assume that the value of a pixel by a fully tested source calculation is (R_f, G_f, B_f) and the value of the same pixel calculated from partially tested light sources is (R_p, G_p, B_p), the error of that pixel caused by partial shadow testing is the sum of the

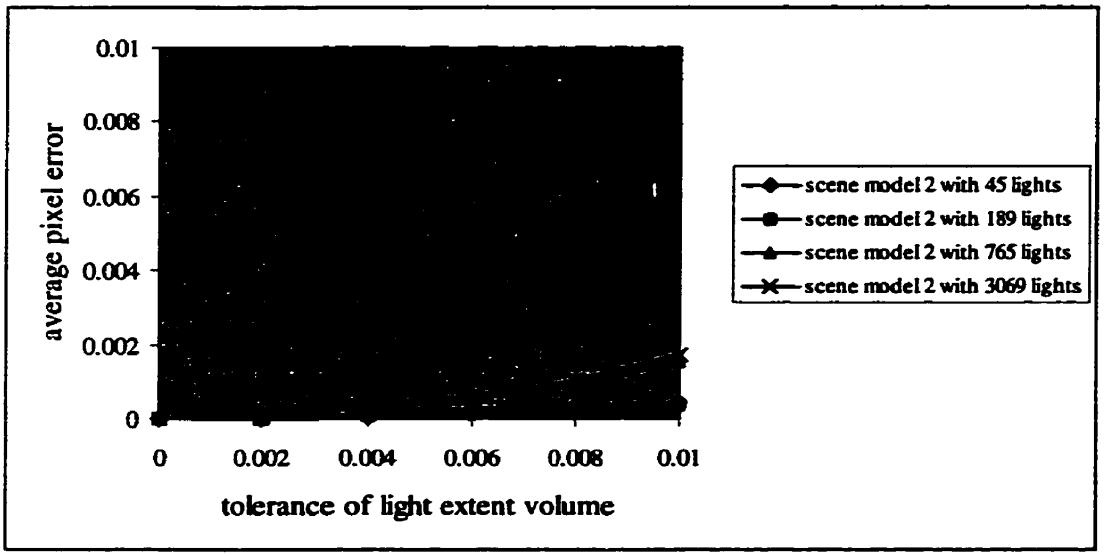
absolute value of the difference of each color channel. Its formulation can be written as:

$$pixel\ error = \mathtt{abs}(R_p - R_f) + \mathtt{abs}(G_p - G_f) + \mathtt{abs}(B_p - B_f)$$

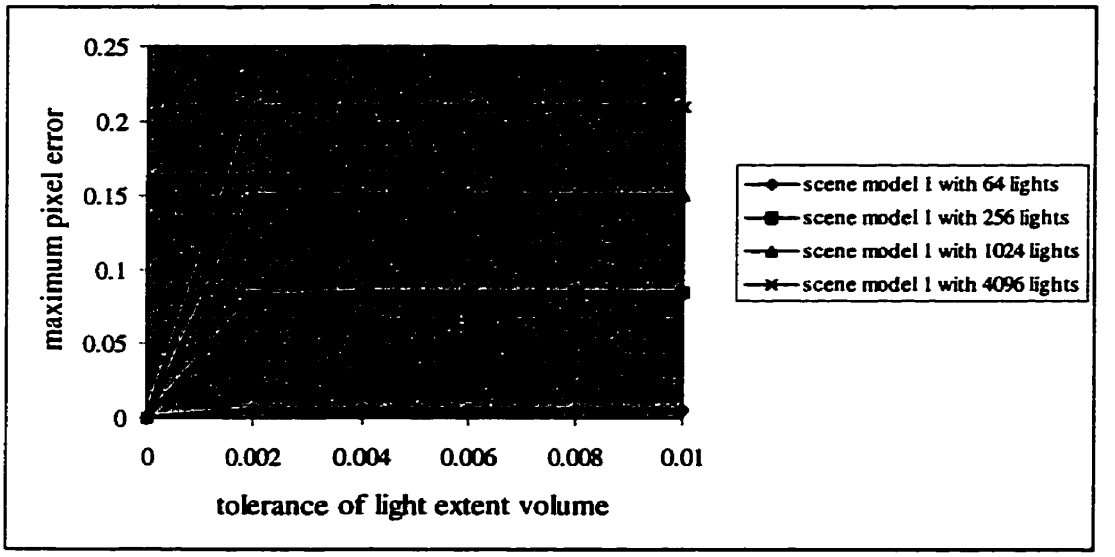
Figure 5.10 (a) and (b) show the average and maximum pixel errors of the output images with increasing light sources at different tolerance of the light extent volume. In this test, the actual average pixel error is always kept within the requested tolerance, but the maximum pixel error is a little bit higher than the threshold. Since the overall pixel error of an output image is far smaller than the tolerance, the quality of the image is guaranteed by our LEV method.



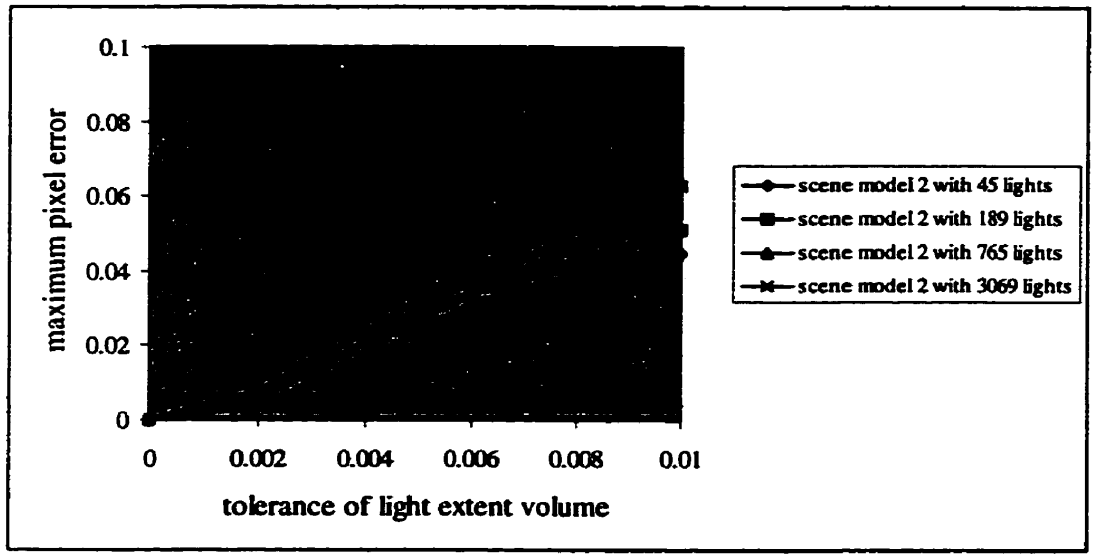
(a) average pixel error based on Scene Model 1



(b) average pixel error based on Scene Model 2



(c) maximum pixel error based on Scene Model 1



(d) maximum pixel error based on Scene Model 2

Figure 5.10 average and maximum pixel error when increasing light sources

Chapter 6

Analysis and Conclusion

6.1 Overview

In this chapter analysis is provided for the observations drawn from the test results. At the very end of this chapter the conclusion of this thesis is presented.

6.2 Observations

From the experimental results several observations can be made. In the next section analysis is provided to explain these observations.

1. The LEV method works well for scenes with many light sources where every object is illuminated by relatively few of the candidate light sources, while as expected, it does not work well for scenes with many light sources where every object is shone by most of them.
2. Compared with the traditional RT and Ward's AST method, only the LEV method has approximately logarithmic performance for scenes where every object is shone by relatively few of the candidate light sources. This computation

complexity allows hundreds or even thousands of light sources in a scene to be rendered in reasonable time.

3. Compared with the traditional RT, the LEV method achieves significant speedups (up to 150 times faster in the case of 12K light sources in the scene). And for a high number of light sources, our new approach consistently faster than Ward's AST approach.
4. The performance of the LEV method improves as the number of light sources increases in the scene.
5. Compared with the AST approach, by building the light hierarchy properly the LEV method requires minimal memory overhead for shadow testing acceleration.

6.3 Analysis

The LEV method has great performance for scenes with many light sources where every object is directly shone by a relatively few of the candidate light sources in the scene. This is typical for virtually all scenes in Computer Graphics. But to scenes where every intersection point on objects is shone by most of the light sources it's a different story because to each intersection point most of the candidate light sources are important and make great contribution to that point.

In the following we examine two key parameters which directly provide the explanation to observations based on Scene Model 1 and Scene Model 2. The parameters are gotten by applying the algorithm to scene models with increasing light sources at the same tolerance of light extent volume.

- **Parameter 1:** average number of important light sources tested for visibility per intersection point

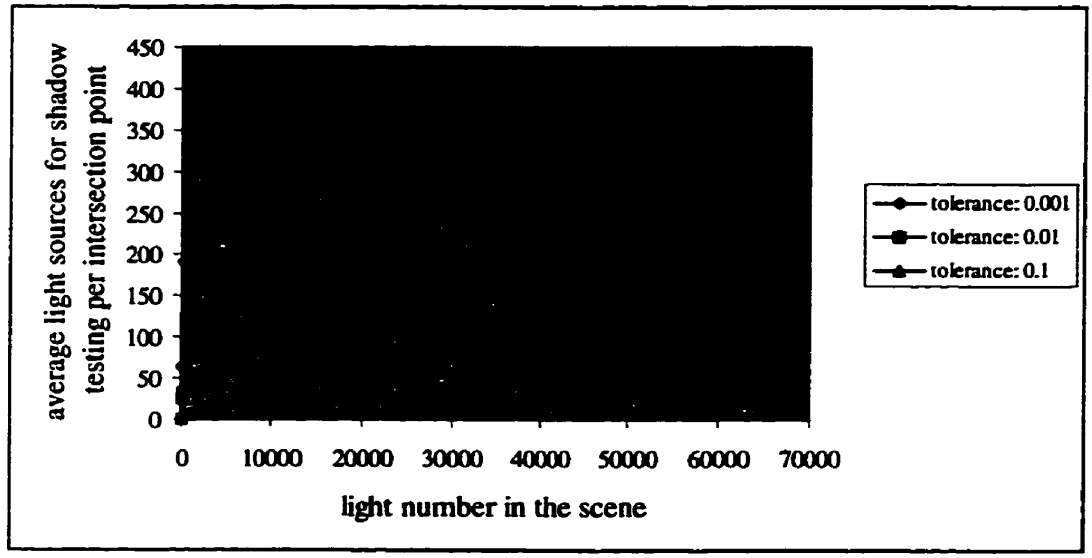
  As we mentioned before, a light source is important to an intersection point only if the intersection point is inside the light source's extent volume. Once the tolerance of light extent volume is set, the radius of lights' extent volumes is fixed. So when scene models have certain light sources, the average number of important light sources tested for visibility per intersection point soon becomes consistent, as the results from Scene Model 1 and Scene Model 2 shown in Figure 6.1 (a) and Figure 6.2 (a) respectively.

- **Parameter 2:** average number of light extent volumes checked per intersection point when traversing the light hierarchy tree to find out the important light sources

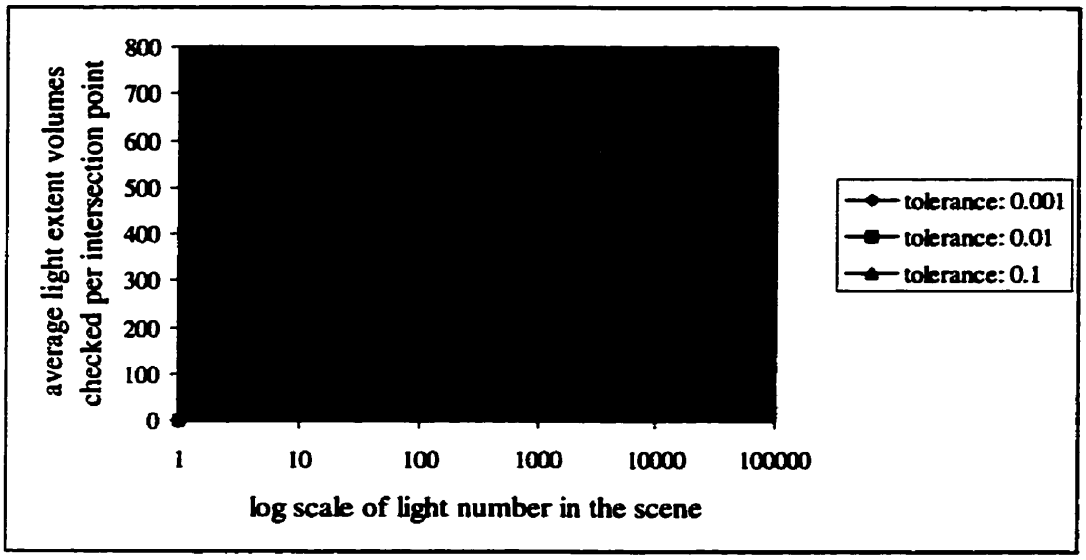
  Since at the same tolerance after scene models have certain light sources, the average number of important light sources for shadow testing at each intersection point approaches to a constant value, the algorithm's performance complexity is then determined by the efficiency of traversing the light hierarchy to find out important light sources. The efficiency can be evaluated by the average number of light extent volumes checked per intersection point during the traversal.

  After considering two issues: 1) light hierarchy trees grow with the increase of light sources in scene models; 2) after scene models have certain light sources, the average number of light sources tested per intersection point approaches to the same, we can say that in the search of important light sources the average number

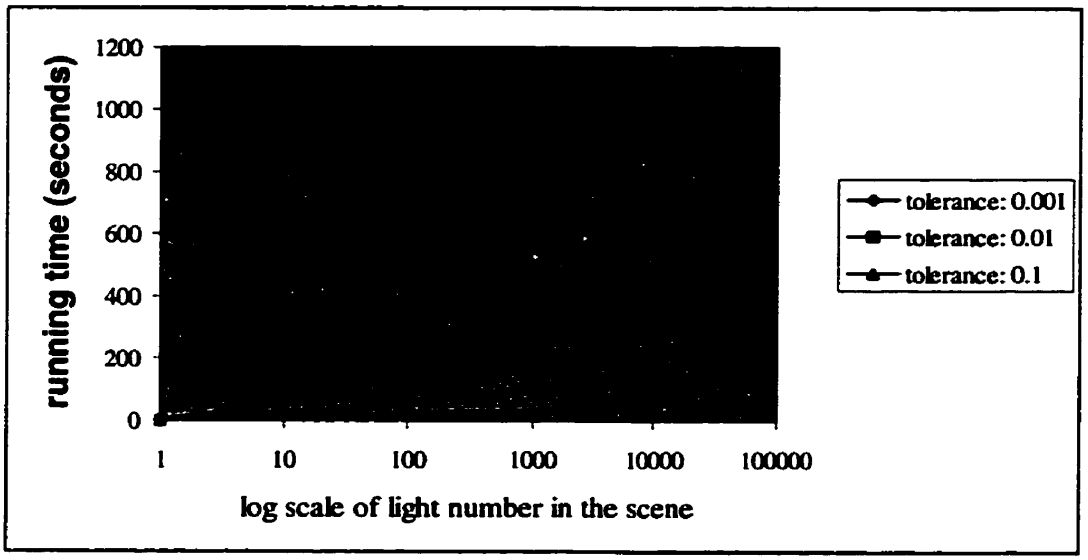
of light extent volumes checked per intersection point increase only a little bit, and it is far smaller than the increase of the light number in the scene. Actually the increase curve has approximately logarithmic behavior, as shown in Figure 6.1 (b) and Figure 6.2 (b). So the computational complexity of the Light Extent Volumes method is approximately logarithmic, as shown in Figure 6.1(c) and Figure 6.2(c). Thus the performance of the Light Extent Volumes method improves as the number of light sources increases in the scene. At high number of light sources, its performance advantage is even more obvious.



(a) average number of light sources tested for visibility at each intersection point based on Scene Model 1

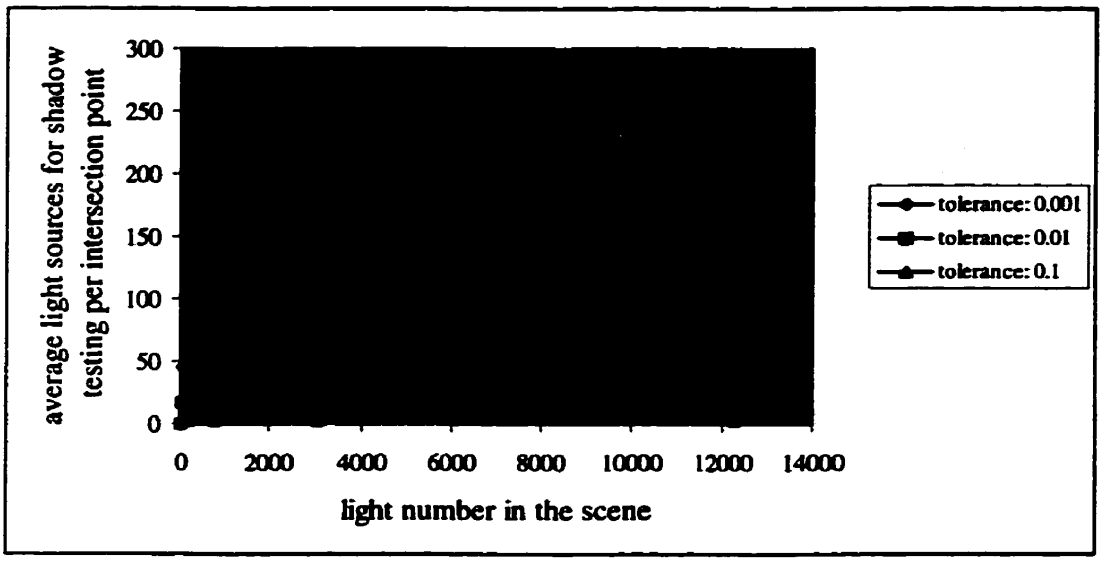


(b) average number of light extent volumes checked at each intersection point based on Scene Model 1

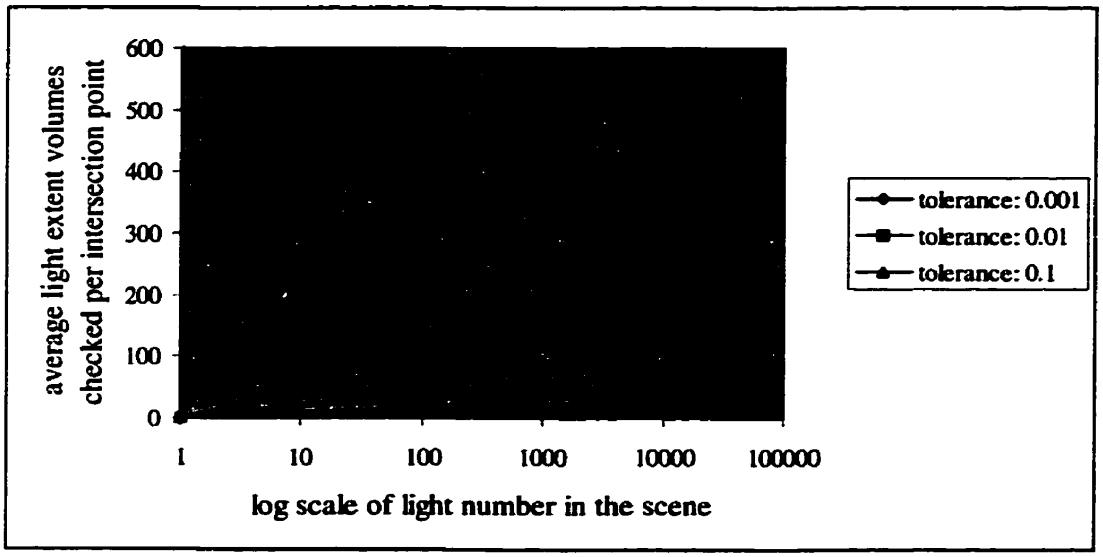


(c) performance of the LEV method under different tolerances based on Scene Model 1

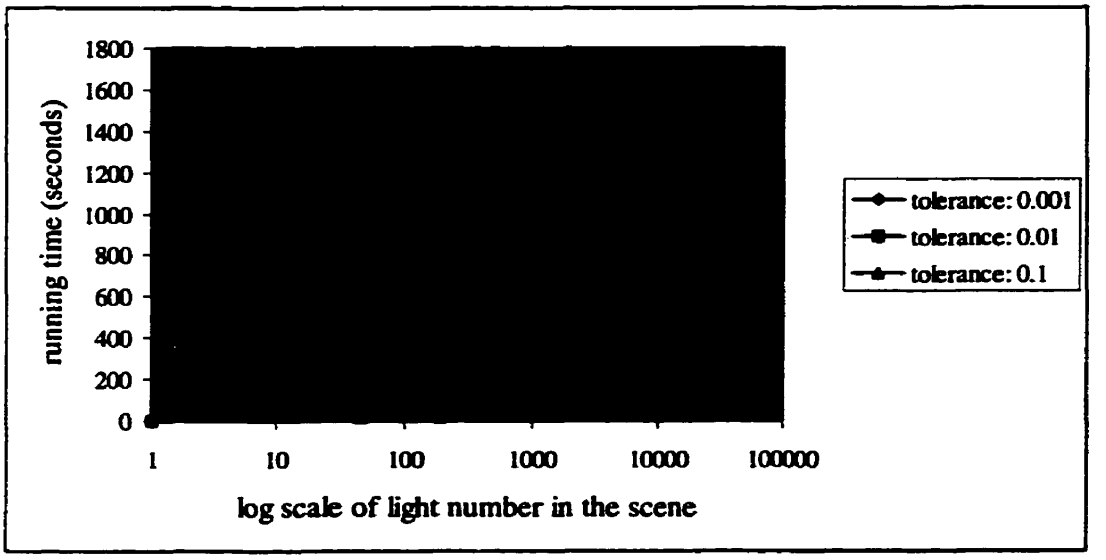
Figure 6.1 two key parameters of the Light Extent Volumes (LEV) method based on Scene Model 1



(a) average number of light sources tested for visibility at each intersection point based on Scene Model 2



(b) average number of light extent volumes checked at each intersection point based on Scene Model 2



(c) performance of the LEV method under different tolerances based on Scene Model 2

Figure 6.2 key parameters of the Light Extent Volumes (LEV) method based on Scene Model 2

6.4 Conclusion

This thesis put forward a new approach to efficiently ray tracing scenes with many light sources. By building a hierarchical tree of light extent volumes one can approach approximately *O(log n)* complexity in determining which light sources contribute significant irradiance to the intersection point, where n is the number of light sources. This allows hundreds or even tens of thousands of light sources in a scene to be rendered in reasonable time.

The Light Extent Volumes method is specially designed for scenes with many light sources where every intersection point is illuminated by few of the candidate

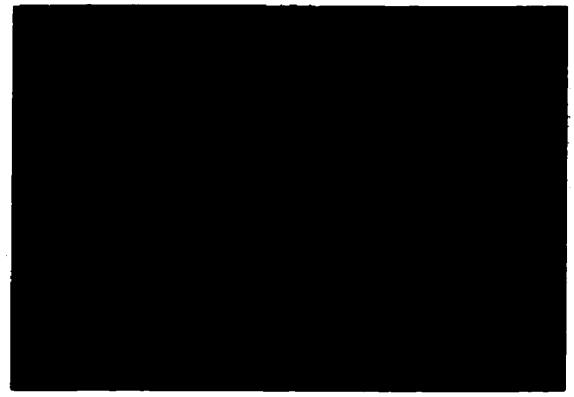
light sources. When rendering those scene models, the algorithm has approximately logarithmic complexity. And the performance of the LEV method improves as the number of light sources increases in the scene. It achieves significant speedup over other approaches (up to 150 times faster), such as the traditional ray tracing and Ward's Adaptive Shadow Testing. Meanwhile the tradeoff between image accuracy and rendering speed is negligible. Moreover, combined with instancing, the algorithm requires minimal memory overhead for shadow testing acceleration. Another important feature of the LEV algorithm is its simplicity of implementation. In addition the approach is orthogonal to most other global illumination techniques, and can be added to existing direct light calculations and optimizations. In conclusion the LEV approach is a practical algorithm for efficiently ray tracing scenes with many light sources.

6.5 Future Work

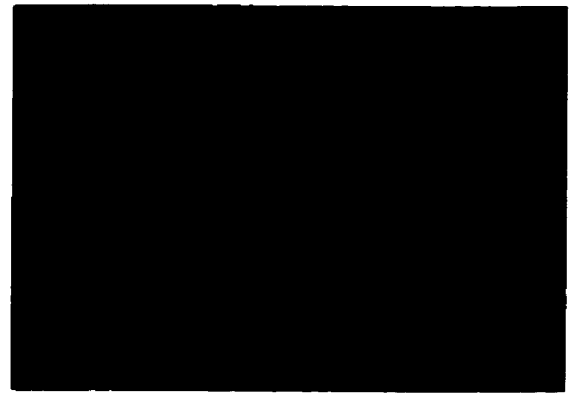
The introduction of the LEV algorithm provides a promising direction for efficient ray tracing of many light sources. Currently the algorithm has great performance for typical scenes. In the near future, we can do further development to complete this method to deal with several special situations. First, we will extend the implementation to linear and area light sources. A possible solution is to combine our LEV method with Ward's AST approach for rendering. The AST method minimizes the number of light sources that must be tested for visibility by sorting the contributions of all light sources. However, the cost of sorting can make the approach impractical for the rendering. So first we can use our LEV algorithm to quickly

identify all the significant light sources for shadow testing with appropriate light extent volumes. And then, we combine the AST method to do the direct contribution calculation on those significant sources and estimate the contributions of the rest light sources. Second, we may explore the impact of specularity on the LEV method. For example, stars have negligible irradiance, but we can see them in the reflections. The current algorithm would treat them as unimportant light sources and ignore them.

Appendix A



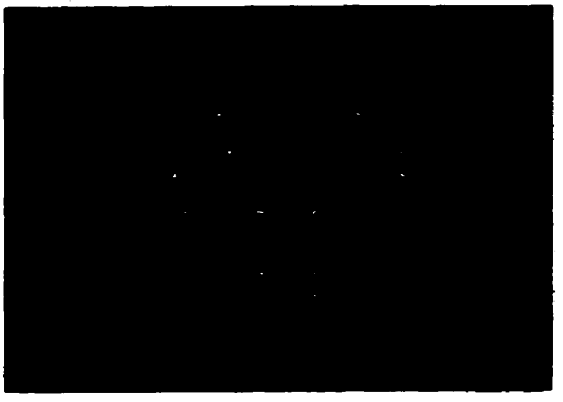
The building model with 256 light sources



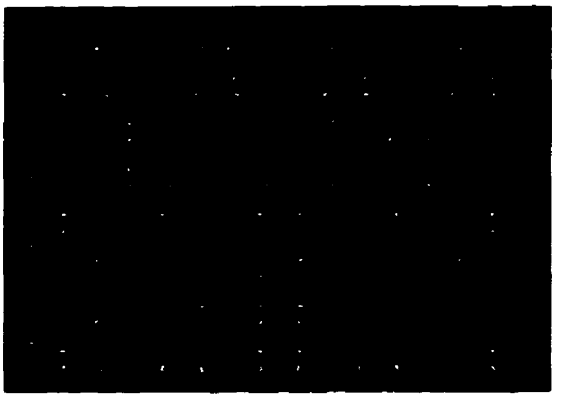
The building model with 1,024 light sources

The building model with 4,096 light sources

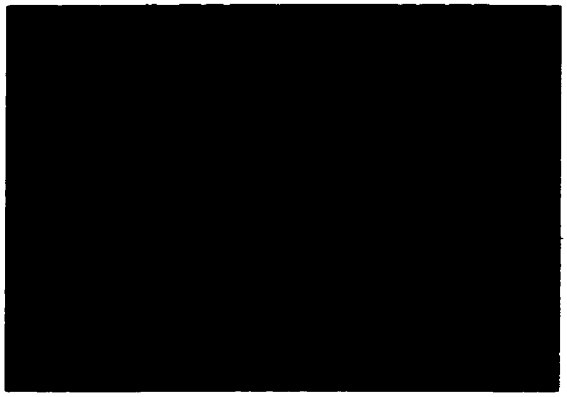
The building model with 16,384 light sources



The peano-curve maze with 189 light sources



The peano-curve maze with 765 light sources



The peano-curve maze with 3,069 light sources

The peano-curve maze with 12,285 light sources

Appendix B

Data collected from result tables in Chapter 5 and Chapter 6.

LIGHTS IN THE SCENE	RT	AST	LEV	AST/RT	LEV/RT	LEV/RT SPEED UP
64 lights	98.86s	42.12s	46.38s	42.61%	46.91%	2.13
256 lights	373.77s	224.48s	78.96s	60.06%	21.13%	4.73
1,024 lights	1268.58s	812.23s	101.29s	64.03%	7.98%	12.52
4,096 lights	5807.60s	3380.42s	134.56s	58.21%	2.32%	43.16
16,384 lights	24289.28s	13665.89s	184.14s	56.26%	0.76%	131.91

Table 1 -Test 1.1 result table based on Scene Model 1 for performance comparison of the traditional ray tracing (RT), Adaptive Shadow Testing (AST) and Light Extent Volumes (LEV)

LIGHTS IN THE SCENE	RT	AST	LEV	AST/RT	LEV/RT	LEV/RT SPEED UP
45 lights	99.58s	39.66s	59.65s	39.83%	59.90%	1.67
189 lights	248.09s	132.53s	62.76s	53.42%	25.30%	3.95
765 lights	2569.58s	1359.05s	218.19s	52.89%	8.49%	11.78
3,069 lights	11989.90s	6155.61s	287.73s	51.34%	2.40%	41.47
12,285lights	46025.75s	22953.04s	296.19s	49.87%	0.64%	155.39

Table 2 -Test 1.1 result table based on Scene Model 2 for performance comparison of the traditional ray tracing (RT), Adpative Shadow Testing (AST) and Light Extent Volumes (LEV)

Lights in the scene	extra memory space usage based on Scene Model1 (bytes)		
	AST	LEV without instancing	LEV with instancing
64 lights	1,536	8,704	2,932
256 lights	6,144	34,816	3,812
1,024 lights	24,576	139,264	4,692
4,096 lights	98,304	557,056	5,572
16,384 lights	393,216	2,228,224	6,452

Table 3 - Test 1.2 comparison of extra memory space used by Adaptive Shadow Testing and Light Extent Volumes for shadow testing acceleration based on Scene Model1. Note that the data for the case of LEV without instancing is got by fully using light extent volumes in the light hierarchy

lights in the scene	Running time in seconds		
	tolerance = 0.001	tolerance = 0.01	tolerance = 0.1
64 lights	97.97s	46.38s	15.73s
256 lights	321.29s	78.96s	23.97s
1,024 lights	490.75s	101.29s	29.20s
4,096 lights	746.31s	134.56s	38.52s
16,384 lights	1081.29s	184.14s	56.99s
65,536 lights	1117.16s	191.14s	57.80s

Table 4 - Test 2.1 logarithmic behavior of the Light Extent Volumes method based on Scene Model 1

# lights in the scene	Running time in seconds		
	tolerance = 0.001	tolerance = 0.01	tolerance = 0.1
45 lights	99.12s	57.88s	12.27s
189 lights	198.42s	62.76s	13.23s
765 lights	908.95s	218.19s	41.63s
3,069 lights	1347.80s	287.73s	52.76s
12,285 lights	1529.20s	296.19s	58.11s

Table 5 - Test 2.1 logarithmic behavior of the Light Extent Volumes method based on Scene Model 2

Tolerance of light extent volume	Average light sources tested per intersection point			Fraction of light sources tested per intersection point		
	Model 1	Model 2	Model 3	Model 1	Model 2	Model 3
0	1024	3069	1024	100%	100%	100%
0.0001	1024	3069	753	100%	100%	73.50%
0.002	159	119	753	15.50%	3.86%	73.50%
0.004	88	56	630	8.62%	1.82%	61.53%
0.006	61	37	455	5.94%	1.20%	44.47%
0.008	47	25	45	4.56%	0.81%	4.44%
0.01	38	19	31	3.67%	0.63%	3.05%

Table 6 – Test2.2 fraction of light sources tested per intersection point

Tolerance of light extent volume	average light sources tested per intersection point				fraction of light sources tested per intersection point			
	256 lights	1024 lights	4096 lights	16384 lights	256 lights	1024 lights	4096 lights	16384 lights
0.00001	188	753	3010	12042	73.50%	73.50%	73.50%	73.50%
0.001	188	753	2532	229	73.50%	73.50%	61.83%	1.40%
0.002	188	753	180	83	73.50%	73.50%	4.39%	0.51%
0.004	188	630	57	30	73.50%	61.53%	1.40%	0.18%
0.006	188	455	32	16	73.50%	44.47%	0.78%	0.10%
0.008	188	45	21	11	73.50%	4.44%	0.51%	0.07%
0.01	188	31	15	8	73.50%	3.05%	0.36%	0.05%

Table 7 – Test2.2 fraction of light sources tested per intersection point based on Scene Model 3

tolerance of light extent volume	average light sources tested per intersection point					fraction of light sources tested per intersection point				
	64 lights	256 lights	1024 lights	4096 lights	16384 lights	64 lights	256 lights	1024 lights	4096 lights	16384 lights
0	64	256	1024	4096	16384	100%	100%	100%	100%	100%
0.002	62	117	159	186	201	96.56%	45.81%	15.50%	4.55%	1.23%
0.004	46	70	88	99	105	71.58%	27.51%	8.62%	2.42%	0.64%
0.006	35	50	61	67	70	54.68%	19.67%	5.94%	1.63%	0.43%
0.008	29	40	47	51	53	44.74%	15.54%	4.56%	1.24%	0.33%
0.01	24	32	38	41	42	37.55%	12.59%	3.67%	0.99%	0.26%

Table 8 - Test 2.3 fraction of light sources tested per intersection point with more light sources on Scene Model 1

Tolerance of light extent volume	average light sources tested per intersection point					fraction of light sources tested per intersection point				
	45 lights	189 lights	765 lights	3069 lights	12285 lights	45 lights	189 lights	765 lights	3069 lights	12285 lights
0	45	189	765	3069	12285	100%	100%	100%	100%	100%
0.002	42	81	106	119	124	93.87%	43.03%	13.87%	3.86%	1.01%
0.004	33	42	52	56	58	72.59%	22.31%	6.77%	1.82%	0.47%
0.006	24	29	35	37	37	53.46%	15.29%	4.54%	1.20%	0.30%
0.008	20	21	24	25	25	44.83%	10.95%	3.10%	0.81%	0.20%
0.01	17	17	19	19	19	36.94%	8.81%	2.46%	0.63%	0.16%

Table 9 - Test 2.3 fraction of light sources tested per intersection point with more light sources on Scene Model 2

tolerance of light extent volume	average pixel error (compared to (255, 255, 255) RGB)				maximum pixel error (compared to (255, 255, 255) RGB)			
	64 lights	256 lights	1024 lights	4096 lights	64 lights	256 lights	1024 lights	4096 lights
0.002	0%	0.0248%	0.0883%	0.1664%	0.5229%	8.1046%	15.0327%	20.92%
0.004	0%	0.0258%	0.0923%	0.1750%	0.5229%	8.4967%	15.0327%	20.92%
0.006	0%	0.0258%	0.0923%	0.1750%	0.5229%	8.4967%	15.0327%	20.92%
0.008	0%	0.0258%	0.0923%	0.1750%	0.5229%	8.4967%	15.0327%	20.92%
0.01	0%	0.0258%	0.0923%	0.1750%	0.5229%	8.4967%	15.0327%	20.92%

Table 10 – Test2.4 average and maximum pixel errors by increasing light sources in Scene Model 1

tolerance of light extent volume	average pixel error (compared to 255, 255, 255 RGB)				maximum pixel error (compared to 255, 255, 255 RGB)			
	45 lights	189 lights	765 lights	3069 lights	45 lights	189 lights	765 lights	3069 lights
0.002	0.0005%	0.0007%	0.0035%	0.0039%	0.3922%	0.5229%	0.7843%	0.6536%
0.004	0.0045%	0.0102%	0.0384%	0.0394%	1.5686%	2.0915%	2.3529%	2.3529%
0.006	0.0272%	0.0149%	0.0565%	0.0584%	3.1373%	3.3529%	3.4295%	4.0830%
0.008	0.0339%	0.0243%	0.0947%	0.1046%	3.9216%	4.3137%	4.7059%	5.2131%
0.01	0.0427%	0.0403%	0.1544%	0.1724%	4.4575%	5.0980%	6.3660%	6.2754%

Table 11 – Test2.4 average and maximum pixel errors by increasing light sources in Scene Model 2

lights in the scene	average light sources for shadow testing per intersection point based on Scene Model 1		
	tolerance = 0.001	tolerance = 0.0	Tolerance = 0.1
64 lights	64	25	4
256 lights	191	33	5
1024 lights	289	38	5
4096 lights	361	41	5
16384 lights	405	43	5
65536 lights	427	44	5

Table 12 average number of light sources tested for visibility at each intersection point based on Scene Model 1

lights in the scene	average light extent volumes checked per intersection point based on Scene Model 1		
	tolerance = 0.001	tolerance = 0.01	Tolerance = 0.1
64 lights	86	46	19
256 lights	279	69	25
1024 lights	443	88	31
4096 lights	569	101	36
16384 lights	651	110	42
65536 lights	696	118	45

Table 13 average number of light extent volumes checked at each intersection point based on Scene Model 1

lights in the scene	average light sources for shadow testing per intersection point based on Scene Model 2		
	tolerance = 0.00	Tolerance = 0.01	Tolerance = 0.1
45 lights	45	17	2
189 lights	132	17	2
765 lights	195	19	2
3069 lights	234	19	2
12285lights	254	19	2

Table 14 average number of light sources tested for visibility at each intersection point based on Scene Model 2

lights in the scene	average light extent volumes checked per intersection point based on Scene Model 2		
	tolerance = 0.001	tolerance = 0.01	tolerance = 0.1
45 lights	66	34	12
189 lights	214	43	14
765 lights	344	56	16
3069 lights	432	65	19
12285lights	487	71	21

Table 15 average number of light extent volumes checked at each intersection point based on Scene Model 2

Bibliography

AMAN87 Amanatides, J., Woo, A., "A Fast Voxel Traversal Algorithm for Ray Tracing", *Eurographics'87, Proceedings of the European Computer Graphics Conference and Exhibition*, Amsterdam, 1987, pp. 3 – 10.

APPE68 Appel, A., "Some Techniques for Shading Machine Renderings of Solids", *Proceedings of the Spring Joint Computer Conference*, 1968, pp. 37-45.

BERG86 Bergeron, P., "A General Version of Crow's Shadow Volumes", *IEEE Computer Graphics and Applications*, Vol. 6, No. 9, 1986, pp. 17 – 28.

EOKS89 Eo, K.S., Kyung, C.M., "Hybrid shadow testing scheme for ray tracing", *Computer Aided Design, Vol. 21, No. 1*, Jan/Feb 1989, pp. 38-48.

FUJI86 Fujimoto, A., Tanaka, T., and Iwata, K., "ARTS: Accelerated Ray Tracing System", *IEEE Computer Graphics and Applications, Vol. 6, No. 4*, April 1986, pp. 16 – 26.

GLAS84 Glassner, A. S., "Space Subdivision for Fast Ray Tracing", *IEEE Computer Graphics and Applications, Vol. 4, No. 10,* October 1984, pp. 15-22.

HALT70 Halton, J. H., "A Retrospective and Prospective of the Monte Carlo Method", *SIAM Rev. 12*, January, 1970, pp. 1 – 63.

HAMM64 Hammersley, J. M., Handscomb, D. C., "Monte Carlo Methods", Wiley, New York, 1964

HOUL93 Houle, C., Fiume, E., "Light-source modeling using pyramidal light maps", *Graphical Models and Image Processing, Vol. 55, No.* 5, 1993, pp. 346 – 358.

JANS86 Jansen, F. W., "Data Structures for Ray Tracing", *Data Structures for Raster Graphics, Proceedings Workshop, Eurographics Seminars,* Springer Verlag, 1986, pp. 57 – 73.

KAPL85 Kaplan, M. R., "Space Tracing a Constant Time Ray Tracer", *State of the Art in Image Synthesis (Siggraph'85 Course Notes), Vol.* 11, July 1985.

KAY86 Kay, T.L., Kajiya, J.T., "Ray Tracing Complex Scenes", *Computer Graphics, Vol. 20, No. 4*, August 1986, pp. 269 – 278

KAY79 Kay, D.S., "Transparency, Refraction and Ray Tracing for Computer Synthesized Images", M. S. Thesis, Program of Computer Graphics, Cornell University, Ithaca, NY, January 1979.

KOK91 Kok, A. J. F., Jansen, F. W., "Adaptive Sampling of Area Light Sources in Ray Tracing including diffuse Interreflection", *Eurographics'92, Forum 11, 3*, pp. 289 – 298

PAQU98 Paquette, Eric, et al., "A Light Hierarchy for Fast Rendering of Scenes with Many Lights", *Proceedings of Eurographics'98, Vol. 17, No. 3,* 1998.

PHON75 Phong, B.T., "Illumination for Computer Generated Pictures", *Communications of the ACM, Vol. 18, No.* 6, June 1975, pp. 311 – 317.

RUBI80 Rubin, S. M., Whitted, T., "A 3-Dimensional Representation for Fast Rendering of Complex Scenes", *Computer Graphics, Vol. 14, No. 3*, July 1980, pp. 110-116.

SHIR96 Shirley Peter, Changyaw Wang and Kurt Zimmerman, "Monte Carlo Techniques for Direct Lighting Calculations", *ACM Transactions on Graphics, Vol. 15, No. 1*, January 1996, pp. 1-36.

SHRE66 Shreider, Y. A., "The Monte Carlo Method", Pergamon Press, New York, 1966

SILL94 Sillion, F., "Clustering and Volume Scattering for Hierarchical Radiosity Calculations", *Fifth Eurographics Workshop on Rendering*, Darmstadt, Germany, June 1994, pp. 105 – 117.

SMIT94 Smits, B., Arvo, J., and Greenberg, D., "A Clustering Algorithm for Radiosity in Complex Environments", *Proceedings of SIGGRAPH'94, Computer Graphics*, July 1994, pp. 435 – 442.

STAM95 Stam, J., Fiume, E., "Depicting Fire and Other Gaseous Phenomena Using Diffusion Process", *proceedings of SIGGRAPH'95*, August, 1995, pp. 129 – 136.

STUE94 Stuerzlinger, W., Tobler, R., " Two Optimization Methods for Ray Tracing", Proceedings Summer School of Computer Graphics'94, June, 1994, pp. 104-107.

WARD91 Ward, G. J., "Adaptive Shadow Testing for Ray Tracing", *Eurographics Rendering Workshop, Barcelona, Spain*, May 1991.

WARN83 Warn, D. R., "Lighting Controls for Synthetic Images", *Computer Graphics, Vol. 17, No. 3*, July 1983, pp. 49-54.

WEGH84 Weghorst, H., Hooper, G., and Greenberg, D. P., "Improved Computational Methods for Ray Tracing", *ACM Trans. on Graphics, Vol. 3, No. 1*, January 1984, pp. 52 – 69.

WHIT80 Whitted, T., "An Improved Illumination Model for Shaded Display", *Communications of the ACM, Vol. 23, No. 6*, June 1980, pp. 343-349.

WOO90a Woo, A., Amanatides, J., "Voxel Occlusion Testing: A Shadow Determination Accelerator for Ray Tracing", *Graphics Interface '90*, May 1990, pp. 213-220.

WOO90b Woo, A., Poulin P., and Fournier, A., "A Survey of Shadow Algorithms", *IEEE Computer Graphics and Applications, Vo.10, No. 6*, 1990, pp. 13 – 32

YAKO77 Yakowitz, S. J., "Computational Probability and Simulation", Addison-Wesley, New York, 1977