TIME-CRITICAL MODELING AND RENDERING: GEOMETRY-BASED AND IMAGE-BASED APPROACHES

by

Tien-Tsin Wong

黃田津

A dissertation submitted to the faculty of the Chinese University of Hong Kong in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science and Engineering.

1998

Accepted by the Department of Computer Science and Engineering, The Chinese University of Hong Kong, in fulfillment of the requirements of the degree of Doctor of Philosophy.

Prof. Pheng-Ann Heng
(Principal Adviser)

Prof. Wai-Yin Ng
(Principal Adviser)

Prof. Qunsheng Peng

Prof. Hanqiu Sun

Prof. Kin-chuen Hui

To my parents,

Wong, Kun-Ming and Liu, Siu-Fen.

黃昆明　　劉素芬

Abstract

In this thesis, we describe two approaches, geometry-based and image-based, to address the problem of time-critical modeling and rendering. Time-critical modeling and rendering aims to improve the rendering speed using both modeling and rendering techniques. Improving the rendering time enables interactive display of large scale complex scene. This is important in many applications such as virtual reality, medical visualization, flight simulation, etc.

We first describe a new geometry-based simplification algorithm, *adaptive skeleton climbing*, which generates simplified mesh directly from volume data. By partitioning the volume into variable-sized rectangular boxes, the algorithm generates triangles to approximate the enclosed isosurface adaptively (larger triangles approximate smooth regions and vice-versa). Since we apply binary tree organization on each dimension of the rectangular boxes, it allows more flexibility in forming rectangular boxes which are not allowed in previous octree approaches. Therefore a coarser mesh can be generated using the proposed algorithm.

Although generating simplified meshes can significantly reduce the rendering time, the rendering speed is still dependent on the complexity of the scene. Note that the scene can be arbitrarily complex. If the only goal of the graphics system is to provide the visual experience, we can model and render the desired image using previously recorded images (reference images). This leads to our development of image-based computer graphics. For pure image-based rendering, the rendering time will now only depend on the resolution of the images.

Since the illumination of the scene is fixed during image capture, the illumination in the synthesized images is also fixed. We propose a novel concept of measuring apparent BRDFs of image plane pixels to overcome the unchangeable illumination problem in previous image-based approaches. By treating the image plane pixel as an ordinary surface element and measuring its apparent BRDF from the reference images, we can record the *pixel BRDF*. Using this apparent reflectance information, we are able to re-render the image-based scene/object under any desired illumination condition.

The idea is verified by applying it to various image-based data structures, light field, Lumigraph and panorama. We also devise a *practical compression scheme* to handle the huge amount of data of pixel BRDFs.

摘要

在這論文裏，我們探討了基於幾何和基於圖像的限時造形和繪制問題，目的是要加速繪制，從而可以實現複雜場景的實時顯示。這項研究的結果可以應用在虛擬現實，醫學數據可視化，飛行模擬等領域。

首先，我們會提出一個新的基於幾何的簡化算法，稱爲 adaptive skeleton climbing。它能直接地從三維體數據生成簡化的等值面。它可以根據等值面的平滑度來把網格劃分成大小不一的矩形盒子，然後生成簡化的三角片（即是平的地方用大的三角片來表示，而細緻的地方則用小的三角片來表示）。由於我們採用了二叉樹結構，因此可以比較靈活地把網格劃分成大小不一的矩形盒子，而不需要像以往利用八叉樹結構時必須劃分成立方體。因此我們提出的方法可以生成較簡化的等值面。

雖然簡化了的等值面可以大大地減少繪制所需時間，但是繪制時間仍然取決於場景的複雜度，而場景是可以無限制地複雜。如果圖形系統是爲了提供視覺效果，我們可以已有的圖像來合成新的圖像，這便是基於圖像的計算機圖形系統。一個純粹的基於圖像的繪制系統，其繪制所需的時間應該取決於圖像的解析度。

由於拍攝時燈光是固定的，因此合成出來的新圖像裏的燈光也是固定的。我們提出一個新的概念來解決這問題。這概念把像素看做正常的面元來量度其 BRDF。因爲它不是眞的 BRDF，我們稱它爲像素 BRDF。應用它，我們可以實時地改變和控制合成圖像裏場景的燈光。

這概念已推廣應用在多個基於圖像的數據結構，例如 light field， lumigraph，及全景圖。爲減少所需的數據存量，我們採用了一些實用的數據壓縮方法。

Acknowledgements

I wish to thank my advisors, Pheng-Ann Heng and Wai-Yin Ng, for their guidance and support throughout my graduate studies. Their advice and infallible judgment have been a tremendous influence and an invaluable resource for me. I am fortunate to have the chance to work with them during my time at the Chinese University of Hong Kong. Without them, I may not be able to finish the hard road in pursuing a PhD in Hong Kong. I would also like to thank my thesis committee, Qunsheng Peng, Hanqiu Sun and Kin-chuen Hui for their comments and insights on my work.

I am especially grateful to Tim Poston of CieMed of National University of Singapore for his endless suggestions and advice during our collaboration in the development of the simplification algorithm, *adaptive skeleton climbing*. I also owe tremendous debts of gratitude to many department colleagues, in particular, Siu-Hang Or, Wai-Shing Luk and Chi-Shing Leung for their collaborations in the image-based rendering framework.

I have also benefited from interactions with many people during my graduate studies. I would like to thank Chong-Yuen Li, Chi-Lok Chan, Chong-Kan Chiu, Sau-Yuen Wong, Chor-Tung Yau, Wing-Kay Yip, Sau-Ming Lau, Yui-Wah Lee, Man-Leung Wong and Chi-Hong Leung.

Most of all, I am deeply grateful to my parents, Kun-Ming Wong and Siu-Fen Liu, for their unlimited tolerance and patient for allowing their only son to finish the risky study. Without their support and encouragement, I cannot finish my graduate studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since the introduction of Sketchpad [SUTH63] in the early sixty, technologies in modern interactive computer graphics has been much improved. We now can render millions of antialiased, texture-mapped polygons within a second using state-of-art graphics hardware [AKEL93, MONT97]. Radiosity development [GORA84, NISH85, COHE93, SILL94] further allows us to render realistic images that are nearly indistinguishable from real photographs. However, rendering realistic image of arbitrarily complex scene in real time is still far from satisfactory.

The ultimate goal of *time-critical modeling and rendering* research is to allow rendering realistic images in real time. Rendering algorithm alone is not enough to solve the problem, a suitable modeling algorithm is also needed. It is a research problem of both modeling and rendering.

Time-critical modeling and rendering has been studied for a long period of time. Newell and Blinn [NEWE77] have already pointed out the following problem in their 1977 review.

> *There is a huge disparity between the complexity of scenes in the real world and synthetic scenes.*

Ten years later, Heckbert [HECK87] further confirmed that even with faster computers and larger memories, real time rendering of complex scene still cannot be achieved.

> *Since 1977 we have made considerable progress in complexity due to faster computers, larger memories, improvements in algorithm efficiency, and increased use of procedural modeling, but we cannot yet regard the complexity problem as solved. We have found that handling complexity is as much of a modeling and logistics problem as it is a rendering problem.*

Most common rendering algorithms used nowadays are dependent of the scene complexity. One practical rendering algorithm is the depth-buffering, which has a time complexity of $O(n)$, where n is the number of primitives in the scene. Unfortunately, real world scene can be arbitrarily complex. Imagine a scene of a forest, each tree is modeled with thousands of polygons. Even with the fastest graphics engine, it is not possible to realistically render such a scene in real time.

Although the scene can be infinitely complex, the human perception is limited. The limitations can be subdivided into two classes, the internal and external limitations. The internal limitation is due

to the number, sensitivity and distribution of the rod and cone sensors on the human retina [GOLD89]. On the other hand, external limitation is due to the physical laws of light propagation. For example, object occluded by another opaque object cannot be seen from human eye since the light ray propagates along a straight line.

By utilizing the various types of limitations, the scene can be rendered in a shorter period of time. Hence, the research of time-critical modeling and rendering is basically *a research of how to make use of limitations of human perception*.

1.1 Geometry-based and Image-based Approaches

There are now two basic approaches to computer graphics, namely the *geometry-based* and the *image-based* approaches. Geometry-based computer graphics is the approach used by most of the computer graphics systems nowadays. Figure 1-1 shows the geometry-based approach graphically. It is a physical simulation of light propagation in the space. The geometry models (including the geometry representation and the surface properties) and the physical laws are input to the simulation process. The final result (desired image) is a synthetic image simulating the visual appearance of the scene when physically viewed by the human eye.

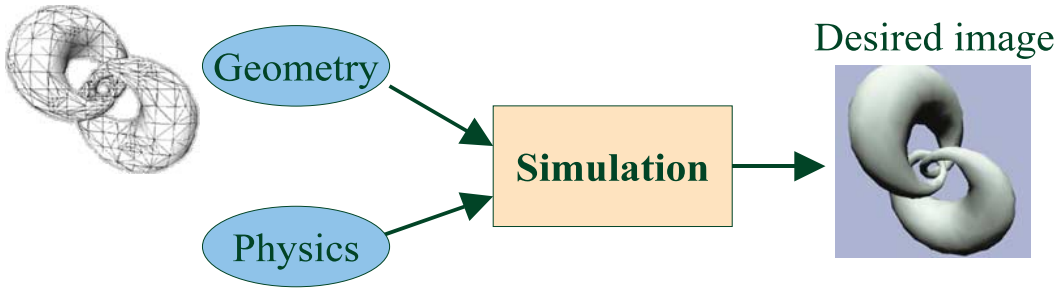


Figure 1-1: Geometry-based computer graphics.

Geometry-based time-critical modeling and rendering can be done on the two inputs of the physical simulation and the rendering algorithm (the simulation process) itself. Using the simplified equations to approximate the physical laws, the simulation can be done more efficiently. The Phong's illumination model [PHON75] is a good example of approximating physical reflection by simplified formula. Even the infamous physical-based radiosity model [COHE93, SILL94] involves certain amount of simplification. However, simplifying the physical laws alone cannot achieve the goal.

Hence, simplification has also been applied on the geometry models. Various aspects of simplification of geometry models will be described in Chapter 2. Another direction is to design a renderer that can efficiently render large scale models using hierarchical rendering approach [GREE93, GREE96]. In this thesis, we will concentrate on simplification of models instead.

A completely different approach to computer graphics is the image-based approach. Figure 1-2 illustrates the approach. Instead of going through a physical simulation, the desired images are synthesized by means of warping and composition. A set of *reference images* and minimal geometry information are input to the system. The final *desired image* is done by warping and compositing the input reference images based on the minimal geometry information.

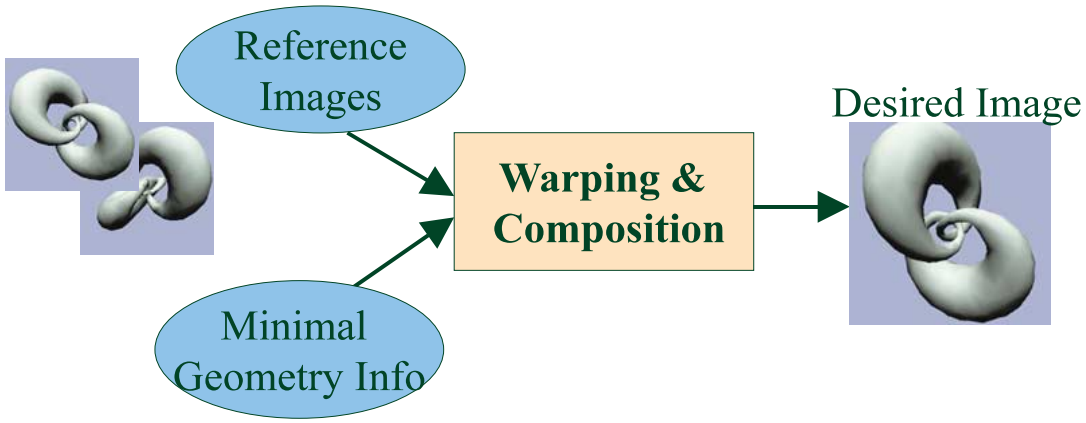


Figure 1-2: Image-based computer graphics.

One major reason of the emergence of image-based computer graphics is the need of time-critical modeling and rendering. When using reference images as input, the rendering (warping and compositing) time is no longer associated with the scene complexity, instead it now depends on the resolution of the images. Moreover, modeling is no longer a process of mimicking real world object using geometry representation, instead modeling becomes taking photographs of the scene.

1.2 Thesis Contributions

The main contribution of this thesis is a collection of concepts and algorithms that model and render realistic images efficiently in both geometry-based and image-based computer graphics. The introduction of these concepts and algorithms pushes a step toward to the ultimate goal of time-critical modeling and rendering.

The first contribution is a brand new geometry-based simplification algorithm, named as *adaptive skeleton climbing* [POST98, POST97]. It generates simplified polygonal isosurface representation directly from volume data. The coarsest mesh generated is about 4-25 times fewer triangles than the existing common method. Nevertheless, the geometry details are preserved. In other words, the resultant mesh can be displayed 4-25 times faster than that generated from the common approach. At the same time, there is minimal loss in visual quality. More importantly, the proposed algorithm generates simplified mesh *on-the-fly* and runs in a short period of time. As opposed to the mesh optimization approach [HOPP93] which is post-processing algorithm and requires tens of minutes to execute. An attractive feature of the proposed algorithm is that it generates coarser mesh in a smaller period of time than that of generating finer mesh.

The second contribution is a *concept of measuring pixel BRDF* [WONG97b, WONG97a] in image-based computer graphics. Most existing techniques concentrate on finding the correct view while assuming the illumination of the scene is fixed. Hence there is no way to adjust the illumination in the desired image. The image-based renderer can only change the viewpoint and but cannot change the lighting condition. In other words, it is not a complete renderer. A few approaches have been proposed to allow the adjustment of illumination in the desired image. However, they are either only for still image or with a lot of restrictions, such as Lambertian object assumptions. More seriously, most of them are *uncontrollable* (described detailly in Chapter 5). We proposed a *unique* concept and algorithms that allow a *general*, *controllable* illumination for arbitrary image containing both Lambertian or highly specular surfaces.

To control the illumination, we need to know the reflectance of the object surface. However, for image-based computer graphics, we are no longer accessible to the geometry models and surface descriptions. All we have are a set of reference images and minimal geometry information. We proposed a new concept that regards a image plane pixel as an ordinary surface element and measures its *apparent* reflectance. Extracting this reflectance information from the reference images, not just allows the change of the illumination, but also allows us to correctly synthesize image of the same scene illuminated by light source which does not present in the input reference images.

Using the proposed concept, we extend two major image-based representations to include illumination which is not available before. The final contribution is that we apply a set of compression schemes [WONG98], including spherical harmonics transform and LBG vector quantization, to compress the new pixel reflectance data in order to make it *practical*.

1.3 Thesis Outline

This thesis is divided into two main parts. Part I describes our work in geometry-based time-critical model and rendering while Part II describes the image-based approach. Readers are referred to Chapters 2 and 5 for a quick overview of the motivations of our work.

Part I begins by presenting the problem of geometry simplification and overviewing the related work in Chapter 2. We will also discuss the motivation of introducing a new geometry-based algorithm.

The following two chapters discuss the details of the new *adaptive skeleton climbing* algorithm. Chapter 3 first describes how the volume data is partitioned into variable-sized rectangular boxes whose size is adaptive to the geometry of the enclosed isosurface. In Chapter 4, the process of generating simplified triangular mesh is described.

Then, in Part II, we starts by introducing the current status of image-based computer graphics. The missing of controllable illumination is discussed when reviewing the existing image-based techniques. Our motivation to include illumination into image-based computer graphics will also be described.

Since image-based computer graphics is a new area, many fundamental concepts are still being developed. In Chapter 6, we first describe a fundamental model proposed by McMillian [MCMI97] for image-based computer graphics. It is known as *plenoptic function*. We will later point out that the original formulation of plenoptic function is not very suitable for computer graphics which requires the control of illumination. By modifying the original plenoptic function, we propose to use a new fundamental model for computer graphics, which we call a *plenoptic-illumination function*.

In Chapter 7, we introduce the concept of measuring pixel reflectance. The capturing of the pixel reflectance and the manipulation of them to re-render the image-based scene will be discussed in details. Some results are also shown to verify the concept.

Chapter 8 applies the proposed concept to two major image-based data structures, light field [LEVO96, GORT96] and panorama [CHEN95a] in order to include illumination. Note that these two image-based data structures do not allow any control of illumination originally.

To make the idea practical, we need a good compression scheme in order to store the pixel reflectance data efficiently. In Chapter 9, we describe a series of compression algorithms that can compress the huge reflectance data to a compact size.

Finally, in Chapter 10, we conclude by comparing the geometry-based and image-based approaches. The pros and cons of the proposed concept and algorithms will be discussed in detail. We will also discuss the future directions.

Part I

Geometry-based Approaches

Chapter 2

Geometry-based Approaches

The major key to geometry-based time-critical modeling and rendering is *scene simplification*. Although we can also improve the rendering speed by simplifying the physical laws, this kind of simplification also reduce the quality of the synthesized images. Moreover, it is still not very helpful in rendering a very large scale scene.

As mentioned before, the research of time-critical modeling and rendering is a research of making use of the limitation of human perception. The objects that are too small or too far away can be simplified. The objects that are occluded can be simplified or even removed (if it does not contribute any radiance in the final image). The portion of object with a smooth geometry can be represented by larger simple geometry representation, such as planar surface. Small geometry details such as cotton fibre can be approximated by texture mapping. All these approximations utilizing the human perception limitations. The phrase *multiresolution representation* is used to describe the data structure used in achieving the mentioned effects. A review of previous work is described in Section 2.3.

Each simplification algorithm is specially designed for a specific geometric representation. Several representations of geometry models are proposed. Each representation owns a unique feature which is not replaceable by another. A brief look of the major geometry representations are described in Section 2.1.

Without exception, we proposed a simplification algorithm which is specific to a representation, the planar surface model. Planar surface model is the most popular geometry representation due to its simplicity and the wide popularity of graphics accelerator. Our algorithm can also be regarded as a converter which converts the volumetric representation to planar surface model. During the conversion, simplification takes place to simplify the resultant planar surface representation.

2.1 Geometry Representations

A geometry representation can be characterized by its nature of describing (resembling) shapes in three-dimensional Euclidean space. The major reason of using Euclidean geometry in modeling is

because three-dimensional Euclidean geometry is the geometry of our real world. We are familiar to it. Moreover, we often model real world scenes in computer graphics.

Surface model is a popular model in computer graphics. It represents object as a set of two-dimensional surfaces embedded in the three-dimensional space. The two-dimensional surface can be planar or curved. Planar surface models, such as polygonal mesh and triangular mesh, are the most popular geometry representation in current graphics system, due to their simplicity and the wide availability of graphics accelerators which are specially designed for them. Curved surface models are usually expressed in parametric forms and controlled by a set of control points. They are useful in manual object modeling. Examples are Bezier, B-splines, and NURBS surface patches.

Solid representation describes model by a solid volume instead of hollow volume enclosed by surfaces. There are many variants of solid representations. Constructive solid geometry (CSG) is one such solid representation. It is a Boolean combination of primitive solids, such as sphere, box and cone, etc. It is usually used in modeling mechanical components. However, it is not very useful in representing object with irregular curved shape. Binary spatial partitioning (BSP) is another solid representation which is very similar to CSG. But the only primitive allowed is the half-space, the half space partitioned by a plane.

For the previous two representations, we can ask whether a point is inside or outside the enclosed volume of the modeled object. There is a representation which does not allow the query of such question. One such example is the implicit surface [BLOO97]. It does not define a surface explicitly, but it defines a 3D field such that each point in space associates with a value. What we can ask is only the value at a specific point. Implicit function is good at representing natural objects such as water, fire and cloud, since natural objects do not have any rigid or solid shape. A discrete variant is the volumetric data. It is a lattice of discrete values. Volume data is often collected from Computed Tomography (CT). Another discrete variant is heightfield. Heightfield is a two-dimensional function $f(x, y)$ which returns a scalar height value at each (x, y) position. In most cases, it is used in describing landscape.

Even though the planar surface model is the most popular, objects are seldom created directly in planar surface representation due to the tediousness. If the object is modeled by hand, object is usually created in curved surface representation and then converted to planar surface model. If the object is acquired from 3D scanner such as CT or range scanner, it is first created in volumetric representation and then converted to the planar surface model using *isosurface extraction* technique.

2.2 Motivation

Our work concentrates on finding simplified planar surface model. Early surface models are mostly constructed by hand. However, due to the increasing popularity of the 3D scanners, there is an increasing trend in acquiring planar surface models from 3D scanning. Hence, the need of an efficient isosurface extraction algorithm becomes apparent.

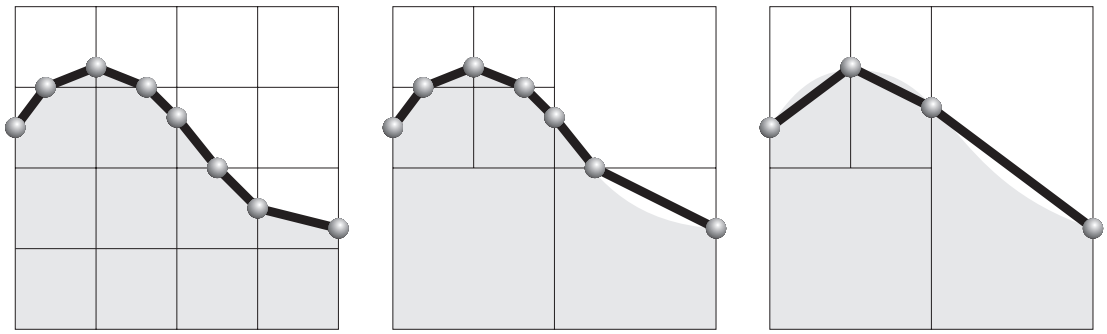
An isosurface extraction is an algorithm that given the volume data and a *user-specified threshold*, it generates a planar surface representation that approximates the true *isosurface*. An isosurface is a surface that every point on the surface has the same value as the given threshold.

A common isosurface extraction algorithm is the marching cubes algorithm [LORE87]. It is a fast and simple algorithm generating triangular mesh by looking up a 256-entry voxel-cube configuration table. The major advantage of this algorithm is its speed. However, its major criticism is that it produces exceedingly huge amount of triangles. Although there are several post-processing mesh simplification algorithms [HOPP93, SCHR92, TURK92] that can reduce the triangles in the generated mesh, they usually require tens of minutes to simplify the mesh.

An ideal isosurface extractor should produce minimum amount of triangles that can sufficiently approximate the true isosurface and at the same time it can generate the mesh in a short period of time. We designed a new isosurface extraction algorithm with an on-the-fly mesh simplification. That is, given a volume data and the threshold, the algorithm directly converts it to a simplified triangular mesh. Since the algorithm directly generates the simplified mesh, the algorithm is much faster than those post-processing mesh reduction algorithms.

The problem of marching cubes is that it subdivides the volume into unit size voxel cubes (a voxel cube composes of eight neighbor voxel samples) and then generates triangles within the cubes. The number of triangles is clearly related to the size of the cubes. If larger cubes are used, the number of triangles will reduce. Unfortunately the size of the cube is independent of the geometry of the enclosed isosurface in the original algorithm. Therefore, even the enclosed isosurface is smooth enough to be approximated by larger triangle, many small triangles are still generated due to the unit size partitioning approach. Figure 2-1(a) illustrates the problem of unit size partitioning in 2D. In 2D, the isosurface becomes iso-curve and cube becomes square. Even though longer edges can be used in approximating the iso-curve in the diagram, the 2D marching cube algorithm still generates 7 edges in this example.

Wilhelms and Van Gelder [WILH92] and Shekhar *et al.* [SHEK96] noticed this problem. They



(a) (b) (c)

Figure 2-1: Volume partitioning schemes. (a) 2D marching cube, (b) quadtree (2D analogy of 3D octree), (c) 2D adaptive skeleton climbing.

used octree to partition the volume data. The idea is to fit smooth region with large cubes while complex region with small cubes. The 2D analogy of octree is quadtree shown in Figure 2-1(b). However octree is still quite restrictive in partitioning the volume. The partitioned region should always be a cube. Moreover, unnecessary fragmentation occurs in some cases due to the hierarchical octree structure. In the 2D example of Figure 2-1(b), the quadtree algorithm still generates 6 edges.

To overcome the restriction imposed by octree organization. We apply binary tree organization along each dimension of the partitioned boxes. This approach allows more flexibility in partitioning the volume. The partitioned region needs not be a cube. It can be a rectangular box. This approach also reduces the number of unnecessary fragmentation which is quite often in the octree scheme. Figure 2-1(c) shows the 2D analogy of our partitioning scheme. Note the subdivided regions can now be a rectangle. This flexibility further reduces the number of generated edge to 3 edges.

All adaptive marching cube approaches [SHU95, WILH92, SHEK96] face the problem of gap-filling. Gap exists between a large box and its small neighbors. One more unique feature of our algorithm is that it does not need to fill the gap because the gap is prevented by sharing information among neighbor boxes. We will go through the details in the next two chapters. Before that, let us review some of the related work.

2.3 Related Work

Due to the increase of computer power and capacity, more complex scene can be modeled and rendered. However this increase does not satisfy the need of rendering complex scenes. Instead there is

an increasing demand of rendering even more complex scenes. But the increase in computer capability allows more sophisticated algorithms in the area of time-critical modeling and rendering to be developed.

There is a significant amount of work have been done in this area. We will concentrate on the work generating simplified planar surface models. These work can be roughly partitioned into two major classes based on the view dependency of the algorithm. In the next two subsections, we will try to classify the work along this criteria.

2.3.1 View Independent Simplification

Most early simplification algorithms are view independent. Williams [WILL83] proposed an algorithm which simplifies a polygonal mesh organized in grid structure. Due to the special structure of the mesh, simplification can be easily done by low-pass filtering the vertices.

Another typical type of algorithms converts heightfield data to simplify polygonal mesh for display [SCAR92, POLI93]. This type of algorithm is commonly used for the application of real time flight simulation. A common approach is first to use a large triangle patch to approximate the landscape. If the approximation error is greater than a user-specified threshold, the large triangle will be subdivided into smaller patches. The process repeats until the error is below the threshold. This approach can be called *adaptive subdivision*. DeHaemer and Zyda [DEHA91] developed an algorithm that converts 3D range data to simplified polygon mesh. They used two approaches. One is the mentioned adaptive subdivision (top-down) approach. The another one is a polygon growing (bottom-up) approach. The polygon growth algorithm starts from a small polygon, try to merge it with its neighbors to form larger facets. The merging stops whenever the approximation error is greater than the threshold.

Another common type of simplification algorithm converts volume data to simplified polygonal mesh given a threshold [WILH92, SHEK96, SHU95]. Wilhelms and Van Gelder [WILH92] and Shekhar *et al.* [SHEK96] used octree partitioning to subdivide the volume into variable-sized cubes in order to reduce the number of generated triangles.

Other algorithms accept general mesh as input. Rossignac and Borrel [ROSS93] presented an algorithm that uses a signal processing approach. The algorithm treats the vertices on the mesh as sample points in 3D. By grouping neighbor points and replacing them with a representative point, it

can then generate simplified models. Garland and Heckbert [GARL97] proposed to use the quadric error metrics in finding the representative vertices. Schroeder, Zarge and Lorenson [SCHR92] proposed a mesh decimation algorithm. The algorithm removes less important vertices on smooth regions and performs retriangulation to preserve the geometry details of the mesh. Cohen *et al.* [COHE96] used a similar approach but the process is guided by *simplification envelopes* (inner and outer envelopes). Turk [TURK92] proposed another algorithm with a completely different approach. The algorithm places sample points on the input dense mesh and re-tile the surface based on the distribution of the sample points. To prevent non-uniform distribution of sample points, a repulsion force is applied at each sample point. Hoppe *et al.* [HOPP93] applied the optimization technique to optimize the number of triangles in the input mesh. Hence, the main problem is how to design a good objective function.

In general, view-independent algorithms simplify the objects only based on the geometry complexity of the input object. Since they don't make use of any viewing information to simplify the object, the simplified object is hence view independent. That is, no matter where the viewpoint is, there is no need to modify the simplified object.

To use the simplified objects in multiresolution modeling, the original high resolution object is first converted to multiple simplified versions using any of the mentioned algorithms. During rendering, high resolution version is displayed when the viewpoint is close to the object. On the other hand, low resolution version is used when the viewpoint is far away. The criteria of choosing which version depends on the distance between the viewpoint and the object. This capability of switching different resolutions of the object is called *level of detail*. This technique is first described by Clark [CLAR76]. It is usually not very difficult to modify a renderer to support the level of detail. This is another advantage of using view-independent algorithms. However, visual artifact may appear when switching from one resolution to another. The magnitude of artifact depends on several factors: the number of resolutions stored, the specific resolution chosen, the rendering algorithm and the geometry complexity of the object. Funkhouser and Séquin [FUNK93] used constrained optimization to choose a level of detail and rendering algorithm to generate images within the target frame rate.

2.3.2 View-Dependent Simplification

Viewing information, such as position of the viewpoint, viewing direction, and field of view, provides extra information besides the geometry complexity of the object or scene. With this information, we usually can simplify more. For example, scenes/objects outside the viewing frustum can be simplified

or removed. Objects closer to the eye should be represented in more detail while those far objects can be simplified. The trade-off of designing a view-dependent algorithm is an increase in algorithm complexity. When the viewpoint or the viewing direction changes, the simplification done for the previous frame has to be adjusted. Hence the renderer used must be specially designed to forward the viewing information to the simplification algorithm at the back end. Moreover, the data structure storing the polygonal mesh is also specially designed to support the *selective refinement* on the mesh. That is, to represent portion of the mesh in high resolution while the rest in low resolution.

Lindstrom *et al.* [LIND96] proposed a view-dependent simplification algorithm for heightfield data. The algorithm uses a regular grid to represent the heightfield and employs a screen-space threshold to control the error of the projected image of the grid. By projecting the polygon to the screen space, error can be measured in this space. Measuring error in screen space effectively make use of the viewing information. The polygon will be further subdivided if the error exceeds the threshold.

For arbitrary mesh, a dedicated data structure is needed to support selective refinement. Such data structure is basically a *history* recording how the highest resolution mesh is simplified to the lowest resolution one. With this history, the mesh can be selectively refined by replaying the simplification process for a selected region in the mesh. The criteria of refinement usually depends on a screen-space threshold. Xia and Varshney [XIA96] used *ecol/vsplit* transformation (a step of simplification, for example merging two vertices to one) during the simplification process. Hoppe [HOPP96, HOPP97] proposed the progressive mesh which is constructed by simplifying the mesh using unconstrained, geometrically optimized *vsplit* transformation. Popović and Hoppe [POPO97] used a more general transformation known as *generalized vertex split* in their simplification process.

Previously mentioned Rossignac and Borrel's approach [ROSS93], which treats the mesh as a set of unstructured vertices, can be extended to view-dependent version. Luebke and Erikson [LUEB97] developed a view-dependent polygon simplification based on the similar idea by recording the process of selecting the representative vertices.

2.4 Summary

In this chapter, we introduce the major key to geometry-based time-critical modeling and rendering, *scene simplification*. Due to the simplicity and wide popularity of graphics hardware, the planar surface model is the most popular geometry representation. Hence most of the research are done for planar surface model. The motivations of our work is to generate simplified triangular mesh during

the isosurface extraction in a short period of time. Our algorithm is a view-independent simplification algorithm.

Chapter 3

Volume Partitioning

We describe here a direct construction of isosurfaces with between 4 and 25 times fewer triangles than marching cubes algorithms [LORE87, WYVI90] (depending on the complexity of the volume), in comparable running times. Hence more complexity can be handled at interactive speed. The proposed algorithm is named as *adaptive skeleton climbing*. Since we construct the isosurfaces by first finding iso-points on grid edges (1-skeleton), then iso-lines on faces (2-skeleton) and finally isosurfaces within boxes (3-skeleton), it is known in topology as *skeleton climbing*. Moreover, the size of the constructed boxes will adapt to the geometry of the isosurface (*e.g.* larger boxes enclose smoother regions), hence it is *adaptive*.

The proposed algorithm can generate isosurfaces in multiple resolutions directly. The coarseness of the generated meshes is controlled by a single parameter. The triangle reduction is done on the fly as the isosurfaces are generated without going through a separate postprocess. The proposed on-the-fly triangle reduction approach can preserve geometric details of the true isosurface and generate more accurate meshes because it directly makes use of the voxel values in the volume. In contrast, the postprocessing triangle reduction approaches [DEHA91, HOPP93, SCHR92, TURK92] usually use the indirect geometrical information from the approximated dense meshes.

Our approach is quite different from the adaptive marching cubes algorithms [SHU95, SHEK96]. No crack-patching step is needed because we build compatibility (described shortly) into the faces where cells meet before generating triangles.
The algorithm can be intuitively subdivided into four major steps:

1. 1D Voxel Analysis and Grouping.
2. 2D Adaptive Skeleton Climbing.
3. 3D Adaptive Skeleton Climbing.
4. Isosurface Extraction.

In order to fit large triangles to smooth isosurfaces, the content inside the volume must be first analysed. Our goal is to find out the size-maximal boxes that enclose the smooth isosurfaces. To

find size-maximal boxes in 3D (step 3), we have to find the size-maximal rectangles in 2D (step 2). This further leads us to the need of finding out the length-maximal segments in 1D (step 1). In step 3, we build 3D *simple* boxes (described in Section 3.3) whose sizes are closely related to the geometry complexity of the enclosed isosurface. Information is then shared between adjacent boxes to prevent existence of gap. And finally in step 4, the triangular mesh is generated. Figure 3-1 shows the processes of adaptive skeleton climbing graphically. The basic idea is to group voxels first in 1D (segments), then in 2D (rectangles) and finally in 3D (boxes).

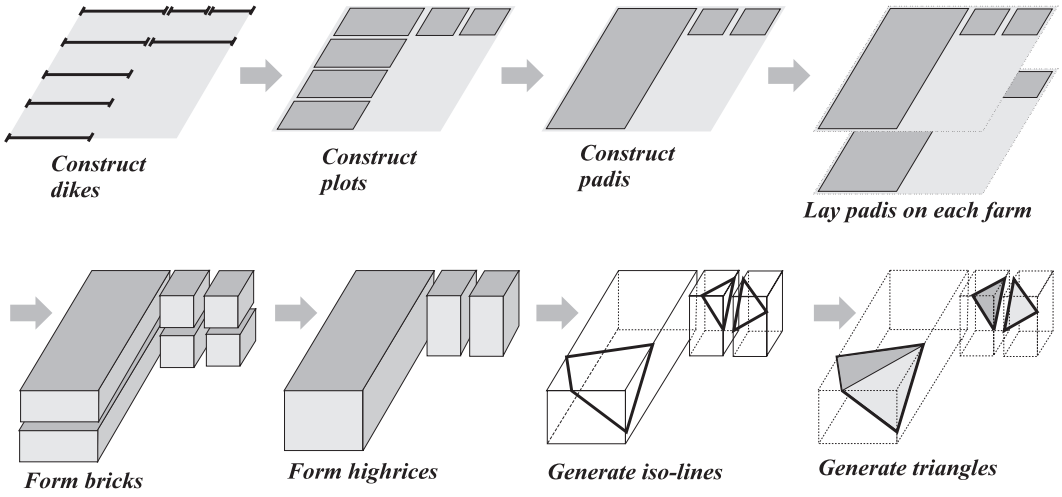


Figure 3-1: Overview of adaptive skeleton climbing.

In this chapter, we will describe the first three steps of the algorithm. The goal of executing the first three steps is to partition the volume into size-maximal boxes. It is actually a bottom-up construction process, instead of a top-down subdivision process. Section 3.1 describes manipulation and grouping of the basic 1D data structures in detail. Section 3.2 carries on the grouping in 2D. Section 3.3 discusses the construction of simple boxes. Details of triangular mesh generation are described in Chapter 4.

3.1 1D Data Structures and Manipulation

We start the volume analysis in 1D, *i.e.* consider a linear sequence of voxel samples. Try to find out the length-maximal subsequences of voxels with *simple* structure (described shortly).

It helps to think of the volume data as giving sampled values at points (dots in Figure 3-3), rather than voxel values filling cubes. For the sake of discussion, let's define the 1D terminologies and data

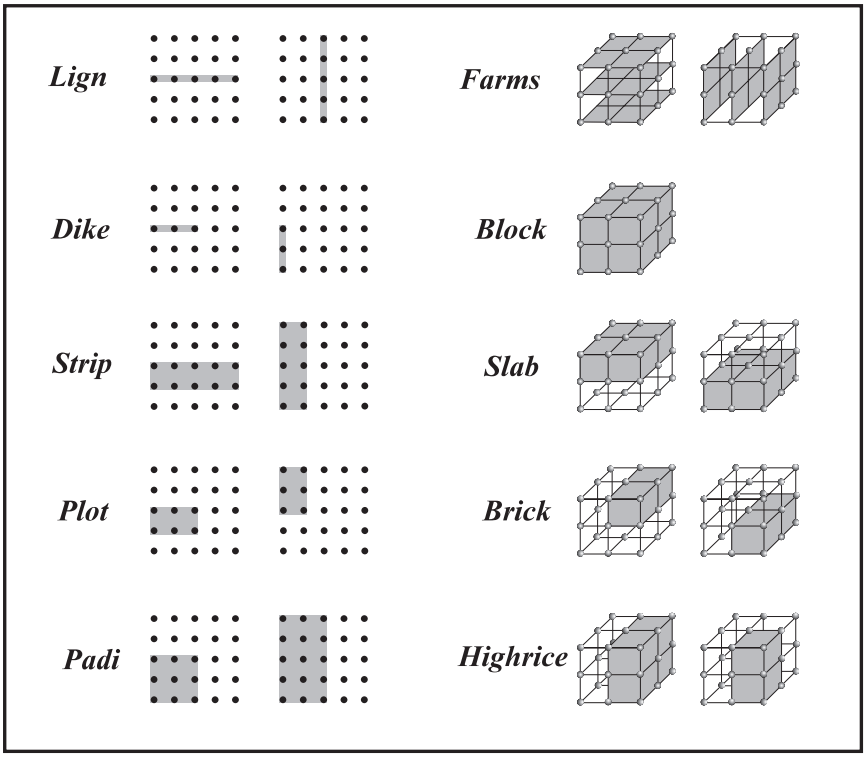


Figure 3-2: Glossary of various terminologies.

structures. To provide a quick reference to all terminologies (1D, 2D & 3D), a graphically-explained glossary table is available in Figure 3-2. A line of 2^n+1 sample points is called *lign* (Figure 3-3(a)) where n is an integer ≥ 0. A *dike* (Figure 3-3(b)) is a segment of lign which covers voxel samples in the interval $[a2^m, (a+1)2^m]$, where $0 \leq m \leq n$ and $0 \leq a < 2^{n-m}$, both a and m are integers. That is, all dikes are organized in a binary tree (Figure 3-4). The reason to use binary tree on 1D data instead of octree on 3D data [SHEK96] is that binary tree provides more flexibility in grouping voxels.

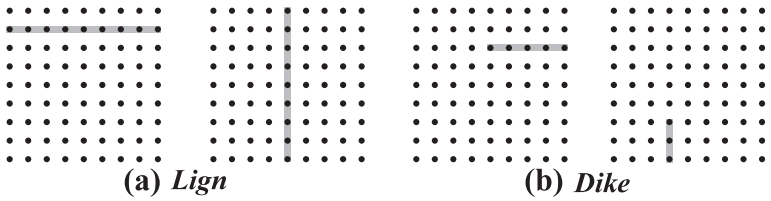


Figure 3-3: Basic 1D data structures.

Figure 3-4(b) shows the binary tree organization of 15 dikes which covers 9 voxels. The voxels

covered by each dike are shown graphically in Figure 3-4(c). The nodes are labeled in a breadth-first-search order, with the root node as 1. With this dike-labeling scheme for each lign, we can store two length-$(2^{n+1}-1)$ arrays of dike information, *occupancy* and *simple dike*, for a lign of 2^n+1 samples. For simplicity, let $N = 2^n$ for short hand.

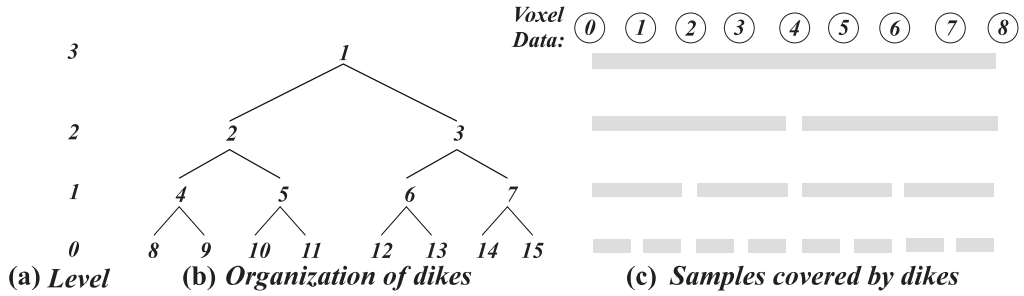


Figure 3-4: Binary tree organization of 1D voxel data.

The *occupancy array* of a lign describes the presence of iso-points (1D analogy of 3D isosurface) on its dikes. With this occupancy array, we can accurately locate the position of iso-point and how the isosurface crosses the lign. Now, let us denote the voxel sample with value above or equal to the threshold (τ) as $\bullet$, and sample with value below τ as $\circ$. Then the binary value of the i^{th} entry in the occupancy array means:

$$\texttt{occ}[i] = \begin{cases} 00_2 & \text{all samples in dike } i \text{ are on the same side of } \tau. \\ 01_2 & \text{if dike } i \text{ is crossed by isosurface once, upward } \circ \rightarrow \bullet. \\ 10_2 & \text{if dike } i \text{ is crossed by isosurface once, downward } \bullet \rightarrow \circ. \\ 11_2 & \text{if dike } i \text{ is crossed by isosurface more than once.} \end{cases}$$

Note the binary values symbolize the crossing conditions. For instance, if the isosurface crosses the dike once and the voxels within the dike change from $\circ$ (0) on the left to $\bullet$ (1) on the right, then the value in `occ[]` is 01_2 ($\circ \rightarrow \bullet$). Once the entries of unit dikes (leaf nodes of the binary tree) are initialized directly from volume data, the entries of the non-unit dikes (upper interior nodes) can be found by a recursive bitwise OR operations on the leaf nodes since the value in `occ[]` are specially designed.

$$\texttt{occ}[i] \texttt{ := (occ}[2i]\texttt{) OR (occ}[2i+1]\texttt{)}$$

Another array is *simple dike array*. It tells us the length-maximal *simple* dikes inside the lign. A dike i is *simple* if $\texttt{occ}[i] < 11_2$; that is, the dike is crossed at most once by the isosurface. The entry `simple[`i`]` holds the index of the length-maximal simple dike with the same left end as dike i.

Intuitively speaking, simple dike array tells us which voxels can be grouped together without violating the binary boundary due to the tree organization (*binary edge* for short) and the simplicity constraints. The length-maximal simple step following dike i is the dike `simple[`i`+1]`. By performing the following pseudocode fragment, we can walk through the lign in steps of length-maximal dikes in an efficient way.

```
current := simple[1]
while current ≠ "end of walk mark"
    current := simple[current+1]
```

Figure 3-5(a) illustrates that a lign is subdivided into length-maximal dikes (shown in black in Figure 3-5(b)). In this 9-voxel lign example, the lign is subdivided into 4 dikes. The first two dikes are unit dikes, since the isosurface crosses both of them. Although the isosurface crosses the rest of the segment only once, it is still subdivided into two dikes due to the binary edge constraint imposed by the binary tree organization.

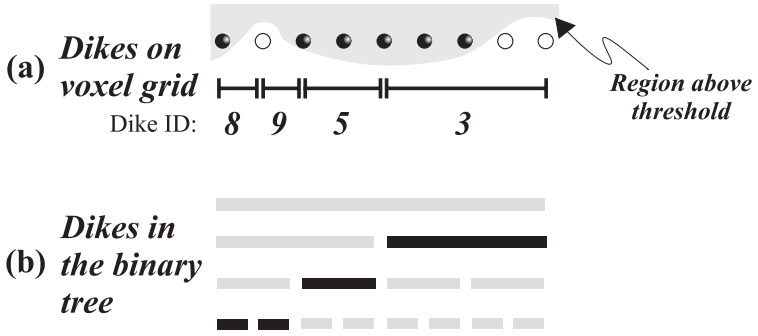


Figure 3-5: (a): The lign is subdivided into length-maximal dikes by algorithm `InitSimple`. (b): The dikes visited when walking through the lign.

Values in `simple[]` are found by a binary tree depth-first-search traversal, whose pseudocode is shown in Figure 3-6. Note that due to the binary edge constraint, the subdivision may not be always minimal (Figure 3-5). But this restriction simplifies the merging process in the 2D adaptive skeleton climbing discussed in next section.

3.2 2D Adaptive Skeleton Climbing

Next, we go on to the 2D data structure and find out the size-maximal rectangles of voxel samples with *simple* structure.

Input: initialized occupancy array and unfilled simple dike array
Output: filled simple dike array
Algorithm:

```
InitSimple(myID, legacy):
    if current dike is a bottom node (unit dike)
        return myID
    if parent dike is simple
        if current dike is right child, it must be simple
            simple[myID] := myID
        else /* inherit the simplicity */
            simple[myID] := legacy
        /* Propagate simplicity downward */
        InitSimple(2*myID, simple[myID])
        InitSimple(2*myID+1, simple[myID])
    else /* parent is not simple */
        if current dike is simple, i.e. occ[myId]< 11_2
            simple[myID] := myID
            /* Propagate simplicity downward */
            InitSimple(2*myID, simple[myID])
            InitSimple(2*myID+1, simple[myID])
        else /* Is the left descendant simple? */
            descendent := InitSimple(2*myID, dummy)
            InitSimple(2*myID+1, dummy)
            simple[myID] := descendent
    /* Propagate simplicity upward */
    return simple[myID]
```

Figure 3-6: Algorithm `InitSimple`.

3.2.1 Data Structures

The 1D data structures allow us to group voxels into length-maximal simple segments (dikes). Similarly, in the 2D, we want to group voxels to form size-maximal *simple* rectangles. Consider a $(N+1)\times(N+1)$ *farm* of voxel samples, with $N+1$ horizontal and $N+1$ vertical ligns, each with its own occupancy and simple dike arrays. First, let's define the 2D terminologies (Figure 3-2) and data structures. A *strip* (Figure 3-7(a)) consists of two consecutive ligns. A *plot* (Figure 3-7(b)) is analogous to the dike which consists of two consecutive dikes.

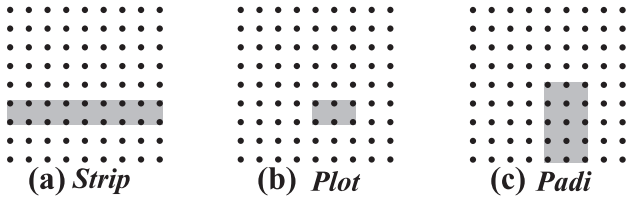


Figure 3-7: The 2D data structures.

Plots are also organized by a binary tree. Similarly a plot is *simple* if and only if its two dikes are also simple. Hence, we can define a *simple plot array* which is similar to the simple dike array.

Since the shorter dike has a larger dike ID, the length-maximal simple plots can be easily found by performing a MAX operation on each pair of elements in the simple dike arrays of the two consecutive ligns.

```
strip[j].simple[i] := MAX(lign[j].simple[i],
                          lign[j + 1].simple[i])
```

Figure 3-8 shows one such operation graphically. The calculated plots are overlaid with the voxel samples in Figure 3-8(b). Note that each plot is crossed at most twice by the isosurface.

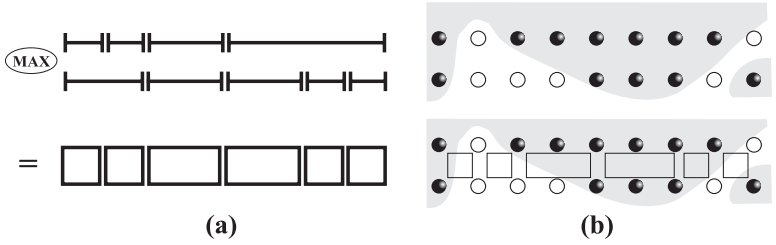


Figure 3-8: Length-maximal plots from consecutive ligns.

3.2.2 Merging Plots to Form Padis

A rectangle with dikes as sides is called *padi* (Figure 3-7(c)). A padi is *simple* if all plots inside it and its four side dikes are simple. Our goal is to subdivide the 2D farm of voxels into size-maximal padis. To do so, neighboring simple plots are merged to form simple padis(Figure 3-9), as large as possible. Note there is no unique way to merge plots. Different merging strategy gives different sets of padis. Figure 3-9 shows two alternatives when merging the two consecutive strips. Even an optimal merging is found for 2D, it may not yield an optimal merging in 3D (discussed in next section). Moreover, a fast algorithm is crucially required since it will be frequently executed. A slow optimistic algorithm is useless in this case. Hence we do not use any optimistic algorithm to search for the optimal merging. A heuristic bottom-up merging (ASC2D, Figure 3-10) is used due to its efficiency and simplicity.

The algorithm ASC2D accepts $2N$ initialized simple plot arrays as input. There are N arrays for horizontal strips and N arrays for vertical strips. The array can be initialized by the MAX operations discussed previously. The basic idea of the algorithm ASC2D in Fig 3-10 is as follows. Let us denote the horizontal direction from left to right as direction x and vertical direction from bottom to top as direction y. For each length-maximal plot on each horizontal strip (x-strip), expand it in y direction

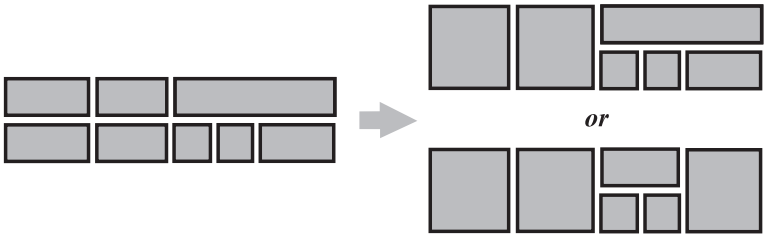


Figure 3-9: Merging plots to form padis.

Input: $2N$ initialized simple plot or layout arrays.
N for horizontal strips & N for vertical strips.
Output: A set of size-maximal padis (and iso-lines).
Algorithm:
Initialize an empty candidate list of padis.
For each x-strip (horizontal strip)
  /* Expand the plots to form padis */
  For each length-maximal simple plot a
    Let rectangle $r := a$
    While $\exists$ neighbor simple plot b on the adjacent strip
      $r := r \cup b$ (Figure 3-9)
    Subdivide r in y-direction into pieces according to the
    binary edge restriction and give k padis $r_1, r_2, \ldots r_k$ (Figure 3-11)
    For each generated padi r_i
      For each padi p_j inside the candidate list
        If p_j encloses r_i
          Delete r_i
        If r_i encloses p_j
          Remove p_j from the candidate list
        If p_j partially overlaps with r_i
          Clip r_i (Figure 3-12(a))
      If r_i is not removed
        Add r_i to the candidate list
/* Optional Iso-line Generation */
For each padi p_j in the candidate list
  Generate iso-line for p_j by looking up the table (Figure 3-15).

Figure 3-10: Algorithm ASC2D.

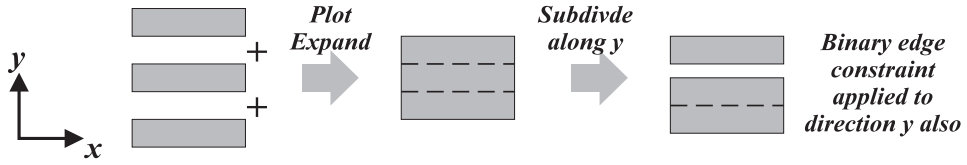


Figure 3-11: Plots are first merged to form rectangle. The rectangle is then subdivided along y direction to satisfy the binary edge constraint applied to the y direction.

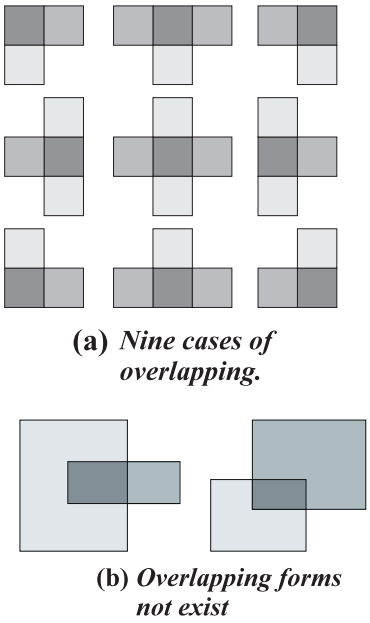
by merging it with consecutive plots having the same length (Figure 3-9). Note the neighboring plots need not be length-maximal. With the binary edge constraint, it is more likely to find neighbor plots with same length and align to each other. A candidate rectangle is then formed. Since the binary edge constraint is also applied to the vertical direction, this rectangle is subdivided to form size-maximal padis (Figure 3-11).

During the execution of the algorithm ASC2D, many padis will be generated. They may overlap with each other or one may enclose another. All padis which are enclosed by any other padi will be removed. Those overlapping padis will be clipped with each other. This seems to be tricky. Instead, only 9 overlapping cases will exist (Figure 3-12(a)). Overlapping cases like Figure 3-12(b) do not exist. Once again this is the advantage of applying binary edge constraint on both dimension. Figure 3-13 shows an example result of running the algorithm ASC2D. The generated padis are shown as rectangles among the voxel samples.

A layout of padi is generated as the result of ASC2D. This layout information is stored implicitly in the *layout arrays*. Layout array is very similar to the simple plot array but with the constraint that no plot may cross the boundary of any generated padi on the layout. For a farm of $(N+1) \times (N+1)$ voxels, $2N$ layout arrays are defined, N x-strips and N y-strips. The i^{th} entry in x-strip (y-strip) stores the index of the length-maximal plot that fits into the padi layout and shares its left (bottom) end with plot i. Figure 3-14 shows the padi layout of a 5×5 xy-farm, which is represented by x-strips (Figure 3-14(b)) and y-strips (Figure 3-14(c)). The reason to store the layout in this way is to simplify the simple box construction discussed in Section 3.3.

3.2.3 Iso-line Generation

Once the size-maximal padis are found, we can generate 2D iso-lines which separate $\bullet$ voxels from those $\circ$ voxels. Although we will not generate any iso-lines until the 3D boxes have been constructed (discussed in next section). For the sake of discussion, it is more convenient to discuss it here.



(a) ***Nine cases of overlapping.***

(b) ***Overlapping forms not exist***

Figure 3-12: Overlapping between two padis.

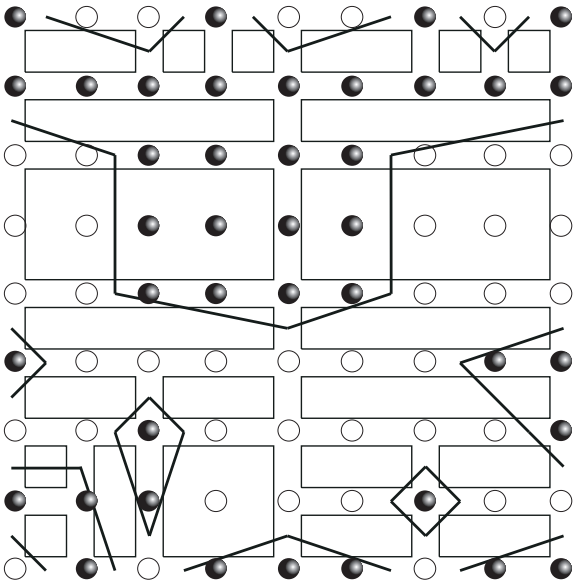


Figure 3-13: Example result of running algorithm ASC2D.

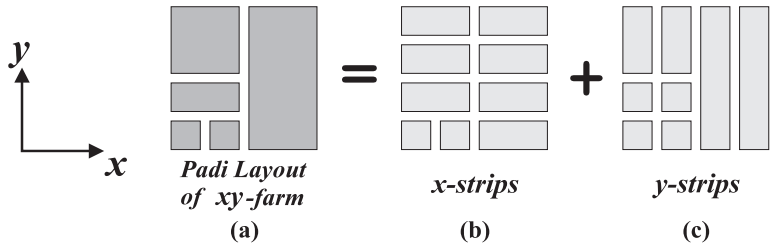


Figure 3-14: Storing the padi layout in layout arrays.

The iso-line can be efficiently generated by looking up a 2D padi configuration table in Figure 3-15, instead of a 3D voxel cube configuration table as in marching cubes algorithms [LORE87, WYVI90]. Figure 3-15 shows all possible padi configurations and their corresponding iso-lines. Note the padi need not be a square. Similar to the 3D voxel configurations, ambiguity also exists on 2D padi configurations (the two lower left configurations in Figure 3-15).

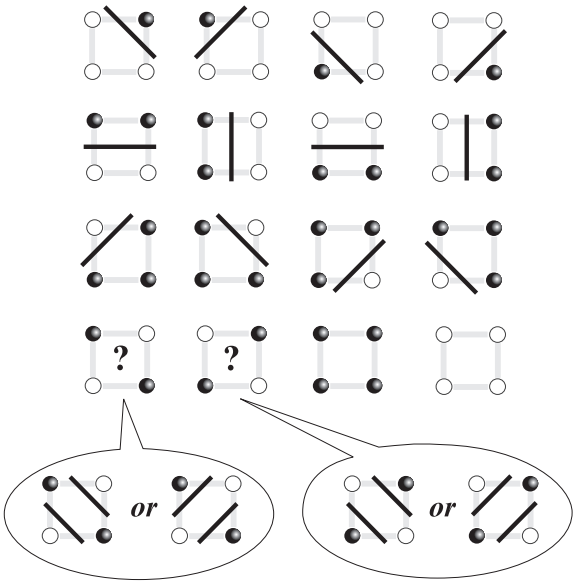


Figure 3-15: Generate iso-lines by a 16-entry table. Two ambiguous cases lead to subsampling.

The ambiguity with two diagonally opposite • corners can sometimes be resolved by subsampling at the center of the padi. However, wrong iso-lines still be generated in some cases (Figure 3-16). Where connectivity is crucial, software should warn the user of ambiguous cases and offer finer, more CPU-costly tools for local investigation. In many cases the warning is as useful to the surgeon, geologist or other user as any silently-attempted best guess by the software.

The generated padis (shown as rectangles) and iso-lines (shown as thick lines) are overlaid on the

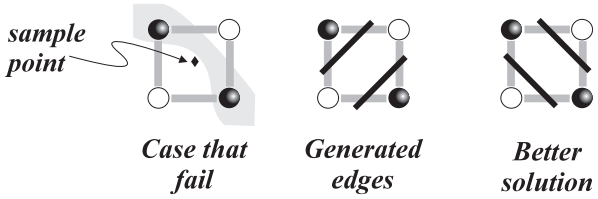


Figure 3-16: Bad ambiguity resolution by subsampling.

2D voxel grid in Figure 3-13. The algorithm isolates ○ from ● voxels, with 30 edges on 23 adaptive padis rather than the 46 edges on 64 unit squares. This is the key how we can reduce the number of triangles.

3.3 3D Adaptive Skeleton Climbing

By manipulating these 1D and 2D data structures, enough information is provided for us to construct 3D *simple* boxes. The information is implicitly stored as the 2D padis. Using this information, we go on to construct simple boxes by stacking simple padis.

3.3.1 3D Data Structures

Consider a $(N+1) \times (N+1) \times (N+1)$ voxel sample grid. For the sake of discussions, we now define the 3D terminologies (Figure 3-2) and data structures. All terminologies are graphically illustrated by a $3 \times 3 \times 3$ volume in the Figure 3-17(a). A *farm* contains $(N+1) \times (N+1)$ voxels on a 2D grid. A *slab*, analogous to a strip containing two consecutive ligns, consists of two consecutive farms. A *brick* in the slab has two matching padis in two consecutive farms as faces. A *highrice* is a rectangular box composed of stacked bricks.

It is convenient to call a farm xy-farm, xz-farm or yz-farm, according to which plane the farm is parallel to (Figure 3-17(b)). Similarly, a brick is called xy-brick if it is parallel to the xy plane.

3.3.2 Merging Bricks to Form Highrices

Our goal is to construct 3D *simple* rectangular boxes. We start by finding *simple* bricks. A brick is *simple* if the two padis forming it are also simple. A highrice is *simple* if all its component bricks are simple, and its six faces are *simple* padis.

Firstly, simple xy-bricks are identified. Then these simple xy-bricks will be stacked one by one to construct the simple xy-highrice. Note a highrice can be treated as composing of xy-bricks, xz-bricks

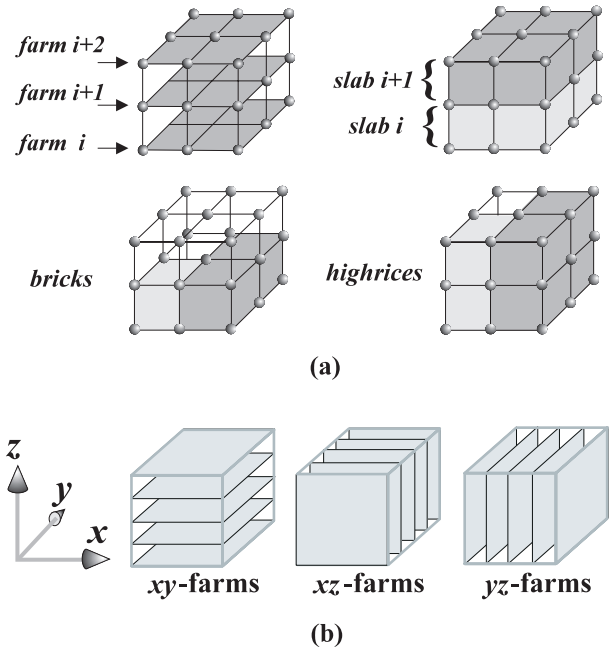


Figure 3-17: Data structures for the 3D algorithm.

or yz-bricks, depends on which dimension the bricks are stacked. We call a highrice the xy-highrice if it is constructed by stacking simple xy-bricks. In our algorithm, we only interest in finding the simple xy-highrices.

Figure 3-18 outlines the main algorithm. The first step generates the padi layout on each farm by algorithm `ASC2D` without the iso-line generation step. Then we identify the simple xy-bricks by performing simple `MAX` operations for each pair of corresponding entries in the layout arrays on two consecutive farms. Just like the 2D version. An example is shown graphically in Figure 3-19.

```
xy-slab[k].x-strip[j].layout[i] :=
              max(xy-farm[k].x-strip[j].layout[i],
                  xy-farm[k + 1].x-strip[j].layout[i])
xy-slab[k].y-strip[j].layout[i] :=
              max(xy-farm[k].y-strip[j].layout[i],
                  xy-farm[k + 1].y-strip[j].layout[i])
```

Next, neighbor bricks merge to form highrices in 3D (Figure 3-20), analogous to merging plots to form padis in the 2D case (Figure 3-9). Again there is no unique merging rule (Figure 3-20), and again we prefer a fast heuristic to a search for a suboptimal subdivision. For each size-maximal xy-brick on each slab, we stack the xy-bricks upward along z direction until no more simple bricks

Input: An $(N+1) \times (N+1) \times (N+1)$ 3D voxel grid.
Output: A set of maximal highrices and isosurface triangular mesh.
Algorithm:
/* Generate the padis on each farm */
For each farm out of (xy-farm, xz-farm and yz-farm)
    Find size-maximal padis by ASC2D.
Set layout arrays according to the padi layout on each xy-farm.
For each xy-slab
    Find the layout of xy-bricks by MAX operations. (Figure 3-19)
Initialize an empty candidate list of highrices.
/* Stack the bricks to form highrice */
For each xy-slab
    For each xy-brick a
        Let rectangular box $r := a$
        While $\exists$ neighbor simple xy-brick b on the adjacent xy-slab
            $r := r \cup b$ (Figure 3-20)
        Subdivide r into xy-highrices with the binary restriction
        applied along z and give k xy-highrices $r_1, r_2, \ldots, r_k$ (Figure 3-21)
        For each generated xy-highrice r_k
            For each xy-highrice h_l in the candidate list
                If h_l encloses r_k
                    Delete r_k.
                If r_i encloses h_l
                    Remove h_j from the candidate list.
                If h_l partially overlaps with r_i
                    Clip r_l.
            If r_k is not removed
                Add r_k to the candidate list
/* Sharing information among highrices */
For each farm (xy-farm, xz-farm, yz-farm)
    Reinitialize layout array to fit xy-highrice boundaries (Figure 4-2)
    Find a padi layout with ASC2D using new layout values
/* Iso-line Generation */
For each xy-highrice in the final candidate list
    For each padi on the surface of the xy-highrice
        Generate iso-lines. (Figure 3-15)
    Connect the iso-lines to form loops
    For each edge loop on the surface of xy-highrice
        Triangulate it and emit the triangles.

Figure 3-18: Algorithm ASC3D.

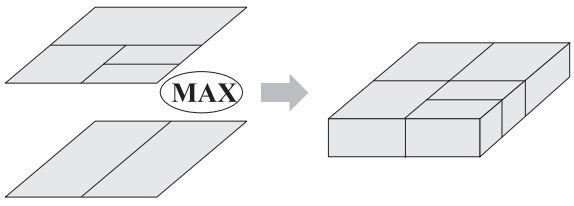


Figure 3-19: To find the simple bricks inside the slab, MAX operations are done on each pair of the layout arrays of neighboring farms.

available. Then this temporary box is subdivided along z direction to fulfill the binary edge constraint (Figure 3-21). During the highrice formation, the generated highrices may overlapped with each other or one may be enclosed by another. Any enclosed highrice will be removed. Overlapping highrices are clipped. Just like the 2D cases, the overlapping cases are quite restrictive due to binary edge constraint. Hence simplifies the clipping process.

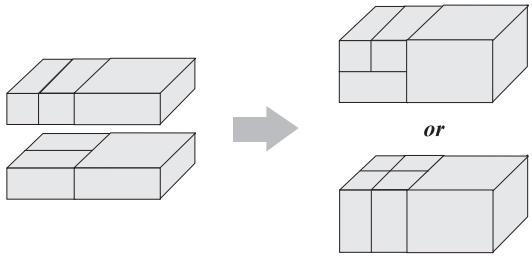


Figure 3-20: Merging bricks to form highrices

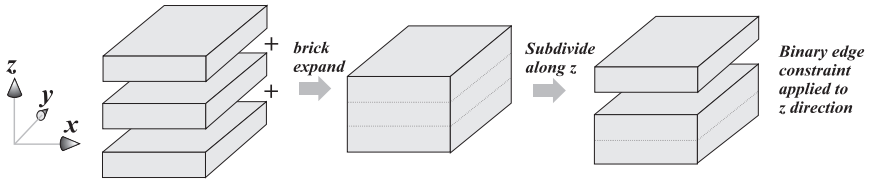


Figure 3-21: Bricks are first merged to form box. Then the box is subdivided along z to form highrices in order to fulfill the binary edge constraint.

3.4 Summary

In this chapter, we described the detail steps to partition the volume into size-maximal boxes. The partition process is basically a bottom-up approach. The goal is to fit smooth region with larger boxes. To increase the flexibility on the shape of the boxes, we apply binary tree partition on the 1D data, instead of applying the octree partition on the 3D data. First, we partition the 1D lign into length-maximal dikes. Then, we go on to 2D to partition the 2D farm into area-maximal padis based on the 1D partition information. Finally, the partition is performed in 3D to find out the size-maximal boxes using the 2D partition information. We will continue to describe the generation of triangular mesh in the next chapter.

Chapter 4

Triangular Mesh Generation

Once the volume has been partitioned into size-maximal rectangular boxes, we can generate triangles within each box. Since the size of the rectangular box reflects the geometry complexity of the enclosed isosurface, hence the generated triangles will also adapt to the geometry. That is, larger triangles are generated to approximate smooth isosurface regions.

In this chapter, we will describe the mesh generation process in details. One important step to prevent the occurrence of gap is *information sharing* described in Section 4.1. After that, we can generate triangles for each box in Section 4.2. Section 4.3 discusses how to apply the original adaptive skeleton climbing algorithm, which only works for $N \times N \times N$ volume, to volume of any size. We will also describe how to generate the isosurface in multiresolution. Then, in Section 4.4, we discuss the techniques to speed up the algorithm. Section 4.5 shows the results and compares them with the classic marching cubes algorithm.

4.1 Sharing Information Between Highrices

At this moment, we can immediately generate triangles inside each xy-highrice with the padi layout on the surface on the xy-highrices. This will yield triangular mesh with crack, just like the cases of Shu *et al.* [SHU95] and Shekhar *et al.* [SHEK96], since the boxes may not be unit cube.

Figure 4-1(a) shows a large highrice next to a small one. The isosurface crosses the plane separating the two highrices (Figure 4-1(b)). If triangles are emitted for each highrice without the knowledge of their neighbors, gaps will appear in the generated triangular mesh. This is because the linear isolines generated on the highrice surfaces may not match each other geometrically (Figure 4-1(c)), even though they are topologically correct. To prevent this mismatch, information must be shared between adjacent highrices.

In our algorithm, the neighbor information can be shared by manipulating the basic data structures. Recall that we store the padi layout of each farm in layout arrays in Section 3.2.2. Layout arrays are variants of simple plot arrays. We can reuse these arrays with the new constraint that no plot may

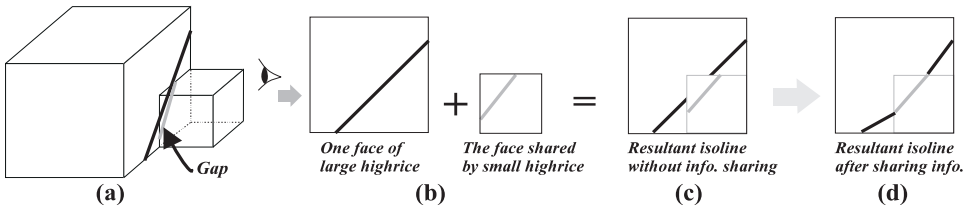


Figure 4-1: Sharing information between neighbor highrices.

cross the boundary of any face of any generated xy-highrice. That is, we store the 3D highrice layout in these arrays this time. Figure 4-2 shows the farm between the two highrices in Figure 4-1. The surface boundary of the larger highrice is shown as thick dark gray line in the farm of Figure 4-2(b), while that of the smaller highrice is shown as thick light gray line.

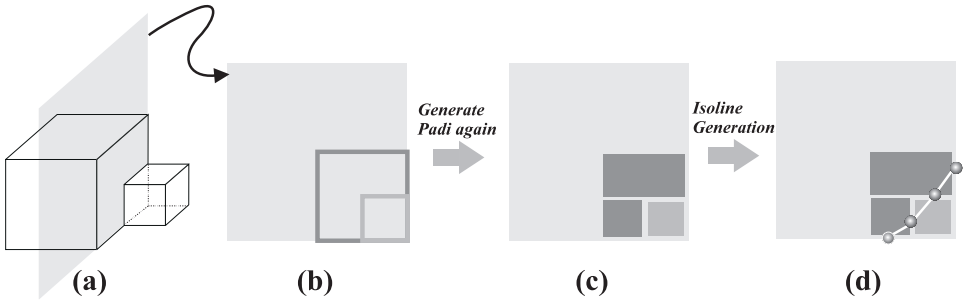


Figure 4-2: Once we reinitialize the layout arrays to store the 3D highrices' layout, `ASC2D` can be run to generate padis that fitted into the surface boundaries of both highrices.

Once the layout arrays are reinitialized, algorithm ASC2D is executed on them (instead of simple plot arrays) to give a new set of padis. Since the length-maximal plots represented by the layout arrays are not allowed to cross any boundary, the generated padis will also fit inside these boundaries. Figure 4-2(c) shows the generated padis for the discussed example. After generating iso-lines on each padi, three segments of iso-lines will be generated in the example (Figure 4-2(d)), therefore no gap will exist.

4.2 Extracting Triangles Within a Box

4.2.1 Generating the Edge Loops

Instead of thinking the padis are laid on the farm, they can also be regarded as padis laid on the six faces of each xy-highrice. Each face of the xy-highrice may contains more than one padis as in Figure 4-3.

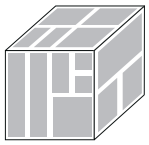


Figure 4-3: The six faces of a highrice are tiled with padis after information sharing.

To generate the isosurface, we first generate iso-lines on padis by looking up the 2D padi configuration table in Fig 3-15, and connect them to form closed edge loops (Figure 4-8(a)). Note that in our algorithm, we only need a 2D padi configuration table, no 3D voxel cube configuration table is needed.

Even though iso-lines (edges) are generated by looking up the configuration table, the edge loop is not yet available since the edges are not yet connected. Hence the next step is to connect the disjointed edges.

First, we need to label all unit dikes on the surfaces of the highrice (Figure 4-4). A highrice of size $p \times q \times r$ contains $l = 4(pq + qr + rp)$ distinct unit dikes. An edge can then be represented by an unordered pair (a, b), a and b are the ID of the unit dikes where the edge begins and ends respectively.

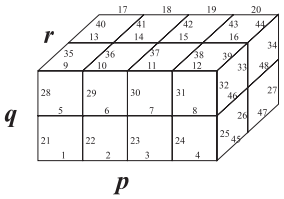


Figure 4-4: Each unit dikes on the surface of the highrice is assigned with a distinct label.

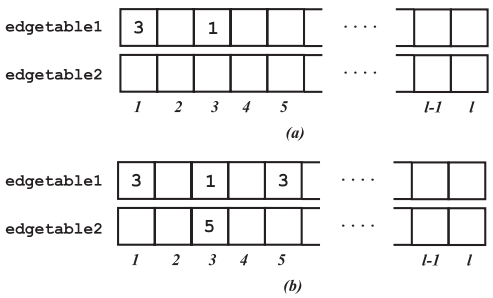


Figure 4-5: To connect the edges, two length-l edge tables are used.

Secondly, two length-l tables are used to connect the edges (Figure 4-5). For each unordered pair (a, b), fill the a^{th} entry with b and the b^{th} entry with a. For example, if there is an edge $(1, 3)$, the

tables will look like Figure 4-5(a). If the entry in the first table (`edgetable1`) is already occupied, we fill the corresponding entry in the second table (`edgetable2`). For instance, a new edge $(3, 5)$ is generated, the two tables will become Figure 4-5(b).

When all edges are entered into the tables. The edge loops can be retrieved from the table by the algorithm `GenLoop` in Figure 4-6. This algorithm contains a simple loop which steps through all nodes inside the edge loop. It works whenever the edge loop is closed and each node in the loop is connected by two edges. It can also handle multiple edge loops within the same highrice. The time complexity of this algorithm is obviously linear.

Input: Two filled edge tables.
Output: A sequence of unit dike ID representing the edge loop.
Algorithm:

```
current := ID of first non-empty entry in edgetable1
start := current
prev := -1    // invalid value
next := -1    // invalid value
while next ≠ start
    if edgetable[current] ≠ prev
        next := edgetable1[current]
    else
        next := edgetable2[current]
    output current
    prev := current
    current := next
```

Figure 4-6: Algorithm `GenLoop`.

4.2.2 Triangulating the Edge Loops

Given an edge loop consisting of several vertices v_i, we emit triangles using the algorithm `EmitTriangle` in Figure 4-9. In each iteration, three consecutive vertices v_i, v_{i+1} and v_{i+2} are selected and one triangle is generated (Figure 4-7). The vertex v_{i+1} is then removed from the edge loop. The algorithm continues until only two vertices are left.

An edge loop can be triangulated in multiple ways. Different sequences give triangular meshes with identical triangle counts, but with different geometry (Figure 4-8(b) and (c)). To generate a mesh that closely approximates the true isosurface, we make use of the gradient. As shown in algorithm `EmitTriangle` in Figure 4-9, we reject any triangle with planar normal vector $\vec{n_t}$ that largely deviates from the gradients $\vec{g_i}$ at three vertices. The deviation is measured by the dot product of $\vec{n_t}$ and $\vec{g_i}$. A threshold is used as a criteria. The threshold constraint will be relaxed if no triangle can be generated under the current constraint.

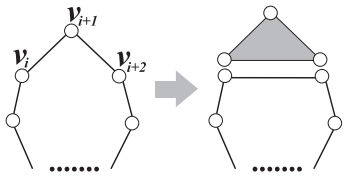


Figure 4-7: In each iteration, one triangle is emitted and one vertex is removed.

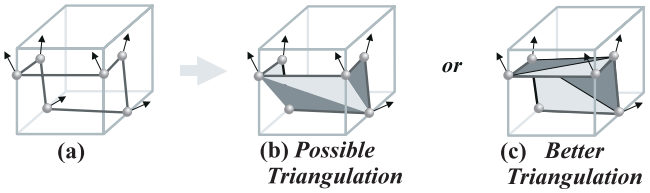


Figure 4-8: Triangulate the edge loop to emit triangles.

4.3 Extending to Arbitrary Volume

The proposed algorithm handles volume with the size of $(N+1) \times (N+1) \times (N+1)$, *i.e.* a cubic *block*. To handle volume with different size, we can simply tile the blocks to cover the whole volume and apply `ASC3D` on each block. Recall that gaps will appear if no information is shared between adjacent highrices. Similar cracks will appear if information is not shared between adjacent blocks. Unlike the case of variable-sized highrices, each block has the same size. This simplifies the process. To share information between blocks, we simply perform `MAX` operations on each pair of layout arrays on the surfaces (which are also farms) of two adjacent blocks. The simple `MAX` operations effectively find out the largest padis that fit the constraints. This information sharing process must be done just

---

Input: An edge loop and an initial deviation threshold T
Output: A triangular mesh.
Algorithm:
Put all vertices on the edge loop into an order preserving list.
While the list contains more than 2 vertices
  Pick three consecutive vertices (v_i, v_{i+1}, v_{i+2}) to form a triangle
  If the deviation is too large, $\vec{n_t} \cdot \vec{g_i} < T$ or $\vec{n_t} \cdot \vec{g}_{i+1} < T$
  or $\vec{n_t} \cdot \vec{g}_{i+2} < T$
    Reject this triangle
  Else
    Emit this triangle
    Remove vertex v_{i+1} from the list
  If no triangle can be generated above current threshold T
    Decrease the value of T

Figure 4-9: Algorithm `EmitTriangle`.

after the information sharing among highrices.

Up to this moment, we have not yet discussed the effect of using different values of N, *i.e.* the size of the block. The block size constrains the maximum size of the highrices. When the block size is small, say $N = 1$, the largest highrice contains $2 \times 2 \times 2$ voxels, *i.e.* same as standard marching cubes. When a larger block size is used, larger highrices are allowed to be generated, hence larger triangles. In other words, by controlling the value N, we can generate isosurfaces in multiple resolutions. Note that parameter N is an indirect control, the actual mesh generated will also depend on the geometry of the real isosurface. More triangles will still be generated if the isosurface geometry is complex. Figures 4-14 to 4-18 show the results of using different block sizes. From (a) to (b), the values of N are 1, 2, 4 and 8. As the block size increases, larger triangles are generated to approximate the smooth surface.

Unlike the triangle reduction algorithms [DEHA91, HOPP93, SCHR92, TURK92] which generate coarser mesh based on the high resolution mesh, the proposed approach generates coarser mesh directly from the original volume data. This ensures no distortion or error is introduced before the triangle reduction. More importantly, the proposed algorithm is an on-the-fly process which requires no time-consuming postprocessing triangle reduction. In fact, the algorithm produces coarser mesh in a smaller amount of time (see Table 4.1). Although our approach may not reduce triangles as much as mesh optimizer does, it is a cost effective method to significantly reduce triangles in a short period of time.

4.4 Speeding Up the Algorithm

4.4.1 Skipping Empty Blocks

In practice, there is no need to process every block of voxels. Since many blocks are empty, *i.e.* contains no isosurface, we can simply ignore them without performing the computational intensive merging processes. This can be done in the early stage of the algorithm. Once we have initialized the values in `occ[]` for each lign in the block, the emptiness of the block can be immediately identified. With this simple technique, the execution time of the algorithm can be reduced to about 25% of the original execution time. The actual speedup depends on the geometry complexity of the isosurface.

4.4.2 Indexing in Span Space

Skipping empty blocks can significantly reduce the computational time. However it still requires the initialization of the occupancy array (`occ[]`) for every block. A further speedup can be achieved by indexing the block in span space. Livnat *et al.* [LIVN96] indexed the voxel cubes using kd-tree [BENT75]. By indexing the minimum and maximum values among the eight voxels within the voxel cube, the non-empty voxel cubes can be rapidly located using the $O(\sqrt{c}+k)$ searching algorithm [LEE77], where c is total number of voxel cubes and k is the number of cells intersecting the isosurfaces.

In our case, indexing the unit voxel cubes is not enough since we need to locate non-empty blocks of various sizes ($(N+1)\times(N+1)\times(N+1)$). Moreover, the original method retrieves voxel cubes in an arbitrary order. That means, if two neighboring voxel cubes are intersected by the isosurface, the method cannot ensure these voxel cubes are retrieved one by one. The retrieval order is important in our case, since information sharing has to be done between neighboring blocks to prevent gaps. To perform information sharing efficiently, the non-empty blocks must be retrieved in a layer-by-layer fashion. Two more keys are added to construct a 4d-tree, in order to solve the mentioned problems. For each block, four keys are defined.

$$(V_{\min}, V_{\max}, N, L)$$

Key $V_{\min}$ and $V_{\max}$ are the minimum and maximum values of voxels within a block. Key N is the block size. With this key, we can store blocks of different sizes in a single kd-tree. When querying the blocks of specific size, the search key N must be set to the corresponding value. Key L is the number of the layer where the block is located (Figure 4-10). By querying the blocks in an increasing order of L, we can retrieve them in a layer-by-layer fashion. Hence, information sharing can be done without storing all the non-empty blocks in the memory.

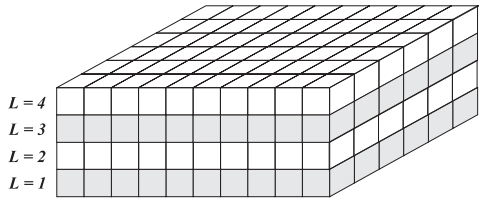


Figure 4-10: The blocks are retrieved in a layer-by-layer fashion.

4.5 Results

Table 4.1, Figures 4-11 and 4-12 quantify for various datasets the results of our implementation of adaptive skeleton climbing with four block sizes, $N = 1$, 2, 4 and 8. Triangle counts and CPU times on a SGI Onyx are compared with the Wyvill implementation [WYVI90] of marching cubes algorithm. The timing results shown Table 4.1 has been sped up by skipping the empty blocks. The kd-tree indexing is disabled since it is not fair to compare it with the original marching cubes algorithm which includes no indexing scheme. Figures 4-14 to 4-18 show the corresponding images. Only the meshes generated by our algorithms are shown. Gouraud shaded isosurfaces are overlaid with triangle edges for clarity.

The "knot" data sets are sampled from an algebraic function, at three resolutions, $64 \times 64 \times 64$, $128 \times 128 \times 128$ and $256 \times 256 \times 256$. Figure 4-14 shows the extracted isosurfaces from the $64 \times 64 \times 64$ volume in multiple resolutions. Volume "Mt. St. Helens" (Figure 4-15) is a landscape heightfield dataset. We also tested three medical computed tomography (CT) datasets, "Head1" (Figure 4-16), "Head2" (Figure 4-17) and "Arteries" (Figure 4-18). The data set "Arteries" contains no large smooth sheets, and its geometrical complexity inherently requires a finer mesh for topological correctness.

In general, as the block size N increases, both the CPU time (Figure 4-12) and the triangle count (Figure 4-11) decrease. There are about four to twenty-five times reduction in the triangle count. Figures 4-14 to 4-18 reveal little change in shape as the triangle count decreases. Note that in some cases increasing the block size N may slightly increase the triangle count when the complex geometry of the isosurface requires sufficient triangles to represent it. In three out of seven tested cases, the optimal block size (in terms of triangle count) is $N = 4$. Depend on the geometry complexity of the true isosurface, the optimal value may vary. From the experiments, the proposed algorithm is not efficient to generate the highest resolution mesh as the marching cubes algorithms do. However, it can generate coarser meshes in an amount of time comparable to that of marching cubes algorithms. In the test cases of "knot256" and "Mt. St. Helens", the running times of generating coarse meshes ($N = 8$) are even faster than that of the marching cubes.

Table 4.2 shows the reduction (in percentage) of execution time if kd-tree indexing is used to locate the non-empty blocks. There is about 10-60% reduction depending on the volume complexity. Figure 4-13 shows the same results graphically. From the graph, it shows that the reduction decreases as the block size increases. As the block size increases, more unit voxels will be involved in the merging process even though they are not intersected by the isosurface.

Data Set	ASC, $N = 1$	ASC, $N = 2$	ASC, $N = 4$	ASC, $N = 8$	MC
knot64	12,712△	3,682△	1,772△	2,054△	13,968△
$64 \times 64 \times 64$	8.16*sec.*	3.61*sec.*	2.59*sec.*	2.43*sec.*	1.79*sec.*
knot128	44,760△	13,088△	4,692△	3,918△	56,208△
$128 \times 128 \times 128$	61.52*sec.*	24.00*sec.*	15.61*sec.*	14.31*sec.*	12.81*sec.*
knot256	152,080△	48,562△	15,370△	8,829△	225,736△
$256 \times 256 \times 256$	470.95*sec.*	178.54*sec.*	105.31*sec.*	87.78*sec.*	94.62*sec.*
Mt. St. Helens	335,096△	119,045△	83,538△	92,740△	335,008△
$258 \times 258 \times 256$	495.17*sec.*	195.45*sec.*	137.45*sec.*	120.80*sec.*	219.87*sec.*
Head 1	580,771△	186,331△	136,909△	159,207△	592,368△
$256 \times 256 \times 113$	339.21*sec.*	138.59*sec.*	97.31*sec.*	100.55 *sec.*	61.91*sec.*
Head 2	98,397△	58,244△	60,885△	63,582△	95,362△
$128 \times 128 \times 57$	44.42*sec.*	20.93*sec.*	17.51*sec.*	19.27*sec.*	8.00*sec.*
Arteries	263,686△	131,769△	139,636△	149,251△	263,438△
$256 \times 256 \times 148$	311.97*sec.*	134.00*sec.*	103.68*sec.*	128.07*sec.*	56.09*sec.*

Table 4.1: Comparison of adaptive skeleton climbing, with block sizes $N = 1, 2, 4, 8$, and marching cubes, in terms of triangle number (△) and CPU time.

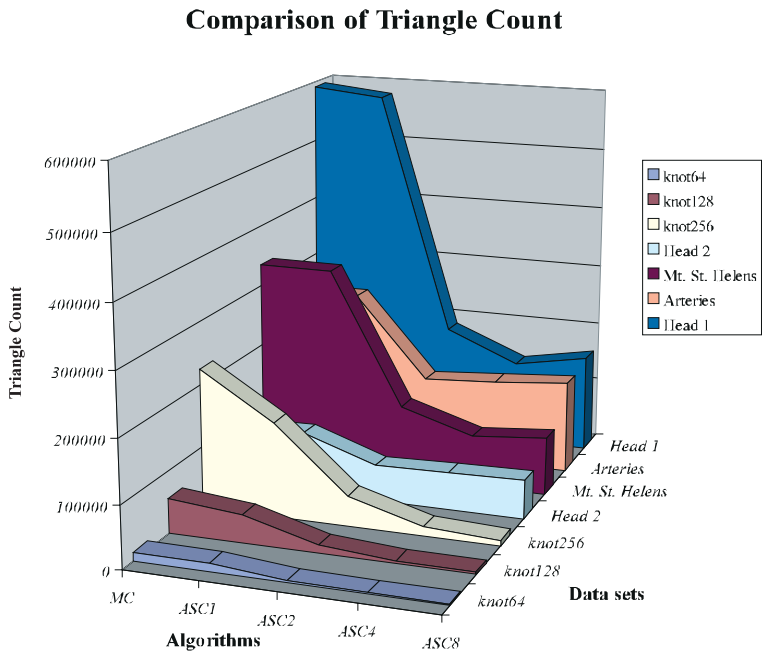


Figure 4-11: Graph of triangle count in Table 4.1.

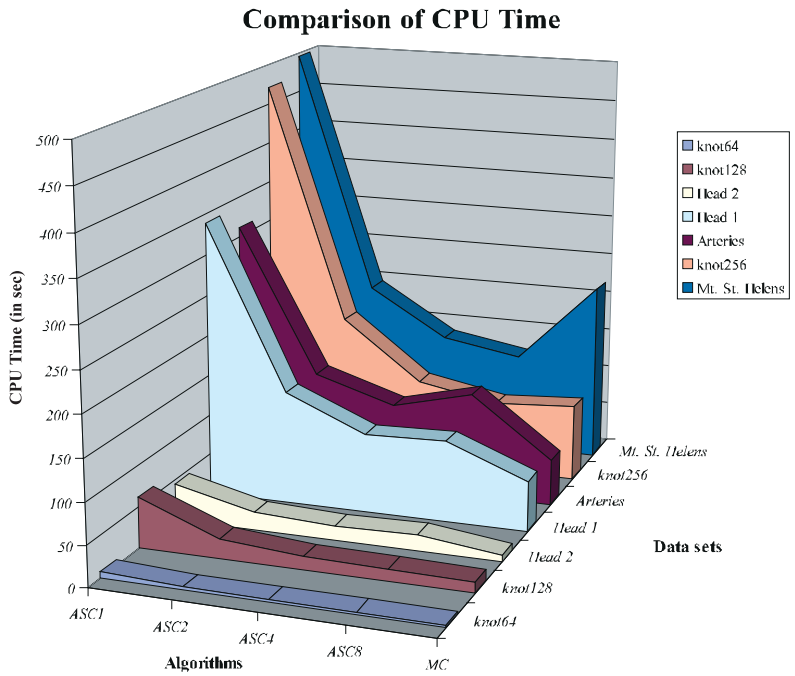


Figure 4-12: Graph of CPU time in Table 4.1.

Data Set	ASC, $N = 1$	ASC, $N = 2$	ASC, $N = 4$	ASC, $N = 8$
knot64	59%	53%	40%	30%
Head 1	46%	38%	29%	23%
Head 2	42%	32%	19%	13%
Arteries	67%	58%	40%	26%

Table 4.2: Reduction (in percentage) of execution time after indexing with the kd-tree.

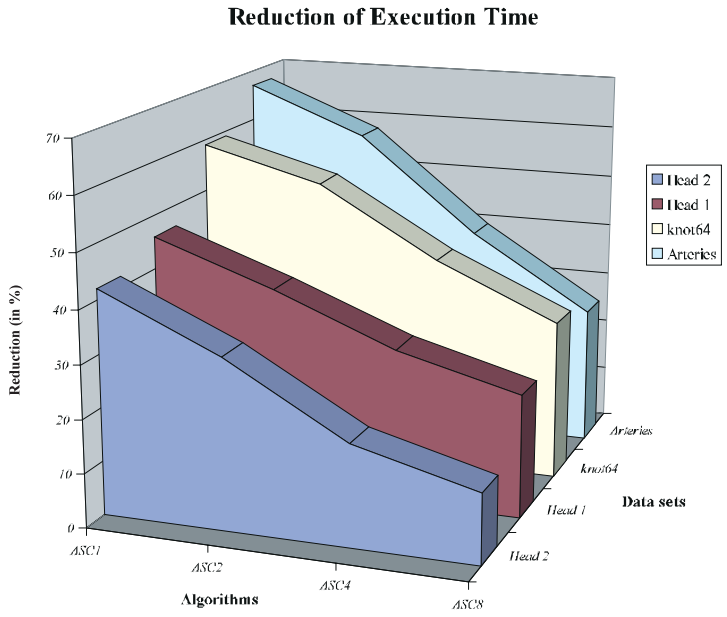


Figure 4-13: Reduction (in percentage) of execution time after indexing with the kd-tree. (Graphical presentation of Table 4.2.)

4.6 Summary & Discussions

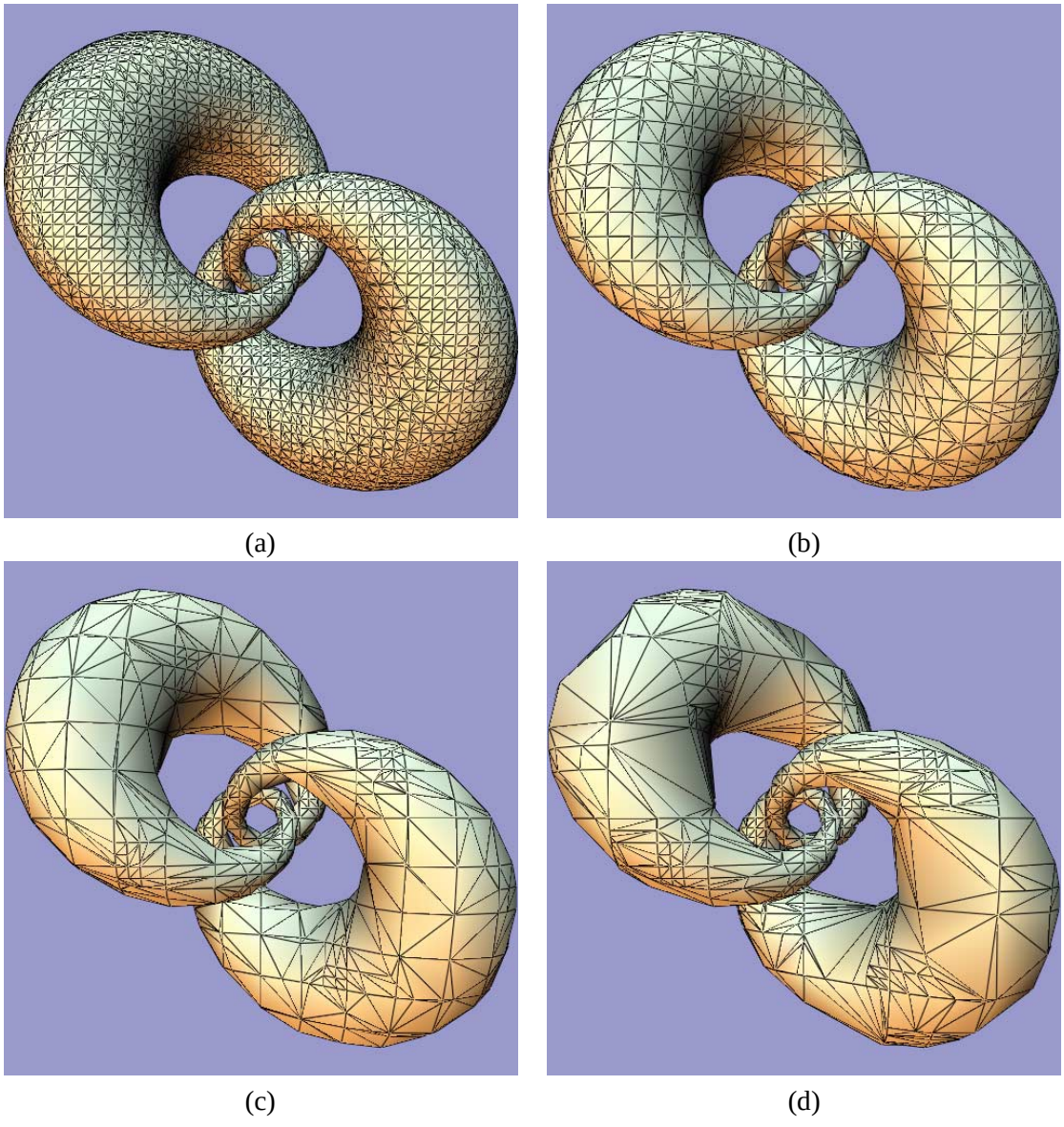
Adaptive skeleton climbing produces isosurfaces in times comparable to marching cubes [LORE87], with substantially fewer triangles, and without the gap-filling problems of adaptive marching cube methods [WILH92]. It directly uses the volume data and produces isosurface in multiple resolutions.

Although optimal coarse meshes can be found by postprocessing mesh reduction algorithm [HOPP93], significant computational time is needed. Moreover, the optimality of the mesh representation is not the main concern of the people such as surgeons. Instead, the speed of the algorithm, the accuracy and the triangle count of the generated meshes are more important to a surgeon who may have to try different threshold values to explore the tumor surfaces and interact with them. Our algorithm can be served for such purpose.

Since we use simple rectangular boxes instead of octree cubes, this approach provides more flexibility in partitioning the volume, hence captures more isosurface regions with simple geometry. It is faster than postprocessing mesh simplification methods [HOPP93, SCHR92, TURK92, DEHA91], though the mesh may not be optimally reduced. The proposed algorithm can serve as a companion to the mesh optimizer, since the coarse mesh produced can be a better initial guess for the optimizer.

Adaptive skeleton climbing is a fast heuristic algorithm, rather than a path to a strict optimum.

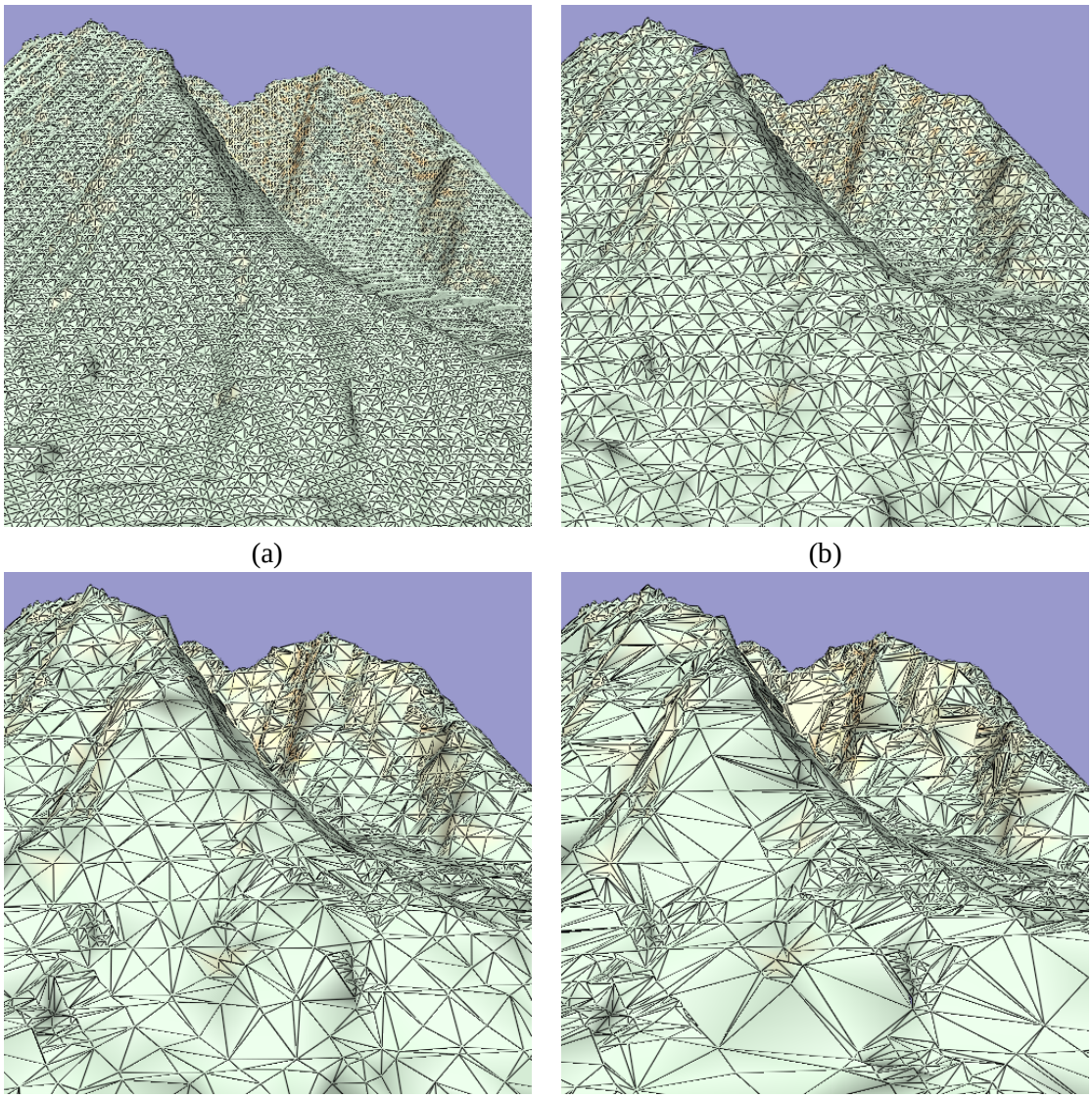
Currently, the proposed algorithm subdivides the volume into sub-volumes based on the simplicity criteria suggested in Section 3.1. It is a topological criteria instead of a geometrical criteria. This leads to a difficulty in measuring and controlling the geometric error between the generated mesh and the true isosurface being represented. To use the generated meshes in the level of detail applications, an error measurement is needed. This can be done if the finest mesh is treated as the "true" isosurface. Then the geometric error can be estimated by comparing the generated mesh with the "true" mesh. A more direct approach to generate mesh with desired geometric error is to modify the subdivision criteria in the adaptive skeleton climbing to a geometrical criteria.



(a) (b)

(c) (d)

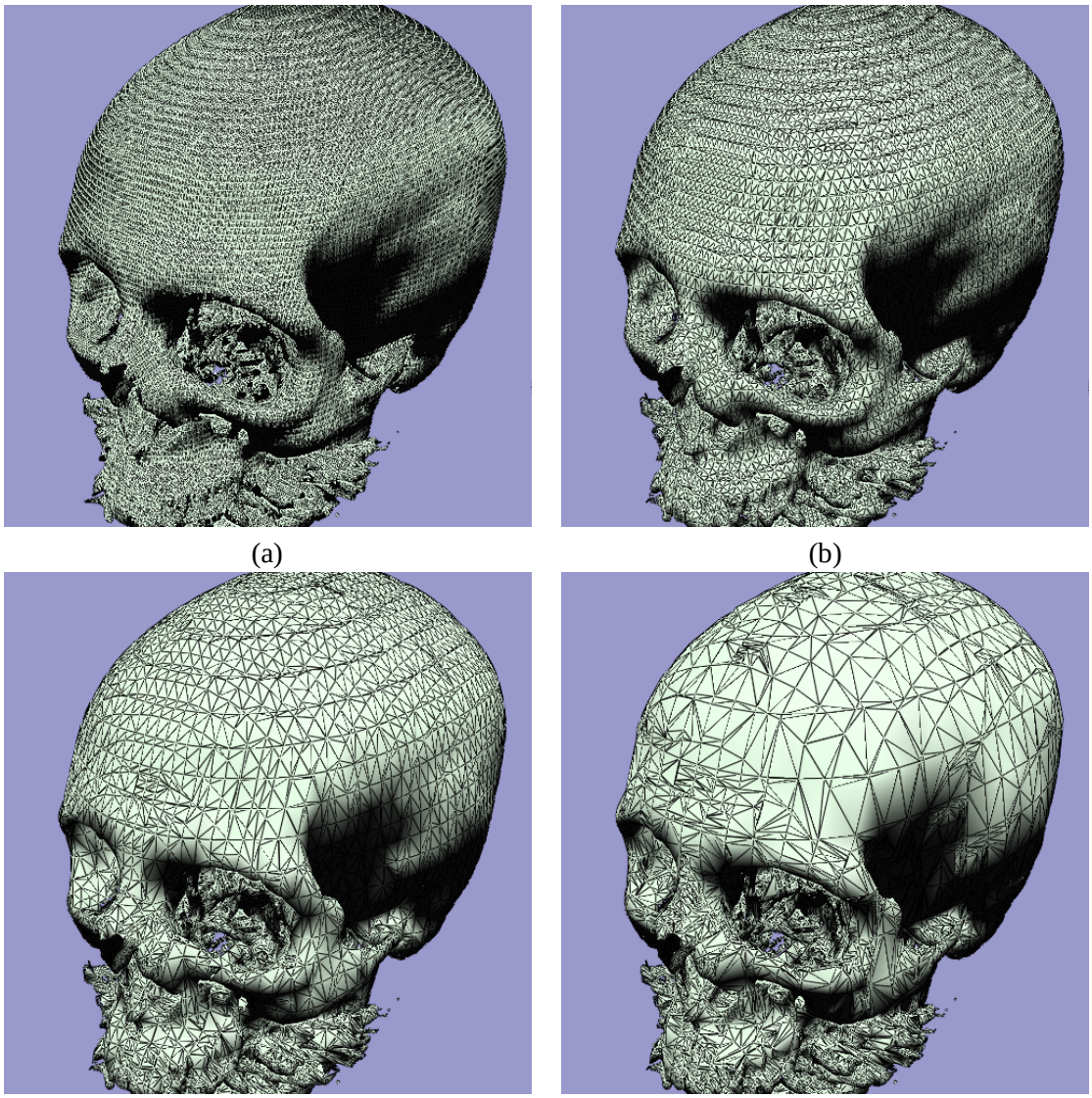
Figure 4-14: Visual comparison of the effects of block size for an algebraic surface, ”knot64”. (a) N=1, (b) N=2, (c) N=4, (d) N=8.



(a) (b)

(c) (d)

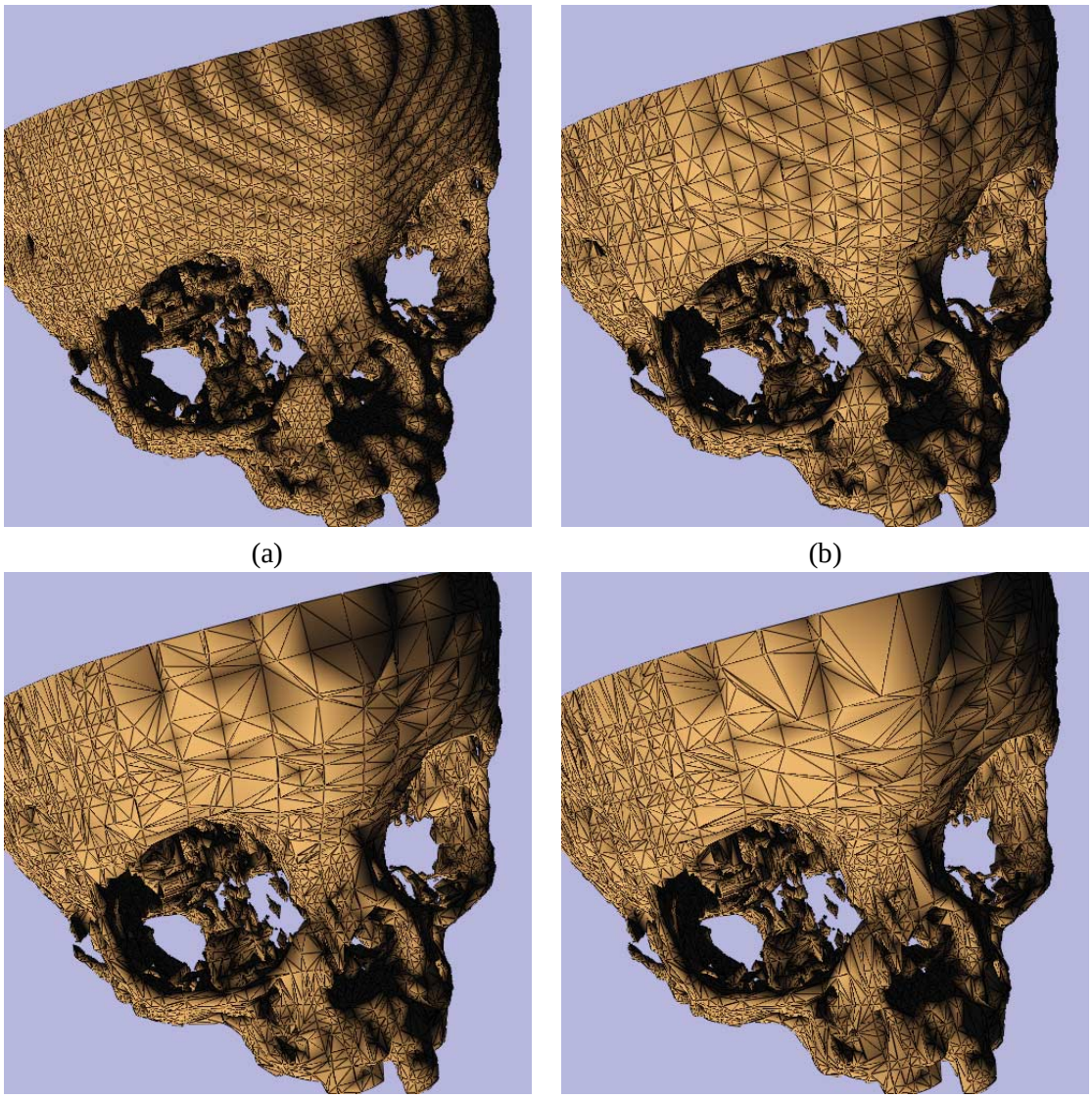
Figure 4-15: Visual comparison of the effects of block size for landscape data ”Mt. St. Helens”. (a) N=1, (b) N=2, (c) N=4, (d) N=8.



(a) (b)

(c) (d)

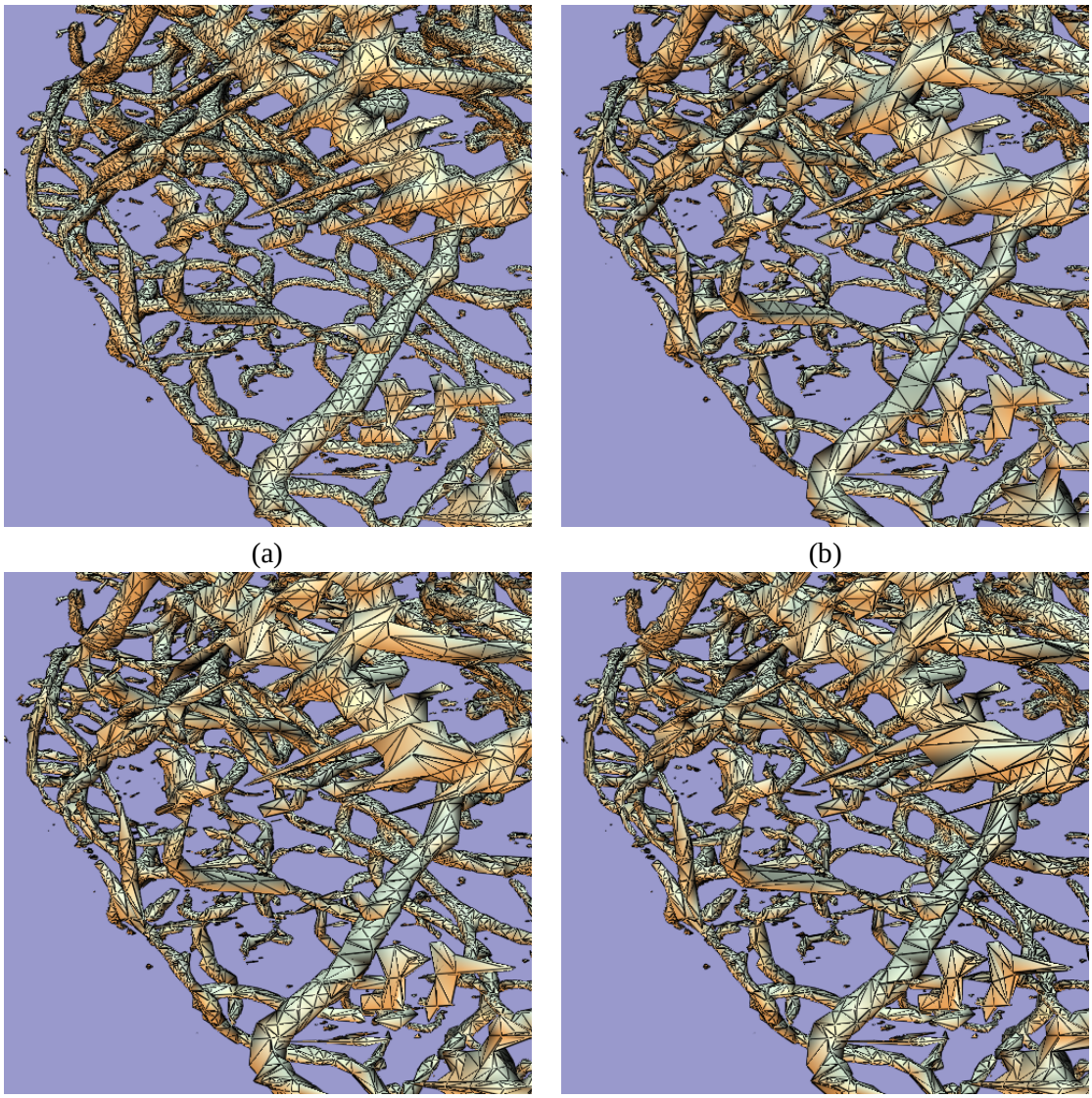
Figure 4-16: Visual comparison of the effects of block size for a CT data of human head, ”Head1”. (a) N=1, (b) N=2, (c) N=4, (d) N=8.



(a) (b)

(c) (d)

Figure 4-17: Visual comparison of the effects of block size for another CT data of human head, ”Head2”. (a) N=1, (b) N=2, (c) N=4, (d) N=8.



(a) (b)

(c) (d)

Figure 4-18: Visual comparison of the effects of block size, for a CT data of blood vessels in the head, ”Arteries”. (a) N=1, (b) N=2, (c) N=4, (d) N=8.

Part II

Image-based Approaches

Chapter 5

Image-based Approaches

In Part I of this thesis, we introduce a geometry-based simplification algorithm which generates simplified mesh directly from the volume data. Take example "Head 1" from Table 4.1. The coarsest resolution of the model requires 159,207 triangles to represent. This is only the number of triangles for one object. If the scene contains thousands of such objects, real time rendering will be impossible given the current technology. Using the view-dependent simplification algorithms can improve the rendering speed by displaying coarser mesh when it is far away or occluded. However, if the scene contains thousands of semi-transparent skull models overlapping each other and it is viewed from a close distance, any view-dependent algorithm is helpless in this case.

There is an increase in believing that geometry-based approach will not achieve the goal of real-time display of arbitrarily complex scene. The time complexity of any geometry-based renderer is a function of n, the number of primitives in the scene, which may be arbitrarily large and slows down the rendering. The solution to this problem is to design a renderer with a time complexity independent of n. This leads to the research of *image-based modeling and rendering*, modeling and rendering using images only.

Besides time complexity, another major reason for the emergence of image-based computer graphics is the difficulty in modeling real world scenes/objects. As mentioned in Chapter 1 and illustrated by Figure 1-2, image-based computer graphics synthesize desired images by warping and compositing the reference images. No explicit geometry model exists during the synthesis. Modeling becomes a process of taking photographs. Taking photographs of real world scene is usually easier than constructing the geometry representation.

5.1 Motivation

Previous work (described in Section 5.2) mainly focus on finding the correct view given a set of reference images. The illumination of the scene captured by the reference images is commonly assumed unchanged and carefully designed. Hence the illumination in the final desired image will also be

fixed. In other words, we have a renderer that supports the change of viewpoint, viewing direction and field of view, but not the change of illumination. Apparently, such renderer is *incomplete* for the computer graphics usage. Our motivation is based on the need of a *complete* image-based renderer.

There are a few work done on the illumination of images. Unfortunately, all of them has some restrictions or limitations. One common limitation is that all object surfaces visible in the image must be *Lambertian*. This is a very strong assumption which is impractical for real world image containing specular surfaces. Another limitation is the *viewpoint is commonly assumed fixed*. That is, although we can change the lighting, we cannot change the viewing direction. One of the previous work *restricts the type of illumination* to only outdoor illumination. Therefore it can only render the scene under the sunlight. One more problem is that some of them *do not allow controllable illumination* even though they can change the illumination. We will discuss it in detail in the next section. Generally speaking, all previous work are not general enough.

In this thesis, we will introduce a completely new and *general* concept that allows the re-rendering of arbitrary image under any illumination. There is *no assumption on the surface properties* seen in the image. There is *no restriction on the type of illumination*. At the same time, the *viewpoint of the image can be changed* as well. One more importance is that our approach offers a *controllable illumination*.

To calculate the reflected radiance outgoing from a surface element, we need to know the reflectance of that surface element. A general description of reflectance is the *bidirectional reflectance distribution function* (BRDF). It is actually a table of reflectance values recording the reflectances of the surface when the surface is viewed from different direction and illuminated by a light source from various direction.

Without the geometry, there is no way to measure or specify the BRDF of a surface element seen in the image. To allow the illumination to be done in image-based systems, we introduce the concept of *measuring the apparent BRDF of a pixel* by treating a pixel as an ordinary surface element. We will show later in this thesis how these pixel BRDF can be used in re-rendering of any image-based scene under any lighting condition. A series of techniques will also be introduced to make the idea practical.

5.2 Related Work

Previous work can be roughly classified into two main streams. The first stream focuses on determining the correct perspective view. The second stream of research focuses on re-rendering the scene under different illumination.

5.2.1 Finding the Correct View

Images have long been used in computer graphics as an approximation of surface details. This application is commonly known as texture mapping. Its simplicity and efficiency make it popular. The basic idea of texture mapping is to modulate various surface properties of the object according to the texture image. Many variants of texture mapping are proposed during the past three decades. The original texture mapping [CATM74] changes only the color parameter of the surface. If the specular reflection is changed according to the texture, it is known as environment mapping [BLIN76, GREE86a]. If the normal vector is perturbed according to the texture, it is known as bump mapping [BLIN78b]. Others change the glossiness coefficient [BLIN78a], transparency [GARD84, GARD85], diffused reflection [MILL84], surface displacement [COOK84a], shadows [COOK84a, REEV87, SEGA92], and local coordinate system [KAJI85, CABR87]. Heckbert [HECK86] provided a comprehensive survey of various usage of texture mapping. Most of texture mapping techniques treat images as an approximation of small geometry details. Images are frequently used as a component of the geometry models. Since the small geometry details being approximated by the texture image are in microscopic scale, they can be regarded as fixed even the viewpoint changes.

Recently, researchers generalized the idea and began to use images to approximate larger geometry. In other words, images are used to model the whole object, not just a component. To do so, one major issue has to be solved, *finding the correct images when the viewpoint changes*. Foley *et al.* [FOLE90] developed a system which can rotate raytraced voxel data interactively by interpolating images captured from certain viewing direction. The interpolation method used is not physically correct. Chen and Williams [CHEN93] interpolated views by modeling pixel movement, resulting in physically correct interpolation. Missing pixels due to occlusion are filled by partial rendering. Max and Ohsaki [MAX95] used a similar approach in modeling trees.

Later, Chen [CHEN95a] described an image-based rendering system, QuickTime VR, which generates perspective views from cylindrical panoramic image data by warping [CHEN95b]. Note that the panorama is actually the texture image used in previous environment mapping research [BLIN76,

GREE86a]. The perspective view can also be achieved by drawing a sphere from inside with the environment texture mapped onto the sphere. McMillan and Bishop [MCMI95] also used panoramic image as the fundamental images in their image-based renderer. They also mentioned that image-based rendering is a problem of sampling and manipulating the plenoptic function [ADEL91]. A method to sample and reconstruct this plenoptic function is proposed. Faugeras and Robert [FAUG93] applied the epipolar geometry to reconstruct the desired image using only a few reference images.

Image morphing [BEIE92] can be classified as a special type of image-based rendering. The major goal is to find a smooth interpolation from one image to another one. The interpolated images need not be physically correct. Interpolation is done by first warping the shape of object in both the source and target images. Then these two warped images are cross-dissolved to produce the desired image. If extra geometric information (such as camera parameters) can be determined, the morphing can produce physically correct images. Seitz [SEIT96] extracted the camera information from the reference images and then used it to produce morphed frames which are physically correct.

Levoy and Hanrahan [LEVO96] and Gortler *et al.* [GORT96, GU97] reduced the 5D plenoptic function to a 4D light field or Lumigraph. They used two planes to parameterize any ray passing the volume (light slab) enclosed by these two planes. The light field or Lumigraph is actually a table of radiances along the rays passing through the light slab. The table is hence indexed by four parameters (each plane requires two parameters to address a position on the plane). This simplification allows the view interpolation to be done by standard texture mapping techniques, which can be further accelerated by hardware. This two-plane organization will be described in more detail in Section 8.1.1. Instead of two-plane structure, Ihm *et al.* [IHM97] used spherical structure to record the radiances.

Debevec *et al.* [DEBE96] used a hybrid approach (both geometry and image) to model architectural objects. Geometry representation is used to approximate larger structure while image is used to approximate smaller geometry. This approach is very similar to early texture mapping. One unique feature of their work is that the geometry surface is not only mapped with one single texture image, but with multiple texture images captured from different viewpoints. During rendering, the textures will be blended to give a smooth realistic 3D rotation. A similar method is also proposed by Pulli *et al.* [PULL97].

Recently, animated image-based objects are developed by Live Picture [LIVE97].

5.2.2 Re-rendering Under Different Illumination

Another stream of research focuses on re-rendering the image under different illuminations. Haeberli [HAEB92] re-rendered the scene using simple superposition property. However, the direction, the type and the number of the light sources are limited to the lighting setup during the image capturing. Nimeroff *et al.* [NIME94] efficiently re-rendered the scene under various natural illumination (overcast or clear skylight) with the knowledge of the empirical formulæ that model the skylight in mind. Hence the illumination in the desired image is restricted to the outdoor illumination, *i.e.* illuminated by the sunlight only. Moreover, the viewpoint is always fixed.

Belhumeur and Kriegman [BELH96, ZHAN98] used singular value decomposition to extract a set of basis images from the input reference images. The desired image can be synthesized by linear combination of these basis images given a set of coefficients. In other words, the illumination is changed through controlling the values of the coefficients. Since there is no intuitive relationship between the values of the coefficients and the direction of the light source, they cannot control the direction of the light source in the desired image. That is, the illumination is *uncontrollable*. Moreover, they assumed the objects in the scene must be *Lambertian*. This is a very strong assumption and nearly impractical for real world images. In Belhumeur and Kriegman's work, the viewpoint is also assumed fixed.

Seitz and Kutulakos [SEIT98] proposed an interesting "image-based" editing framework. The algorithm first constructs an intermediate voxel data structure (geometry representation) using the voxel-coloring technique [SEIT97]. Then the voxel data is used for editing. With the voxel data, they can re-render the scene under a different illumination. However, they still have a Lambertian surface assumption during the voxel-coloring process. Hence, the framework only work for limited surface types. Strictly speaking, their framework involves a reconstruction phase, voxel-coloring, which reconstructs an intermediate geometry representation. Therefore, the rendering complexity will still depend on the complexity of the reconstructed scene, in this case, it is the resolution of the voxel representation.

5.3 Summary

In general, the first stream of previous work focuses only on the correct view synthesis and neglects the illumination capability. On the other hand, the work in the second stream is not general enough to allow re-rendering of arbitrary images under arbitrary lighting condition. More importantly, the

illumination is not controllable. Starting from the next chapter, we will introduce the general concept of measuring apparent BRDF of pixel. This allows us to re-render arbitrary images under arbitrary, controllable illumination.

Chapter 6

The Plenoptic Models

In traditional geometry-based computer graphics, the scene is first modeled as a set of geometrical entities. Then a physical simulation of light propagation takes place to approximate the image of the scene formed on our retinas. Hence the fundamental computation model of geometry-based computer graphics is a physical simulation. It allows us to evaluate the correctness of the modeling and rendering techniques.

For image-based computer graphics, a fundamental computation model is also needed to allow us to evaluate the usefulness of various image-based techniques and to develop new techniques based on it. McMillan [MCMI97] suggested *plenoptic function* [ADEL91] to serve as the fundamental model.

In this chapter, we will introduce the plenoptic function in details. By comparing its capability with the physical simulation in the traditional geometry-based computer graphics, we demonstrate the generality of this model.

The plenoptic function is first proposed as a model for evaluating various human vision model, not for computer graphics. Although the original formulation is very general, it is not convenient to express one crucial factor in traditional computer graphics, namely *illumination*. In this chapter, we propose an extended formulation of plenoptic function which includes illumination as well.

6.1 The Plenoptic Function

Adelson and Bergen [ADEL91] proposed a seven-dimension function known as *plenoptic function* [1]. *Plenoptic* is the combination of a Latin root *plenus* means plenty or complete, and the word *optic*. It describes the irradiance from any direction $\vec{V}$ arriving at any point $\dot{E}$ in space, at any time t and over any range of wavelengths λ. They formulate the function as follows,

$$I = \mathrm{P}(\theta_v, \phi_v, E_x, E_y, E_z, t, \lambda), \tag{6.1}$$

[1] A similar terminology known as *light field* was coined by A. Gershun [GERS39] to describe the radiometry properties of light in a space.

or in the short form,

$$I = \mathrm{P}(\vec{V}, \dot{E}, t, \lambda), \tag{6.2}$$

where I is the irradiance,

$\dot{E}$ = (E_x, E_y, E_z) is the position of the center of projection or the eye,

$\vec{V}$ = ($\sin\theta_v \cos\phi_v$, $\cos\theta_v$, $\sin\theta_v \sin\phi_v$) specifies the viewing direction originated from the eye,

t is the time parameter.

Basically, the function is formulated to mimic the idealized human eye. This idealized eye can be placed at any point $\dot{E}$ in the three dimensional Euclidean space. We then can place the reference axes at this position to define the coordinate system with the idealized eye being the origin. In graphics terminology, the space associated with this coordinate system is known as the *eye space*. A viewing direction $\vec{V}$ originated from the eye $\dot{E}$ can be specified by an azimuth angle ϕ_v and a zenith [2] angle θ_v, measured from reference axes $\hat{i}$ and $\hat{j}$ respectively. Figure 6-1 shows the geometric relationship between these parameters.

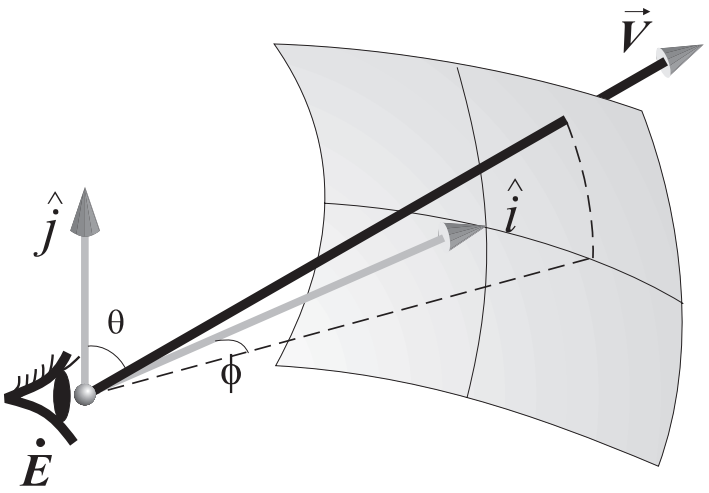


Figure 6-1: Geometric elements of the plenoptic function.

Two parameters are not shown in figure 6-1. Parameter λ specifies the wavelength of the light while parameter t specifies the time. In most computer graphics applications, we are only interested

[2] Note in Adelson and Bergen's original formulation [ADEL91], they use the elevation angle instead of the zenith angle. There is no functional difference between these two types of angle specification. The reason we use zenith angle in this thesis is because it is more convenient when discussing computer graphics related problems, since zenith angle is conventionally used in radiosity literatures [COHE93, SILL94].

in sampling color according to a particular color space, such as red, green and blue (RGB). Others applications such as color publishing may interest in sampling at four wavelengths (CYMK). For an in-depth discussion of color theory in computer graphics, readers are referred to [HALL89]. The time parameter t actually models all other unmentioned factors such as the change of the scene or the change of illumination. When t is constant, the scene and illumination are fixed. Theoretically, the plenoptic function can be continuous over the range of all its parameters.

Geometrically, the parameters (E_x, E_y, E_z) define the Euclidean position of the center of projection. Parameters (θ_v, ϕ_v) define a ray originated from the center of projection. The geometric structure of this ray space is equivalent to a unit sphere. Figure 6-2 shows the space graphically. The complete plenoptic function can be imagined as infinite number of such sphere placing all over the Euclidean space. Each sphere fires infinite number of rays from the center of projection in all directions. From now on, let's call the sphere as *plenoptic sphere*.

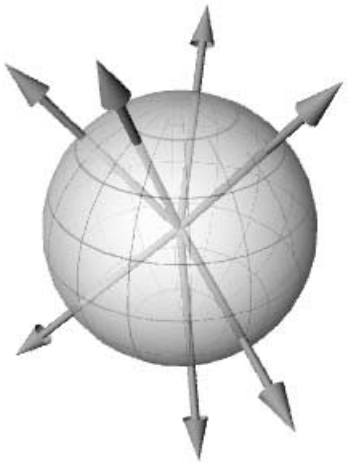


Figure 6-2: The ray space.

Finding the complete plenoptic function is not possible since it is defined all over a continuous parameter space. However, we can sample those regions that we are interested. Basically, finding the plenoptic function is a problem of *sampling*.

6.2 Subsets of Plenoptic Function

A digital image is a set of adjoined pixels, usually (not always) organized in rectangular form. The value of each pixel is dependent on the type of projection used during the scene capturing. Hence

a digital image representing a captured scene is not just a set of adjoined pixels, but also includes a projection. The most common type of projection is perspective projection since it mimics the projection that takes place in the human eyes. Another useful projection type is orthogonal or parallel projection which is commonly used in engineering drawing. Other types of projection like fish-eye and panoramic projections are also used for specific applications. In this section, we will demonstrate that an image with any type of projection can be regarded as a partial sampling of the complete continuous plenoptic function.

6.2.1 Perspective Projected Images

The most popular type of projection used in image capture is perspective projection. Its incorporation of foreshortening allows human to perceive depth in a two dimensional image. To specify a perspective projection, we need a *center of projection* and *projection plane*. The center of projection is simply a position $\dot{C}$ in the Euclidean space. The projection plane can be defined by the viewing vector $\hat{D}$, the upward vector $\hat{U}$, the vertical α_v and the horizontal α_h field of view angles.

By fixing the parameter $\dot{E}$ of the plenoptic function at $\dot{C}$ and restricting viewing direction (θ_v,ϕ_v) to be within the viewing frustum defined by the perspective projection, we have a partial plenoptic function that is equivalent to a perspective projected image.

$$\dot{E} = \dot{C}, \quad \hat{i} = \hat{D}, \quad \hat{j} = \hat{U},$$

$$-\frac{\alpha_h}{2} \leq \phi_v \leq \frac{\alpha_h}{2}, \quad -\frac{\alpha_v}{2} \leq \frac{\pi}{2} - \theta_v \leq \frac{\alpha_v}{2}.$$

Figure 6-3 illustrates the point graphically. The region with noisy pattern on the plenoptic sphere contains the irradiance values that can be represented by a perspective projected image. Hence a perspective projected image is only a subset of the plenoptic function.

6.2.2 Parallel Projected Images

Another useful projection is parallel projection or orthogonal projection which is often used in engineering drawing as it preserves the length. Parallel projection is specified by a rectangular viewing box defined by two extrema points ($x_{\min}$, $y_{\min}$, $z_{\min}$) and ($x_{\max}$, $y_{\max}$, $z_{\max}$) as in Figure 6-4. One surface of the box is defined as the projection plane. To generate the image, all points inside the viewing box are projected onto this plane along the normal vector $\hat{D}$ of projection plane.

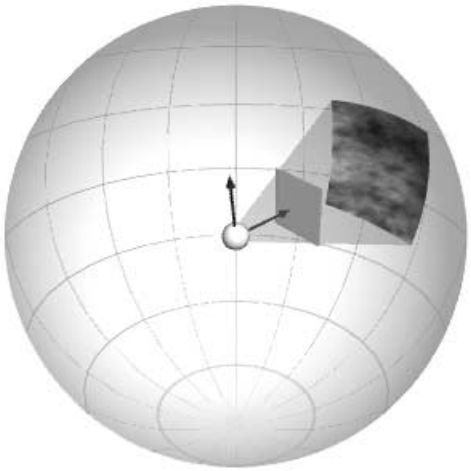


Figure 6-3: The set of irradiance values represented by a perspective projected image.

By fixing the viewing direction (θ_v, ϕ_v) of the plenoptic function along the projection direction and restricting the center of projection $\dot{E}$ on the projection plane, we get a partial plenoptic function that can represent any irradiance values shown in a parallel projected image.

$$\hat{i} = \hat{D}, \quad \hat{j} = \hat{U}, \quad \theta_v = \frac{\pi}{2}, \quad \phi_v = 0, \quad \hat{D} \cdot \dot{E} = 0,$$

$$E_x = x_{\min}, \quad y_{\min} \leq E_y \leq y_{\max}, \quad z_{\min} \leq E_z \leq z_{\max}.$$

Figure 6-4 illustrates the concept graphically. Each sphere in the image represents one instance of the plenoptic sphere at different location $\dot{E}$. Once again we show that the parallel projected image is only a subset of plenoptic function.

6.2.3 Panoramic Images

Panoramic image is first used in computer graphics as the texture map used in environment mapping [BLIN76, GREE86b]. Recently, Chen [CHEN95a] introduced a commercial image-based renderer which efficiently warps [CHEN95b] the cylindrical panoramic image to perspective image. Due to wide availability of these panoramic image viewers, such as QuickTime VR [CHEN95a] and RealSpace VR [LIVE97], there is a growing need of producing panoramic images [MCMI95, SZEL97]. Most currently available panorama are in cylindrical form (the central cylinder in Figure 6-5(a)) since it can easily unfolded to a rectangular image. On the other hand, spherical panorama (the central sphere in Figure 6-5(b)) requires special handling and more computation.

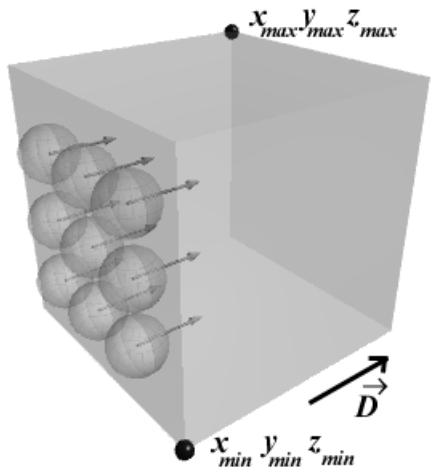


Figure 6-4: Parallel projected image as subset of plenoptic function.

Panoramic image is also a subset of plenoptic function. By fixing the parameters $\dot{E}$ at the center of projection of the panorama, this partial plenoptic function can represent any irradiance values shown in the spherical panoramic image (Figure 6-5(b)).

$$\dot{E} = \dot{C}.$$

For cylindrical panorama, we can further restrict the viewing direction parameter θ_v to be within the vertical field of view (α_v) and align $\hat{j}$ with the cylinder axis $\hat{U}$.

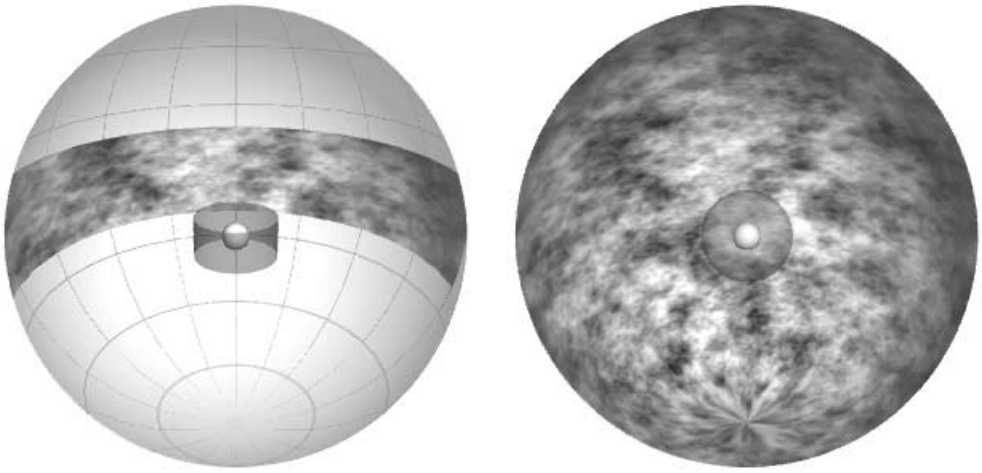
$$\dot{E} = \dot{C}, \quad \hat{j} = \hat{U},$$

$$-\frac{\alpha_v}{2} \leq \frac{\pi}{2} - \theta_v \leq \frac{\alpha_v}{2}.$$

The region with noisy pattern in Figure 6-5(a) shows the subset of plenoptic function that is represented by a cylindrical panorama. Spherical panorama can be represented by a plenoptic sphere (Figure 6-5(b)).

6.2.4 Images With Any Type of Projection

No matter what kind of projection is used during the scene capture, the irradiance should be recorded by shooting a *ray* from the center of projection or along a given projection direction into the scene. Therefore if the parameter space of the plenoptic function can express any ray in the space, the plenoptic function should be able to capture any irradiance coming along that ray. Hence an image



(a) (b)

Figure 6-5: Subsets of plenoptic function represented by (a) cylindrical panorama and (b) spherical panorama.

with any type of projection should be a subset of the plenoptic function. Any ray $\vec{R}$ can be represented in a parametric form composing of an origin $\dot{C}$ and a unit directional vector $\hat{D}$.

$$\vec{R} = \dot{C} + t_o \hat{D},$$

where $\dot{C}$ is the origin,

$\hat{D}$ is the ray direction, normalized,

t_o is a parameter.

By fixing the parameter $\dot{E}$ of plenoptic function at $\dot{C}$ and the direction parameters (θ_v,ϕ_v) along the direction $\hat{D}$, the plenoptic function is able to capture irradiance along any ray $\vec{R}$. To simplify the formulation, we align the reference axis with global coordinate system. Hence, direction $\theta_v = 0$ aligns with y axis and direction $\phi_v = 0$ aligns with x axis. Figure 6-6 illustrates the idea graphically.

$$\dot{E} = \dot{C},$$

$$\vec{V} = \hat{D}.$$

6.3 Comparing with Geometry-based Computer Graphics

In the last section, we show that images captured by any kind of projection are only subsets of the plenoptic function. Now we push a little bit further to see whether image-based computer graphics

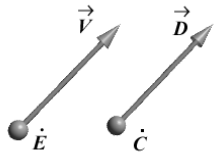


Figure 6-6: Parameters of plenoptic function can express any ray.

based on plenoptic function can perform all the functions that traditional geometry-based computer graphics can do. To simplify the discussion, we assume the geometry-based systems use only perspective projection.

6.3.1 Panning, Tilting and Zooming

When the center of projection is unchanged, geometry-based camera model can perform three basic operations, namely panning, tilting and zooming. To do panning in image-based model, we can simply adjust the range of parameter ϕ_v. Figure 6-7 shows the original (a) and the changed (b) regions in the plenoptic function when panning takes place. By gradually changing the value of the panning parameter δ in the following inequality, a smooth panning can be performed.

$$-\frac{\alpha_h}{2} + \delta_p \leq \phi_v \leq \frac{\alpha_h}{2} + \delta_p,$$

where α_h is the horizontal field of view,

δ_p is the parameter to pan,

ϕ_v is azimuth angle in the plenoptic function.

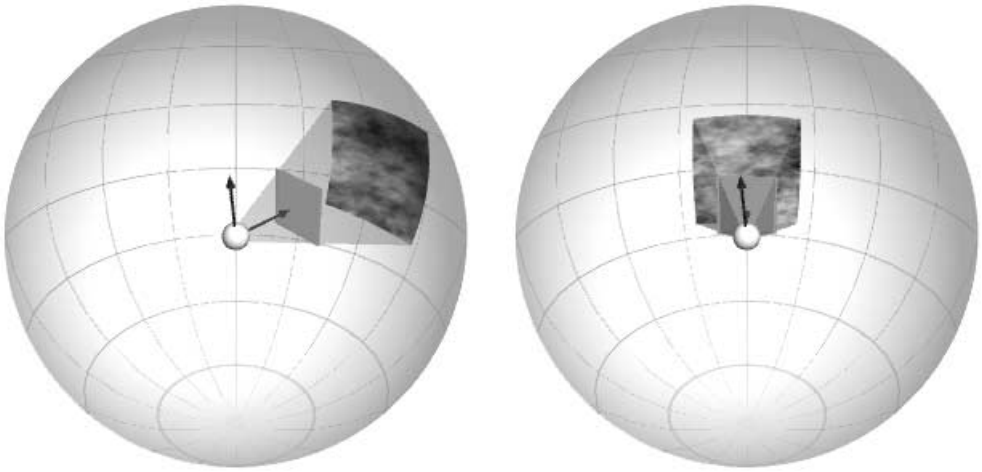
Similarly, tilting can be done by controlling the range of parameter θ_v using the following inequality. Figure 6-8 shows the regions of plenoptic function affected before (a) and after (b) tilting.

$$-\frac{\alpha_v}{2} + \delta_t \leq \frac{\pi}{2} - \theta_v \leq \frac{\alpha_v}{2} + \delta_t,$$

where α_v is the vertical field of view,

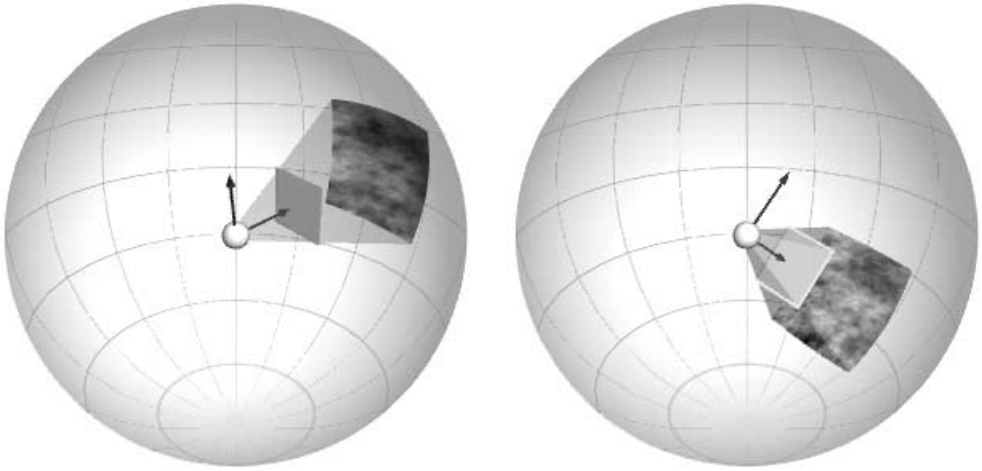
δ_t is the parameter to tilt,

θ_v is zenith angle in the plenoptic function.



(a) (b)

Figure 6-7: Panning in the plenoptic model. (a) Original (b) After panning.



(a) (b)

Figure 6-8: Tilting in the plenoptic model. (a) Original (b) After tilting.

Zooming can also be done by controlling the ranges of both the parameters θ_v and ϕ_v. By increasing the value of the zooming parameter δ_z, we can perform zoom out. On the other hand, decreasing the value of δ_z allows us to zoom in. Figure 6-9 shows the various regions covered in the plenoptic sphere when zooming in and out. Zooming in and out is easily controlled by the following two inequalities.

$$-\frac{\alpha_h}{2}\delta_z \leq \phi_v \leq \frac{\alpha_h}{2}\delta_z, \quad -\frac{\alpha_v}{2}\delta_z \leq \frac{\pi}{2} - \theta_v \leq \frac{\alpha_v}{2}\delta_z,$$

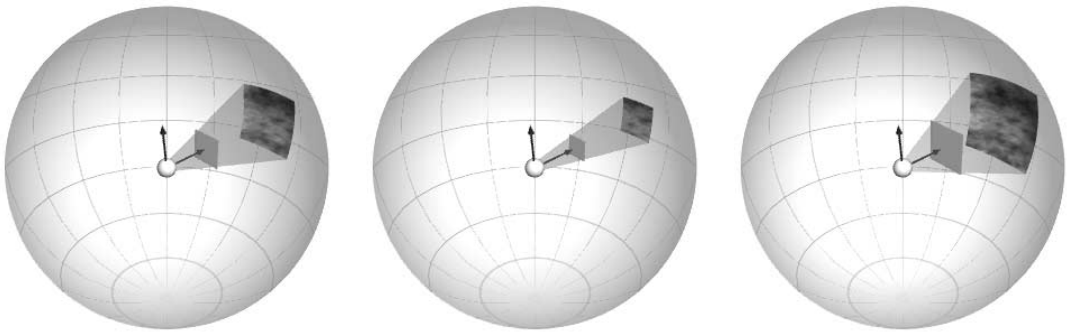
where α_h is the horizontal field of view,

α_v is the vertical field of view,

δ_z is the parameter to zoom, $\delta_z \geq 0$,

ϕ_v is the azimuth angle in the plenoptic function,

θ_v is the zenith angle in the plenoptic function.



(a) (b) (c)

Figure 6-9: Zooming in the plenoptic model. (a) Original, (b) zoom in, and (c) zoom out.

6.3.2 Walkthrough

Walking through a static scene is another basic operation that can be easily done in geometry-based computer graphics. In image-based computer graphics, this can also be done by translating the center of projection $\dot{E}$ along the translational vector $\vec{T}$ as follows,

$$\dot{E}' = \dot{E} + \vec{T},$$

where $\dot{E}$ is the original position of the center of projection,

$\dot{E}'$ is the new position of the center of projection after translation,

$\vec{T}$ is a translational vector.

Figure 6-10 shows the walkthrough done in plenoptic function graphically.

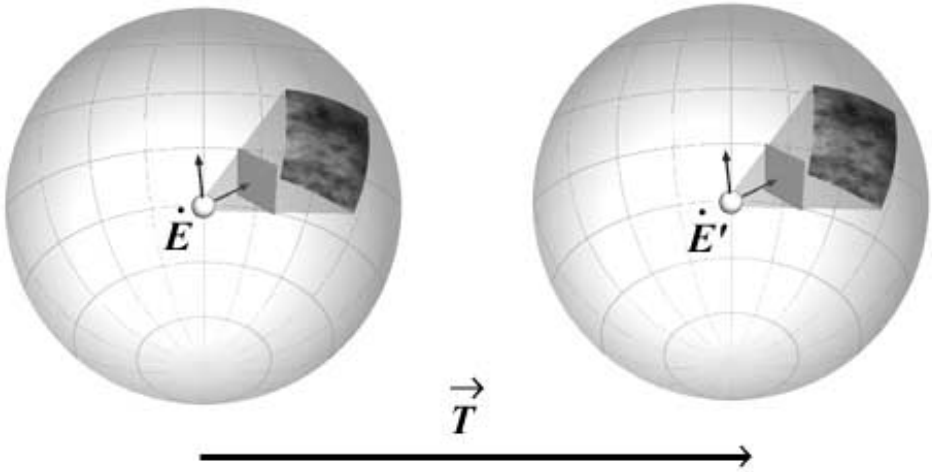


Figure 6-10: Walkthrough in plenoptic model.

6.4 The Plenoptic-Illumination Function

The original formulation of plenoptic function is very general and complete. Since its introduction is used for evaluating human vision model, the scene is almost always assumed fixed and the illumination is unchanged. Hence the time parameter t is fixed in most cases. However, for computer graphics, what we concern is the synthesis of the *desired* images. The capability of changing the lighting setup is essential. Unfortunately, the illumination and other scene changing factors are embedded inside a single time parameter t. In this section, we modify the original plenoptic formulation to include the factor of illumination.

What we do is to extract the illumination factor from t. To do so, we extend the original formulation to allow an explicit specification of the illumination component, the direction of the light source, $\vec{L}$. The new formulation is called *plenoptic-illumination function*.

$$I = \mathrm{P_I}(\vec{L}, \vec{V}, \dot{E}, t', \lambda), \tag{6.3}$$

where I is the irradiance,

$\vec{L} = (\sin\theta_l \cos\phi_l, \cos\theta_l, \sin\theta_l \sin\phi_l)$ specifies the direction of a directional light source,

$\vec{V} = (\sin\theta_v \cos\phi_v, \cos\theta_v, \sin\theta_v \sin\phi_v)$ specifies the viewing direction originated from the eye,

$\dot{E} = (E_x, E_y, E_z)$ is the position of the eye or center of projection,

t' is the time parameter which embeds all other scene changing parameters.

The difference between this new formulation and the old one (Equation 6.1) is the explicit inclusion of parameter light vector $\vec{L}$ which specifies the direction of a directional light which emits unit radiance. The function tells us the radiance coming from a viewing direction $\vec{V}$ arriving at our eye $\dot{E}$ at any time t' over any wavelength λ when the whole scene is illuminated by a directional light source from direction $-\vec{L}$ with unit radiance emission. Graphically, it can be illustrated by Figure 6-11. Comparing to Figure 6-1, a new light vector $\vec{L}$ is added.

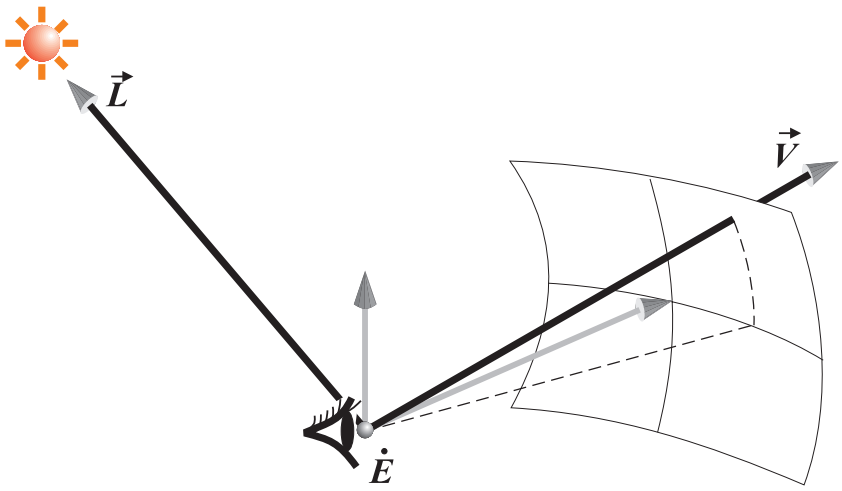


Figure 6-11: Geometry elements of plenoptic-illumination function.

The reason to specify the function using a directional light source is to simplify the construction. Moreover, even a directional source is used for parameterization, we shall see in Section 7.5.5 that it can be converted to other type of light source if extra geometric information is given. Since we are using a directional light source, there is no difference in saying where the light vector $\vec{L}$ is originated from. Therefore we choose to say the vector $\vec{L}$ is originated from the eye $\dot{E}$. This is simply a notation.

With the new formulation, we can easily pose the following question. *What will we see if the scene is illuminated by the sun at 10:00 am and at 6:00pm ?* By changing light vector $\vec{L}$ and fitting it into the plenoptic-illumination function, we can retrieve the required answer. Figure 6-12 demonstrates such query.

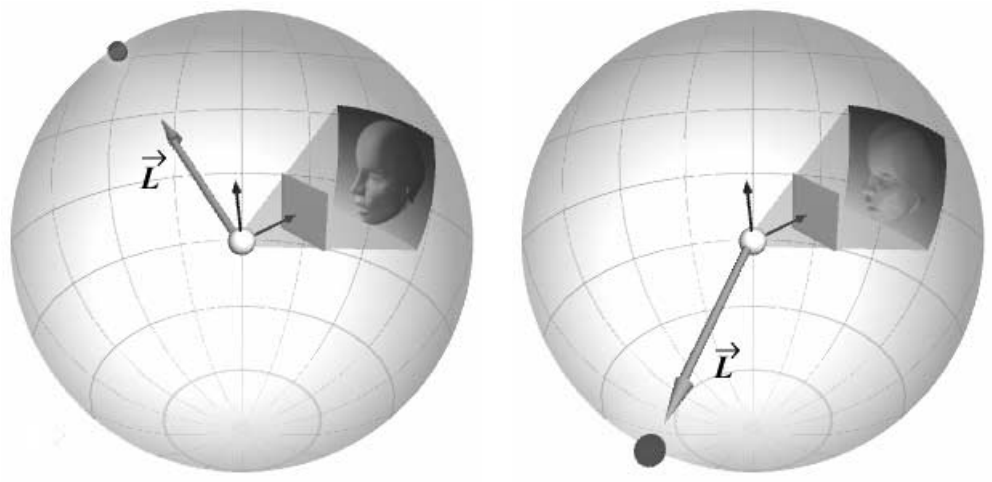


Figure 6-12: Querying the plenoptic value given the light vector $\vec{L}$.

6.5 Summary

In this chapter, we describe the plenoptic function which is suitable to act as the fundamental model for image-based computer graphics. Any image can be regarded as a subset of the complete plenoptic function. To demonstrate the generality of the plenoptic model, we compare the capability of image-based computer graphics based on plenoptic model and the traditional geometry-based computer graphics. The original formulation of plenoptic function allows standard camera motion such as panning, tilting, zooming and walkthrough. For scene and illumination changing, even though the original formulation can still express them using a single time parameter t, it is very inconvenient to query what the scene looks like when the scene is illuminated by a light from a given direction. Illumination is one crucial parameter in image synthesis. Therefore, we modify the original formulation of the plenoptic function to include an explicit illumination parameter. The new formulation is plenoptic-illumination function. We shall see in the next few chapters how useful this new formulation is when explaining our new image-based techniques that allow controllable illumination.

Chapter 7

Pixel's Bidirectional Reflectance Distribution Function

In Chapter 5, we mention that previous work in image-based computer graphics can be subdivided into two major streams. The first stream focuses on determining the correct perspective view while the second stream focuses on re-rendering the static scene under different illumination. In the first stream of previous work, the illumination of the scene is usually assumed to be fixed and carefully designed. On the other hand, the viewpoint is assumed fixed in the second stream. Some previous work [BELH96, ZHAN98] can re-render image under a *uncontrollable* light source. Other [NIME94] re-renders images under outdoor illumination only. Moreover, they *cannot* handle scene with *high specularity* due to their fundamental assumptions. For examples, scenes including a shiny glass or mirror cannot be correctly re-rendered using their approaches.

In this chapter, we present a new representation of image data that allows the change of viewpoint as well as the change of illumination. It is known as *pixel BRDF*. One thing we want to emphasize is that in our new approach the illumination is *controllable* and *general* enough to allow re-rendering of any scene (including highly specular scene) under any lighting environment. It can be thought as a special digital holographic stereogram that gives 3D illusion whenever the viewing direction or the illumination changes.

One major goal of image-based rendering is to minimize the use of geometrical information while generating physically correct images. With this goal in mind, the proposed image representation allows the viewer to change the viewpoint and the scene lighting without knowing geometrical details (say, geometry model) of the scene.

7.1 Illumination Models

To design an image-based representation that supports illumination, we need to know how illumination is done. For surfaces that do not emit light, they reflect the incoming light. Objects with different surface properties reflect different amount and proportion of light energy. This derivation allows us to distinguish surfaces with different color, shininess and surface roughness. For example, the reason

we can tell the object is red is because the surface reflects only the energy of the components around the red light and absorbs energy of other components in the light spectrum. Figure 7-1 shows how the light spectrum has been changed after reflection.

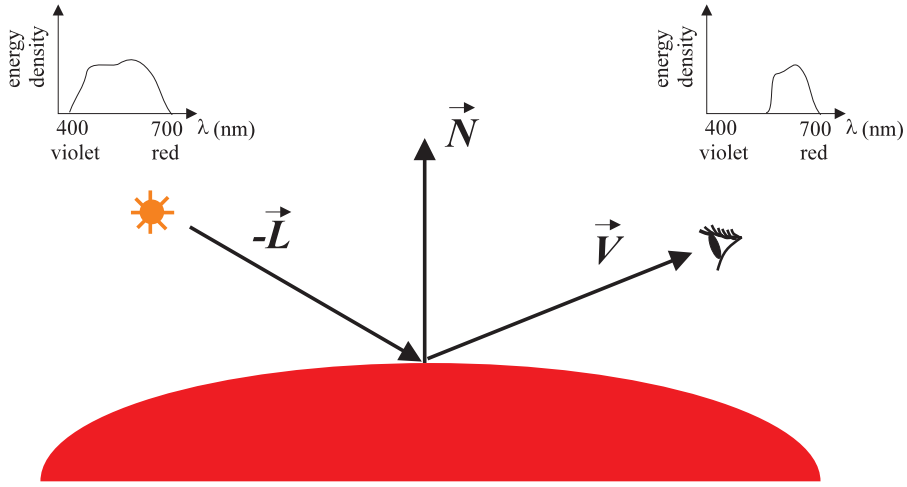


Figure 7-1: How can we tell an object is red?

Besides the surface properties, we know that the reflected light is also dependent on the direction of the incoming light and the direction of the viewing. The details of how a surface reflects light are described by various illumination models.

7.1.1 Local Illumination

In early years, most of the proposed illumination models are empirical. They are designed to mimic what human observed at different lighting condition. They are usually not physical-based. The most popular illumination model used (even in nowadays) is the Phong's illumination model [PHON75]. It is formulated as,

$$I = I_a \rho_a \; + \; \sum I_p [\rho_d (\vec{N} \cdot \vec{L}) \; + \; \rho_s (\vec{R} \cdot \vec{V})^n], \tag{7.1}$$

where I_a is the intensity of the ambient light,

I_p is the intensity of a point light source,

ρ_a is the ambient reflection coefficient,

ρ_d is the diffuse reflection coefficient,

ρ_s is the specular reflection coefficient,

$\vec{N}$ is the unit surface normal vector,

$\vec{L}$ is the unit light vector,

$\vec{R}$ is the unit mirror reflection vector,

$\vec{V}$ is the unit viewing vector,

n is the specular reflection exponent.

Figure 7-2 illustrates the geometric relationship among elements in the above formula. There are many variations in the formulation of the Phong's illumination model. All variations must include three basic reflection components, namely the ambient, diffuse and specular reflections.

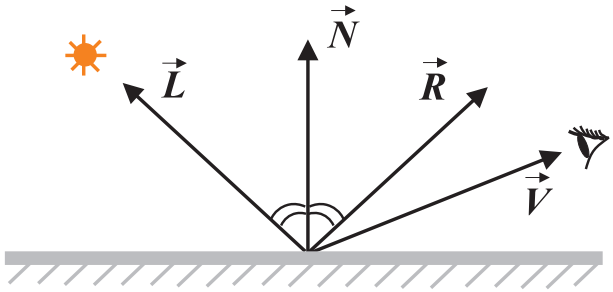


Figure 7-2: Geometry relationship between elements in the local illumination model.

Diffuse reflection component, sometimes called Lambertian reflection, accounts for the dull, matte reflection of surfaces, such as chalk. Diffuse surface reflects equal amount of light intensity in all directions. Diffuse reflection is view-independent. It is proportional to the the dot product $(\vec{N} \cdot \vec{L})$. Figure 7-3(a) shows a diffuse surface which reflects equal intensity in all directions (represented by the spherical gray region). On the other hand, specular reflection component accounts for the highlight on the shiny surfaces. Specular surface reflects unequal amount of intensity in different directions. Most intensity are reflected along the mirror reflection direction (Figure 7-3(b)). Therefore specular component is view-dependent. It is proportional to the power of dot product $(\vec{R} \cdot \vec{V})^n$. Mirror is one extreme case that its reflection contains mostly specular component. However, most

surfaces in the real world are in between diffuse and specular surfaces (Figure 7-3(c)). Although the Phong's model is empirical, it turns out that the physical-based models [TORR67, COOK81] only have a little difference in the formulation of the diffuse and specular components. In most cases, the empirical Phong's illumination model can still give a realistic appearance of surface.

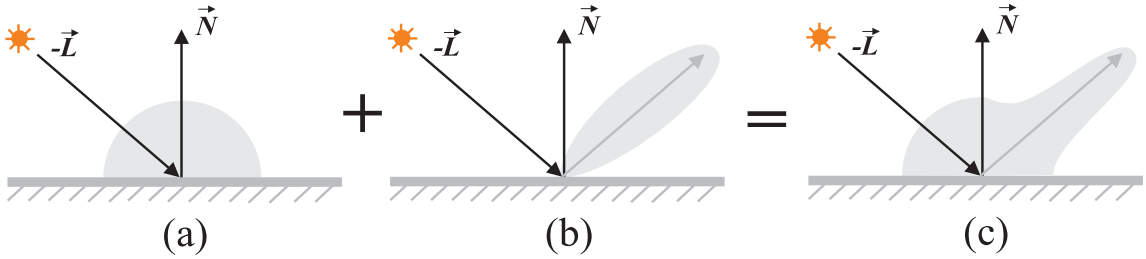


Figure 7-3: Most surfaces are combination of pure diffuse and pure specular surfaces.

The last reflection component is the ambient reflection (the term $I_a \rho_a$). It accounts for all the indirect intensity contribution reflected from nearby surfaces. Traditionally, it is assumed to be constant, since the true value is very difficult to find. It is this constant ambient assumption characterizes the illumination model as *local*. Since the reflected intensity can be calculated *locally* with the knowledge of local surface properties (ρ_a, ρ_d & ρ_s) and the geometric relationship between the interested surface element and the light sources. There is no need to know the surrounding geometry (global) since no indirect intensity contribution is accounted by the formulation.

7.1.2 Global Illumination

The lack of indirect contribution makes the early synthetic image looks artificial. This artifact leads to the research of *global illumination*. One famous technique is ray tracing [WHIT80] which produces more realistic images. In the middle of 1980's, researchers [GORA84, NISH85] began to apply radiosity methods to computer graphics and produced the state-of-the-art realistic imagery. Radiosity methods were first developed for computing the radiant energy interchange between surfaces [SIEG81]. These methods are used in various engineering applications, such as the analysis of radiative transfer between panels on spacecraft.

Early illumination models use *intensity* as one measurement of amount of light contribution. This quantity is usually not well-defined. Different models usually have different interpretation on this quantity. One of the contributions of applying radiosity methods to image synthesis, is radiosity methods use radiometry for measurement of radiant energy transfers. Radiometry provides a set

of physical and objective quantities. From now on, we will replace "intensity" with radiometric quantities, such as radiance and radiosity, in the following discussions.

The details of radiosity methods is out of the scope of this thesis. Interested readers are referred to the more complete literatures [SIEG81, COHE93, SILL94]. Here we will only provide a general idea of radiosity methods. The radiosity (total power) coming out from a surface element can be calculated by the following *Radiosity Equation*,

$$B(x) = E(x) + \rho(x) \int_{y \in S} B(y) \frac{\cos\theta \cos\theta'}{\pi r^2} V(x, y) dy, \tag{7.2}$$

where x is the surface element of interest,

y is any other surface element in the environment S,

$B(x)$ is the radiosity (total power leaving) of the surface element x,

$E(x)$ is the exitance (emitted energy) of the surface element x,

$V(x, y)$ is the visibility between x and y,

θ is the angle between the normal at x and the line connecting x and y (Figure 7-4),

θ' is the angle derivation at y (Figure 7-4),

r is the distance between the x and y,

ρ is the bidirectional reflectance.

Figure 7-4 shows the geometric relationship between each elements in the radiosity equation.

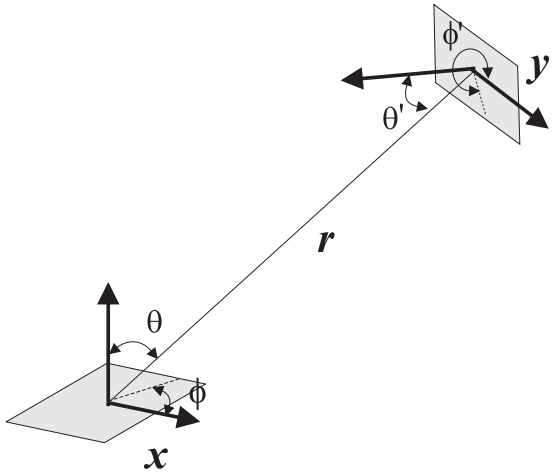


Figure 7-4: Geometric relationship between elements in the radiosity equation.

Intuitively speaking, the radiosity equation tells us that the radiosity of a surface element x is the sum of the radiant energy it emits (the first term on the right side of Equation 7.2) and the reflected radiant energy (the second term) which is contributed by all other surface elements y in the scene. Just like the previous local illumination model, the reflected radiant energy is also dependent on the surface properties ρ and the geometric relationship between the interested surface element x and its surrounding. But here, we are not just accounting for one or more specialized light sources, but all surfaces. In other words, every surface can be regarded as "light source".

7.2 BRDF Representation

By inspecting the illumination models described in Section 7.1, we can find that the elements characterizing the appearance of a surface are the reflectances (reflection coefficients in local illumination models). To calculate the light going out from a surface element in a specific direction, the reflectance of this surface element must first be determined.

The most general form of representing surface reflectivity is the *bidirectional reflectance distribution function* (BRDF) [KAJI85]. It describes the directional distribution of reflected light. In radiometry, BRDF is defined as the ratio of radiance in the outgoing direction and the radiant flux density (irradiance) along the incoming direction. It is a function of four angle parameters (θ_v, ϕ_v, θ_l, ϕ_l) or two vector parameters $\vec{V}$ and $\vec{L}$ which specify the direction of the viewing vector and the light vector respectively.

$$\rho(\vec{V}, \vec{L}) = \rho(\theta_v, \phi_v, \theta_l, \phi_l) = \frac{L_r(x, \theta_v, \phi_v)}{L_r(x, \theta_l, \phi_l) \cos\theta_l d\omega}, \tag{7.3}$$

where $L_r(x, \theta, \phi)$ is the radiance at x along direction (θ,ϕ),

(θ_v,ϕ_v) specifies the viewing direction $\vec{V}$,

(θ_l,ϕ_l) specifies the light source direction $\vec{L}$,

$d\omega$ is the differential solid angle.

Figure 7-5 illustrates the geometric relationship among the elements in the BRDF formulation. In the above formula, there is a basic radiometric quantity we have not yet explained. It is the *radiance*. Radiance $L_r(x, \theta, \phi)$ is defined as the amount of energy traveling at some point x along a specified direction (θ,ϕ), per unit time, per unit area perpendicular to the direction of travel, per unit solid angle.

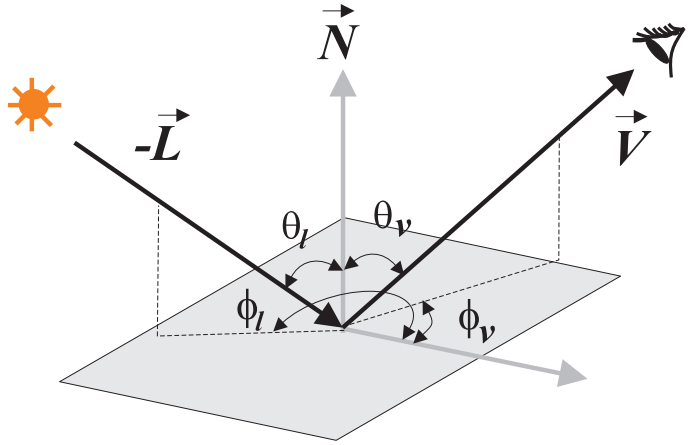


Figure 7-5: BRDF

Although it seems complex, BRDF can intuitively be regarded as the ratio of energy leaving in direction $\vec{V}$ and entering from direction $\vec{L}$ (Figure 7-5). The term $d\omega$ in the denominator is the differential solid angle (a differential area on the unit sphere, see Figure 7-6). It is equal to $\sin\theta_l d\theta_l d\phi_l$. The term $\cos\theta_l$ is used to project the differential area (solid angle) onto the surface plane. Therefore the aggregate term $\cos\theta_l d\omega$ (projected solid angle) can be thought as a weight to normalize $L_r(x, \theta_l, \phi_l)$. The BRDF is basically a table of reflectances indexed by four angles or two vectors.

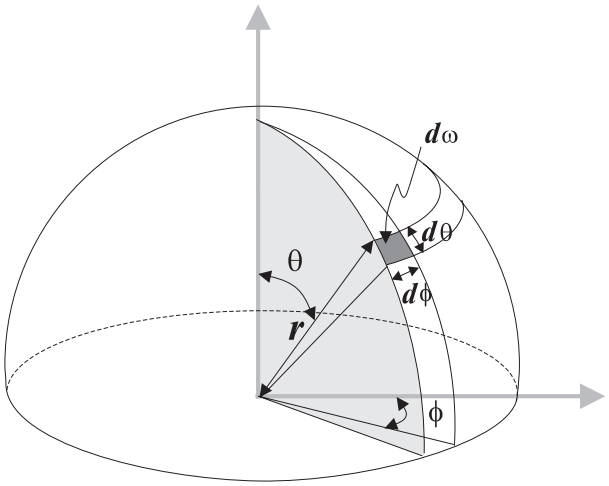


Figure 7-6: Differential solid angle.

Since the BRDF is a four-parameter function, it is not straightforward to imagine what will it looks like in 3D. It will be easier to visualize if one of the two vector parameters is fixed. When we fix one vector parameter, the BRDF reduces to a spherical function or a unidirectional reflectance

distribution function (URDF) which only depends on one vector parameter. When $\vec{L}$ is fixed, it becomes a function of $\vec{V}$ only. It tells us the reflectance of the interested surface when we are looking from various viewing direction inside the hemisphere enclosing the surface element. In most cases, the spherical functions will be very similar to the reflectance function (Figure 7-3(c)) in the empirical Phong's illumination model. Figure 7-7 shows a set of spherical functions (URDFs) of $\vec{V}$ when we fix the $\vec{L}$ at various directions. Note each one of them is quite similar to the one in Figure 7-3(c).

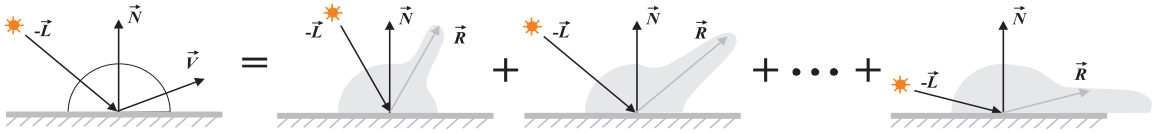


Figure 7-7: BRDF as a set of spherical functions (URDFs).

7.3 BRDF of Pixel

The most straightforward approach to include the illumination variability of the image-based computer graphics is to measure the BRDF of each object material visible in the image. However, this approach has several drawbacks. While the BRDFs of synthesized object surfaces may be assigned at will, measuring those of all objects in a real scene is tedious and often infeasible. Imagine a scene containing thousands of small stones, each with its own BRDF. The situation worsens when a single object exhibits spatial variability of surface properties. Furthermore, associating an BRDF to each object in the scene causes rendering time to depend on the scene complexity.

One might suggest, for each pixel in each view, to measure the BRDF of the object surface seen through that pixel window. This approach breaks the link to the scene complexity, but introduces an aliasing problem. Consider pixel A in Figure 7-8: multiple objects are visible through the pixel window. Note that this will frequently happen in images showing distant objects. Even if only one object is visible, there is still the problem of choosing surface normal for measuring BRDF when the object silhouette is curved (see pixel B in Figure 7-8).

Our solution is to treat each *pixel* on the image plane as a surface element with an *apparent* BRDF. Imagine the image plane as just an ordinary planar surface, and each pixel can be regarded as a surface element. Each surface element emits different amounts of radiant energy in different directions under different illuminations. In order to measure the (apparent) BRDF of each pixel, the location of the image plane must be specified (see Figure 7-9), not just the direction. By recording the BRDF of a

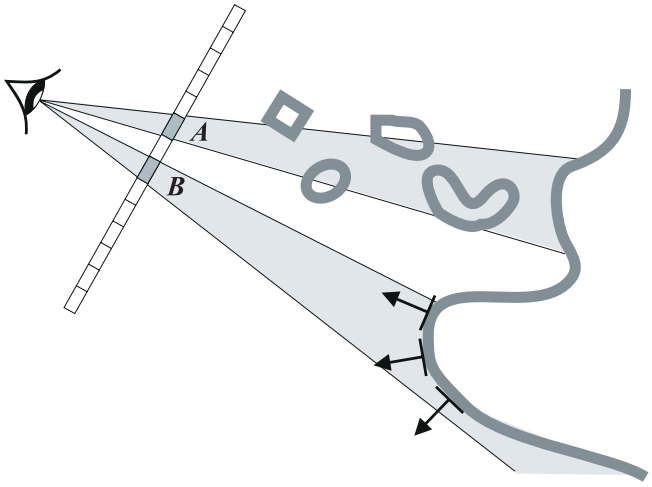


Figure 7-8: Aliasing problem of measuring BRDF of object surface visible through the pixel windows.

pixel (Figure 7-9), we capture the *aggregate reflectance* of objects visible through that pixel window. The light vector $\vec{L}$ from the light source and the viewing vector $\vec{V}$ from the viewpoint $\dot{E}$ define the two directions of the BRDF. This approach does not depend on the scene complexity, and removes the aliasing problems described before. It is also a unified approach for both virtual and real world scenes.

Note that the apparent BRDF represents the response of the object(s) within a pixel to light in each direction, *in the presence of the rest of the scene*, not merely the surface reflectivity. If the captured images (natural or rendered) include shadows, shadows will appear in the re-rendered result. We will show some example scenes that contain shadows later in this chapter.

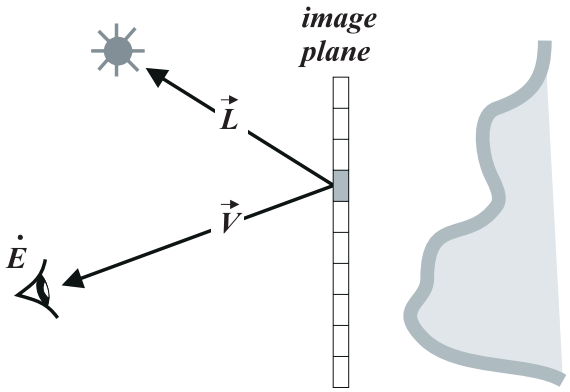


Figure 7-9: Measuring the BRDF of a *pixel*.

7.4 Measuring Pixel BRDF

To measure BRDF of a physical surface sample, the surface sample is first positioned in a gonioreflectometer [WARD92, MC90]. By illuminating the sample from different direction on the hemisphere and detecting the reflectance with a sensor from another direction. The apparatus can record a table of reflectances indexed by light vector $\vec{L}$ and the viewing vector $\vec{V}$. BRDF of synthetic surface [CABR87] can also be recorded using similar approach using an imaginary gonioreflectometer.

In our case, the apparent BRDF of pixel can also be sampled similarly. Of course we cannot physically position the imagery pixel patch inside a gonioreflectometer and capture its BRDF. But we can take photographs (render images) of the real world scene (virtual scene) under various illuminations. To do so, we first fix the direction of a directional light source. Then photographs of the scene are captured from various viewing directions over the sphere enclosing the pixel. Then the directional light source is fixed at another direction and images are captured from various viewing directions again. This process continues until the the directional light source has been placed all over the sphere enclosing the pixel. Figure 7-10 illustrates the process. The process is,

```
For each direction (θ_l, φ_l) or L⃗ of the directional light source
    For each viewing direction (θ_v, φ_v) or V⃗
        Render the virtual scene or take photograph of the
        real world scene illuminated by this directional
        source and denote the image as I_{θ_v,φ_v,θ_l,φ_l}.
```

One choice for sampling pattern is the spherical grid as shown in Figure 7-10. Sampling on the grid points of the spherical grid allows us to calculate the solid angle conveniently.

Careful readers may find that the above approach can only sample the BRDF of a center pixel from each grid point on the spherical grid. For its neighbor pixel, which is not located at the center of the sampling sphere, we cannot sample the BRDF of this neighbor pixel using the same set of directional vectors. Figure 7-11 illustrates the difference between the center pixel and its neighbor in 2D. That is why we use a directional light source to illuminate the scene, since the light vector for every point in space should be the same when using a directional light source. In real life, a directional light source can be approximated by placing a spotlight at a sufficient distance from the scene. For the viewing direction, we can use orthogonal projection to project the 3D scene onto the 2D image.

Figure 7-10: Capturing the BRDF of a *pixel*.

In real life, this can be done by placing the camera at a sufficient distance from the scene. We shall cover more practical issues on sampling in the next chapter.

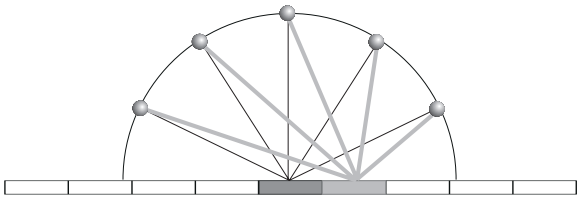


Figure 7-11: Difference in the sampling direction for the neighbor pixel.

Traditionally, the BRDF is sampled only over the upper hemisphere of the surface element, since reflectance must be zero if the light source is behind the opaque surface element. However in the case of pixel BRDF , the reflectance may be nonzero even the light source is from the back of the image plane. This is because the actual object surface may not align with the image plane (Figure 7-12). Instead, the whole sphere surrounding the pixel has to be sampled for recording its BRDF. Therefore, the range of zenith angle θ should be $[0, \pi]$.

If the pixel is real surface element, the pixel BRDF should be defined as follows,

$$\rho(\theta_v, \phi_v, \theta_l, \phi_l) = \frac{\texttt{radiance passing through the pixel in } I_{\theta_v,\phi_v,\theta_l,\phi_l}}{(\texttt{radiance due to light source}) \cos\theta_l d\omega}. \quad (7.4)$$

However, since the pixel is only an imaginary surface, the physical reflection does not take place at the pixel window. Instead the reflection is taken place at the real surface behind the pixel window.

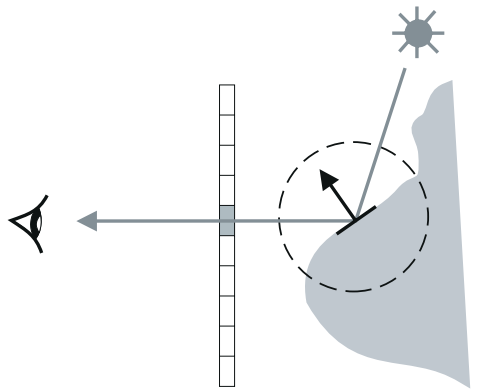


Figure 7-12: The image plane may not be parallel with the object surface.

It is meaningless to scale the radiance due the light source by the term $\cos\theta_l d\omega$, which is the projected solid angle. Hence, we drop this term and simply calculate the pixel BRDF as follows.

$$\rho(\theta_v, \phi_v, \theta_l, \phi_l) = \frac{\texttt{radiance passing through the pixel in } I_{\theta_v,\phi_v,\theta_l,\phi_l}}{(\texttt{radiance due to light source})}. \qquad (7.5)$$

One assumption is that there is no intervening medium, which absorbs, scatters or emits any radiant energy. Moreover, the pixel value in the captured image is simply assumed to be linear to the true radiance. For synthetic images, this may not be a problem, since the pixel value is linear to the computed radiance. For real world photographs, there is a non-linear relationship between the pixel value and the radiance (Figure 7-13). To remove this assumption, we can apply the radiance recovery process proposed by Debevec and Malik [DEBE97]. Although they have pointed out the quantity recovered is not the true radiance, it is linear to the radiance. Unfortunately, more photographs need to be captured in order to perform the recovery process.

Once the recording is finished, we have a 2D array of BRDFs. One BRDF for one pixel on the image plane. Figure 7-14 shows an example 3×3 image. It can be thought as a *special digital holographic stereogram* [BENT83]. Normal holographic stereogram shows a different image whenever the viewing direction change in order to give an illusion of 3D. Our "holographic stereogram" gives a different image not just when the viewing direction changes but also when the light source direction changes to give us an illusion of 3D object with adjustable illumination.

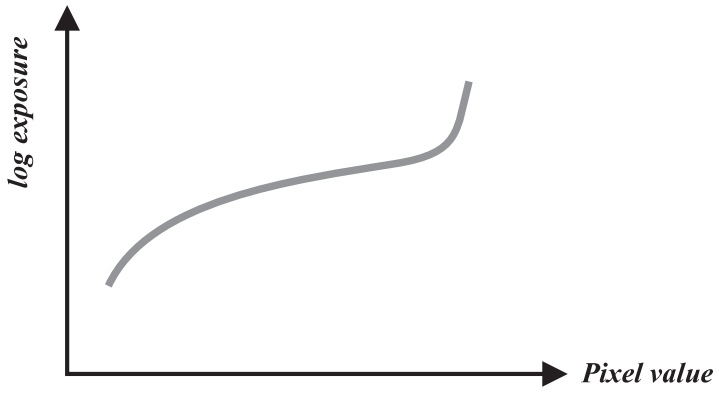


Figure 7-13: Nonlinear relationship between pixel value and true radiance.

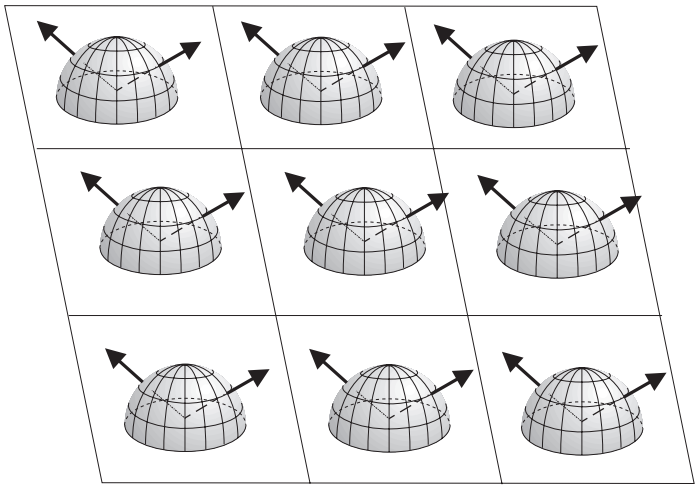


Figure 7-14: A 2D array of BRDFs.

7.5 Manipulating the BRDFs

Once the BRDFs are sampled and stored, they can be manipulated. Using the property of superposition [BUSB60], the final radiance (or simply value) of each pixel can be computed. We proposed a local illumination model (Equation 7.6) that makes use of superposition to re-render the image-based scene from different point of view under different illumination.

$$\text{radiance through pixel } x = \sum_{i}^{n} \rho_(\theta_v, \phi_v, \theta_l^i, \phi_l^i) L_r(x, \theta_l^i, \phi_l^i), \tag{7.6}$$

where n is the total number of light sources,

(θ_v, ϕ_v) specifies the viewing direction $\vec{V}$,

(θ_l^i, ϕ_l^i) specifies the direction, $\vec{L}_i$, of the i-th light source,

$L_r(x, \theta_l^i, \phi_l^i)$ is the radiance along (θ_l^i, ϕ_l^i) due to the i-th light source,

x is the position of the pixel.

Note the above illumination model is local. That is, it only accounts for the direct radiance contribution from the light sources. No indirect radiance contribution is accounted for. One may say that the pixel is not a physical surface element but a window in space, how come we can borrow the illumination model which models the light reflection from real surface. This illumination model is *not* the result of borrowing the existing illumination model but is *a result of utilizing the superposition property of images*.

In Section 7.3, we mentioned that the pixel BRDF is actually an aggregate BRDF of all visible objects behind the pixel. We will show here that how this aggregate BRDF can give us correct image. Consider k unoccluded objects [1] , visible through the pixel viewed from direction $\vec{V}$ and are illuminated by n light sources. The radiance passing through the pixel window in this view will be,

$$\begin{aligned} & \sum_{i}^{n} \rho_i^1 L_r^i + \sum_{i}^{n} \rho_i^2 L_r^i + \cdots + \sum_{i}^{n} \rho_i^k L_r^i \\ = & \sum_{j}^{k} \rho_1^j L_r^1 + \sum_{j}^{k} \rho_2^j L_r^2 + \cdots + \sum_{j}^{k} \rho_n^j L_r^n \end{aligned}$$

[1] If there exist objects that occlude each other, we can always subdivide the objects into visible (unoccluded) portions and invisible (occluded) portions. Invisible portions will never contribute any radiance to the final image. Hence we can consider only the unoccluded objects without loss of generality.

$$= \quad \rho_1 L_r^1 + \rho_2 L_r^2 + \cdots + \rho_n L_r^n$$

where ρ_i^j is the reflectance of the j-th object when illuminated by the i-th light source,
L_r^i is the short hand of $L_r(x, \theta_l^i, \phi_l^i)$, the radiance due to the i-th light source,
$\rho_i = \sum_{j=1}^{k} \rho_i^j$ is the aggregate reflectance we recorded when measuring the BRDF of the pixel.

The first row shows the sum of reflected radiances from all k unoccluded objects. Due to linearity of illumination models, we can reorder the terms to give the result on the third row which is the sum of multiplications of the aggregate reflectances and the radiance of each light source.

7.5.1 Change of View Point

Using Equation 7.6, we can change the point of view by substituting a different viewing vector $\vec{V}$ or (θ_v, ϕ_v). Figures 7-15 shows an image-based teapot from different point of view. Note that we don't have the geometry of the teapot. What we have is a 2D array of BRDFs or a "special digital holographic stereogram".

(a) (b)

Figure 7-15: Change of viewpoint. (a)Front view, (b)Looking from the bottom.

7.5.2 Light Direction

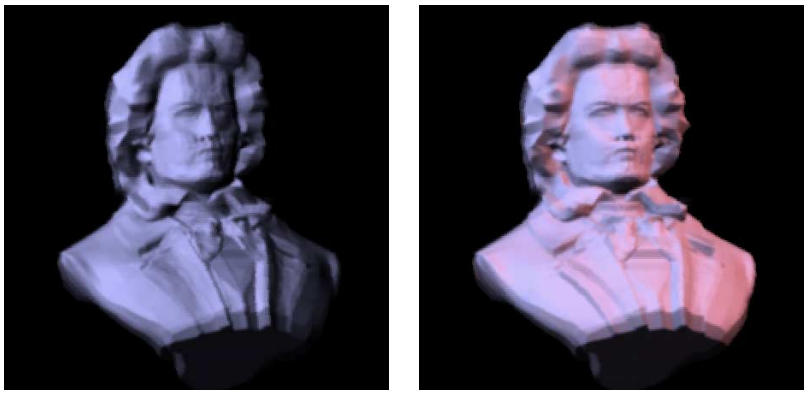
With Equation 7.6, the light direction can also be changed by substituting a different value of (θ_l, ϕ_l). Figures 7-16(a) and (b) show an image-based teapot illuminated by a light source from the top and the right respectively. Again, we don't have any geometry model.

(a) (b)

Figure 7-16: Change of light direction. (a)Light from the top of the teapot. (b)Light from the right hand side

7.5.3 Light Intensity

Another parameter to manipulate in Equation 7.6 is the intensity of the light source. This can be done by changing the value of L_r^i for the i-th light source. Figure 7-17(a) shows the Beethoven statue illuminated by a blue light from the left.



(a) (b)

Figure 7-17: Multiple light sources with different color. (a)Left: Beethoven statue illuminated by a single blue light from the left. (b)Right: One more red light comes from the right.

7.5.4 Multiple Light Sources

We can arbitrarily add any number of light sources. The trade-off is the computational time. An additional multiplication and addition have to be computed in evaluating Equation 7.6 for each newly added light source. In the Figure 7-17(b), the Beethoven statue is illuminated by a blue light from the

left and a red light from the right simultaneously.

7.5.5 Type of Light Sources

Up to now, all light sources we mentioned previously are directional. It is very efficient to evaluate Equation 7.6 if the light source is directional, because all pixels on the same image plane are illuminated by light source from the same direction (θ_l^i, ϕ_l^i). Moreover, no geometry information is required to re-render scene when it is illuminated by directional sources.

However, the method is not restricted to directional light. It can be extended to point source, spotlight or more general light source as well. It will be more expensive to evaluate Equation 7.6 for other type of light sources, since (θ_l^i, ϕ_l^i) will need to be recalculated from pixel to pixel. Since the image plane where the pixels are located is only a window in the 3D space (Figure 7-18), the intersecting surface element that actually reflects the light may be located on any point along the ray $\vec{V}$ in Figure 7-18. To find the light vector $\vec{L}$ correctly for other types of light sources, the intersection point of the ray and the object surface have to be located first. Note there is no such problem for directional source, since the light vector is identical for all points in the 3D space. One way to find $\vec{L}$ is to use the depth image. While this can be easily done for synthetic scenes, real world scenes may be more difficult. Use of a range scanner or computer vision techniques may provide a solution.

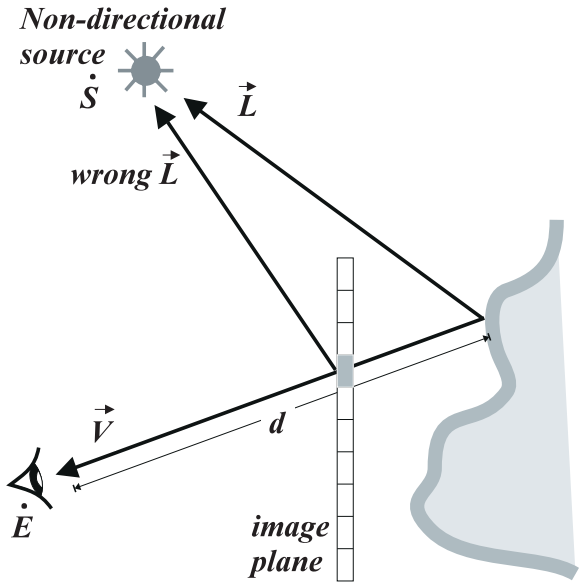


Figure 7-18: Finding the correct light vector.

Without the depth map, the re-rendered image is physically correct only when the scene is illuminated by directional sources. With the additional depth map, we can correctly re-render the scene illuminated by any type of light source by finding the correct light vector $\vec{L}$ using the following equation,

$$\vec{L} = \dot{S} - \dot{E} + \frac{\vec{V}}{|\vec{V}|}d \tag{7.7}$$

where $\vec{L}$ is the light vector,

$\dot{S}$ is the position of the non-directional light source,

$\dot{E}$ is the position of the eye,

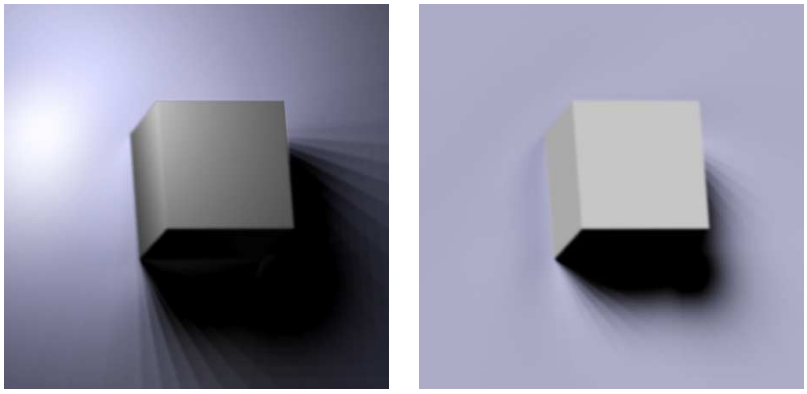
$\vec{V}$ is the viewing direction,

d is the value from the depth map.

It can be a point source, a spotlight or even a slide projector source. Figures 7-19(a) and (b) show a box on a plane illuminated by a point source and a directional source respectively. Note that all input reference images capture the scene *only* illuminated by a directional light source. No other type of light sources are used in the input reference images. Surprisingly, with the extra depth information, we can synthesize the image-based scene illuminated by other types of light sources.

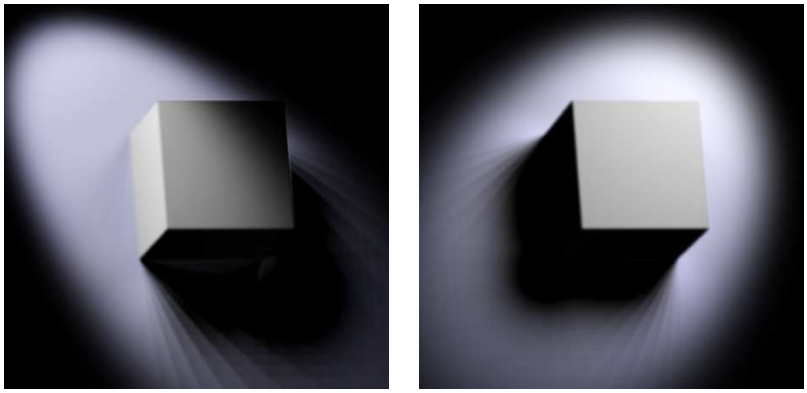
As we have mentioned before, the re-rendered image can contain shadow if it is recorded in the pixel BRDFs during sampling. Note the difference in the shadow cast by these sources. Figure 7-20 demonstrates the re-rendered result of the same scene illuminated by a spotlight. The intensity ramp on the protrusive box surface is differentiable from that on the background plane in Figure 7-20(a). Figure 7-21 shows the result of casting slide images onto the same scene. It demonstrates that the reconstruction process is independent of the type of light source. Theoretically, we can even illuminate the scene with area light source by trading off the computational time.

However, just as we discussed in Section 7.2, there is an aliasing problem in finding the correct intersection positions. Imagine a scene of a furry teddy bear; thousands of objects may be visible through one pixel window.



(a) (b)

Figure 7-19: Point and directional light sources. (a)Left: shadow cast by a point source. (b)Right: shadow cast by a directional source.



(a) (b)

Figure 7-20: Spotlight. (a)Left: Scene illuminated by a spot light source from the left. (b)Right: Same scene illuminated by a right spot light source.

(a) (b)

Figure 7-21: Slide projector. (a)Left: Scene illuminated by a slide projector source from the right. (b)Right: Same scene illuminated by another slide projector source.

7.6 A Subset of Plenoptic-Illumination Function

In Section 6.4, we proposed a new formulation of plenoptic function, the plenoptic-illumination function, as the fundamental computational model for image-based computer graphics due to its capability of specifying the lighting. We have also shown that standard perspective images and panoramic images are subsets of the complete plenoptic function. In this section, we will show that the proposed pixel BRDF is also a subset of the plenoptic-illumination function.

Since the light source we used for sampling the pixel BRDF is directional, the light vector for any point in the space should be the same. There is no difference in saying whether light vector $\vec{L}$ is originated from the pixel or from the viewpoint $\dot{E}$ (Figure 7-22). For each value $\rho(\vec{V}, \vec{L})$ in the pixel BRDF, we can always express it in the form of plenoptic-illumination function in the following manner,

$$\rho(\vec{V}, \vec{L}) = P_I(\vec{L}, -\vec{V}, \dot{E}, t', \lambda).$$

We simply translate the light vector $\vec{L}$ from the pixel to $\dot{E}$ and invert the viewing vector $\vec{V}$.

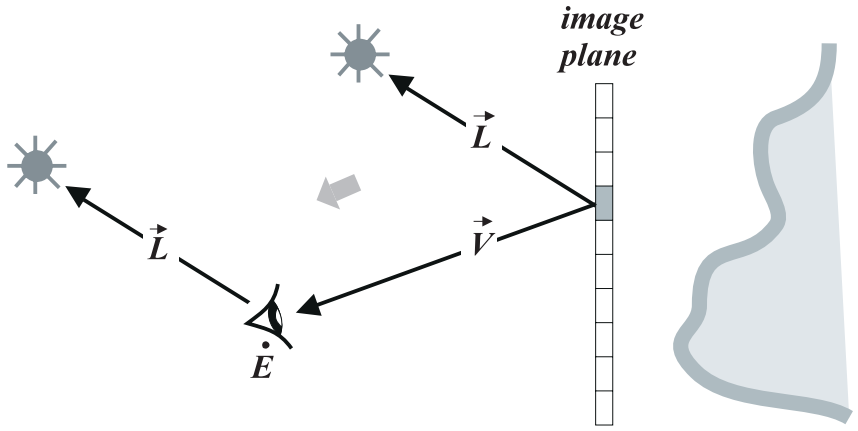


Figure 7-22: Pixel BRDF as subset of plenoptic-illumination function.

7.7 Summary

In this chapter, we introduce the concept of measuring BRDF of a pixel. This concept is useful in representing reflectance in the image-based computer graphics. In image-based computer graphics, we are no longer accessible to the geometry of the scene, hence no way to measure and model the BRDF of a surface element on the object. By measuring the BRDF of pixel, we can re-render the

scene from different point of view under different lighting environment, just like what we do in the geometry-based computer graphics. We also show that the pixel BRDF is actually a subset of the proposed plenoptic-illumination model.

Chapter 8

Applications of Pixel BRDF

We have proposed the concept of measuring pixel BRDF in Chapter 7. The definition of pixel BRDF does not restrict the camera model to be used. A camera model defines the portion of the scene to be visible through a pixel window. For example, an image captured by a planar pinhole camera (perspective projection) should be different from another image captured by a camera with fisheye len (distorted fisheye projection). For the simplicity of discussion in Chapter 7, we assume the camera used is parallel projected model (so that the viewing vector $\vec{V}$ will be the same for each pixel). In this chapter, we will show that the representation of pixel BRDF is applicable to other camera models as well.

We will show how to extend the usage of pixel BRDF to both perspectively projected and panoramic images. Instead of discussing these two camera models one by one, this chapter is organized in a different way. We will discuss the two major actions in image-based computer graphics. The first is the *inward-viewing* action and the other is the *outward-viewing* action. An inward-viewing action is an action that holds an object in hand and allows to change its orientation in order to investigate it. For example, manipulating a diamond in hand to examine its various sides. On the other hand, an outward-viewing action is an action that stands at a fixed point and looks around in an environment. For example, standing in a museum and looking around to view the artworks. The commercial software QuickTime VR [CHEN95a] defines two types of VR movies, namely the object movie and the panoramic movie. The object movie is actually an inward-viewing application while the panoramic movie is an outward-viewing application.

We first describe two existing image-based representations that are useful for implementing inward-viewing and outward-viewing applications. Both representations allow the viewer to change his/her viewpoint only. They do not support the change of illumination. We then apply the concept of measuring pixel BRDF to both of them in order to include the illumination.

8.1 Viewing Inward

Figure 8-1 illustrates what is an inward-viewing action. Basically, it is an action that looks at a fixed point in 3D from different viewing directions. The goal is to give a visual effect of holding an object in hand and allowing the object to rotate in any direction. To achieve an effect of inward-viewing using image-based computer graphics, the most straightforward approach is like the following. Firstly, images of the object viewing from different points of view are captured. Then, interpolation is used to warp the reference images to give desired image if the desired view is in between the sampled viewing directions. Various organizations of reference images have been proposed [FOLE90, LEVO96, GORT96, IHM97]. Among them, the light slab organization [LEVO96, GORT96] is a promising approach due to its capability of being rendered using graphics hardware.

Figure 8-1: An inward-viewing example.

8.1.1 Light Slab Organization

Levoy and Hanrahan [LEVO96] and Gortler *et al.* [GORT96] reduced the plenoptic function to a 4D light slab organization. It is also known as two-plane parameterization [GORT96, GU97] because the light slab is composed of two parallel planes. One plane (focal plane) is closer to the object/scene while the other (camera plane) is closer to the viewer. Let's denote the camera and focal planes as uv and st planes respectively. Figure 8-2 shows the light slab geometry.

Any ray $\vec{V}$ entering the light slab volume from the st plane and exiting through uv plane can be represented by the quadruple (u, v, s, t). Therefore we can record the radiance coming along the ray

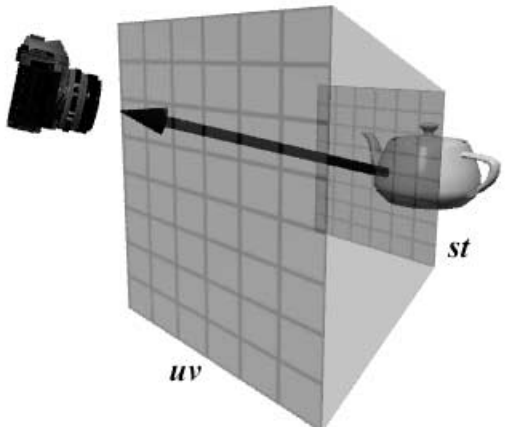


Figure 8-2: The light slab.

$\vec{V}$ in a table indexed by the quadruple (u, v, s, t). Levoy and Hanrahan called this table the *light field* while Gortler *et al.* called it the *Lumigraph*.

To record the radiances along any ray passing through the light slab, every point on the uv and st planes should be sampled. That is, for every point (u_i, v_i) on the uv plane, we should take a look at every point (s_j, t_j) on the st plane and record its radiance. Of course, this is impractical. Instead, we will position the camera at the grid point of a *sparse* grid on the camera plane, and take photographs of the focal plane. That means, we take sparse samples on the uv plane but *dense* samples on the st plane (due to the high resolution of the film). Figure 8-3 shows the process of recording the light field (or Lumigraph) from the top.

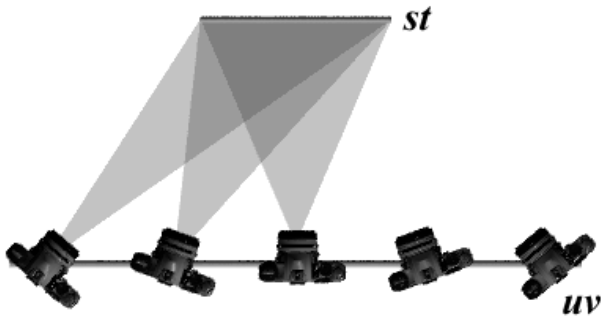


Figure 8-3: Recording the light field or Lumigraph (top view).

In Figure 8-3, all cameras are pointing at the center of the focal plane to capture the st samples on

the whole st plane. To do so, nearly all cameras (except the center one) have to rotate slightly away from the normal of the camera plane (Figure 8-3).

To visualize the recorded light field, we can organize the radiances in a 2D uv array of st images (Figure 8-4). Each image in the array represents the radiance coming from different points on the st plane arriving at one specific point on the uv plane. It also looks like an array of the same scene viewed from various directions.

Figure 8-4: The 2D array of st images captured at the grid points of a 5×5 uv grid.

With the light field, we can synthesize an image viewed from the desired viewpoint using backward ray tracing. Figure 8-5 shows how we generate a desired image. To fill the color of a pixel in the desired image, we fire a ray $-\vec{V}$ from the desired viewpoint into the light slab. As we have mentioned before, this ray can be represented by the quadruple (u, v, s, t). Using the quadruple, we can retrieve a value from the light field table. This value tells us the radiance coming out from the camera plane along the direction $\vec{V}$. Repeating this process for every pixel in the desired image, we finish the image synthesis.

However, we usually take sparse samples on the uv plane. Therefore, we have to estimate the radiance value when the ray shot does not pass through the uv sample point. One simple way to estimate is to use radiance of the nearest sample. In the example of Figure 8-6, the ray sample "$\times$"

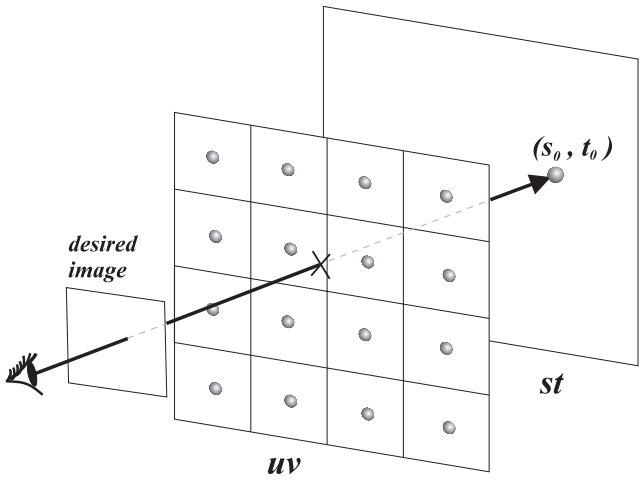


Figure 8-5: Synthesize the desired image using ray tracing.

does not coincide with any recorded uv sample. In this case, we use of the value at uv sample 6.

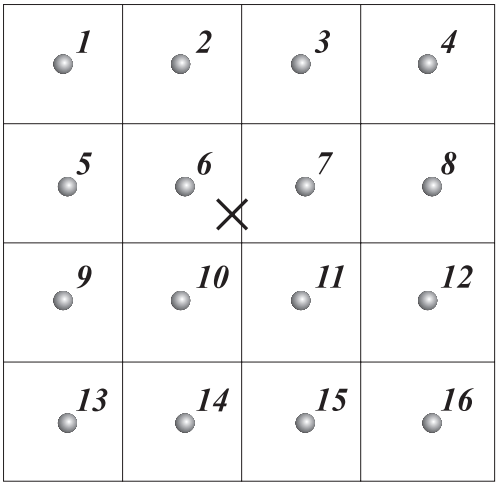
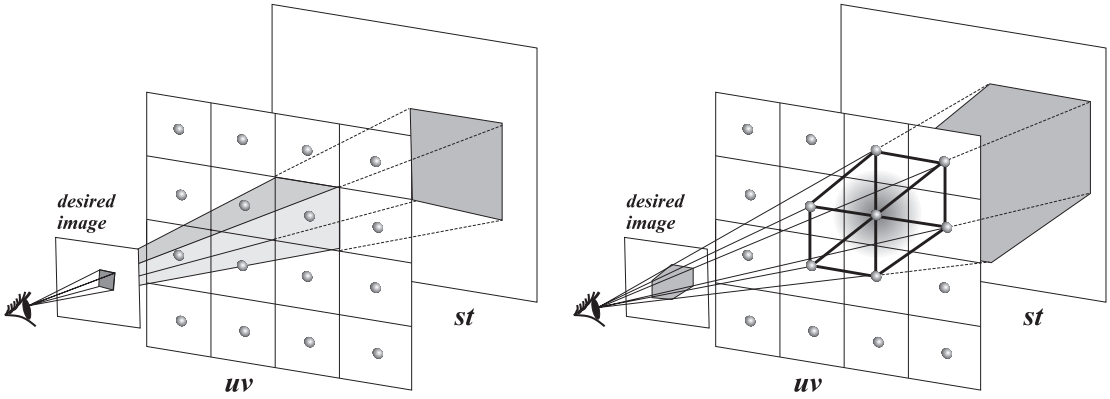


Figure 8-6: Guessing the radiance value.

This is actually a constant basis approach [GORT96]. If we estimate the radiance in this manner, there is no need to use ray tracing. Using texture mapping can give us the same result. Consider the example in Figure 8-7(a), we have taken 4 $\times$ 4 uv samples on the uv plane. The uv plane is first subdivided into 4 $\times$ 4 equal uv rectangles. To synthesize the projection (shaded region on the desired image) of the rectangle on the desired image, we first mount the st image associating with this uv sample on the st plane. Then we draw the uv rectangle with a texture mapped. The texture will be the associated st image. The texture coordinates are calculated by firing four rays through the four

corners of the uv rectangle and intersecting with the st plane.



(a) (b)

Figure 8-7: Constant basis (a) vs. linear-bilinear basis (b)

Constant basis will give annoying discontinuity when moving the viewpoint gradually from left to right. A smoother approach is suggested by Gortler *et al.* [GORT96], called linear-bilinear basis (Figure 8-7(b)). Basically, it uses weighted sum of neighbor samples to estimate the radiance. The linear-bilinear basis approach uses six more neighbor samples to interpolate the radiance values. The rendering is can be done by drawing texture mapped polygons with alpha blending. The gray ramp in the span on the uv plane shown in Figure 8-7(b) indicates the weight. The darker the color is, the higher its weight is.

8.1.2 Illuminating Light Field

The light field model only supports the change of viewpoint. It does not support any change of illumination. We now apply the proposed pixel BRDF concept to the light field data structure. In the light field, the viewing direction $\vec{V}$ is implicitly specified by the quadruple (u, v, s, t) as in Figure 8-8. What is missing is the light vector $\vec{L}$. Therefore, by extending the light field with one more dimension (the dimension of the light vector), it can then support illumination. The new formulation of light field is a function of six elements, instead of four elements in the original light field function.

$$f(u, v, s, t, \ \theta_l, \phi_l) \ \ or \ \ f(u, v, s, t, \ \vec{L}), \tag{8.1}$$

where (u, v, s, t) specifies the viewing direction $\vec{V}$,

(θ_l, ϕ_l) specifies the light vector $\vec{L}$.

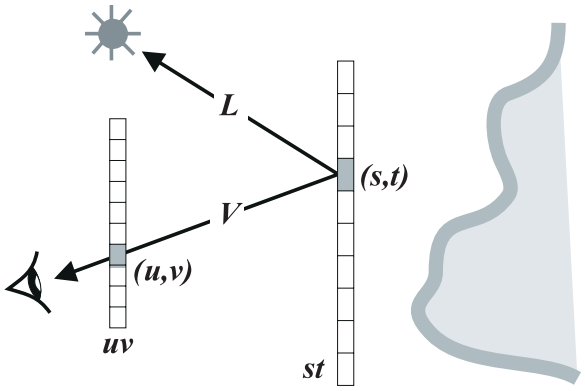


Figure 8-8: Extending the light slab based systems to allow change of illumination.

Note that Equation 8.1 is very similar to the pixel BRDF (Equation 7.3). The only difference is the viewing vector is now replaced by (u, v, s, t).

8.1.3 Sampling Light Field With Illumination

The basic idea to sample the light field with illumination is the same as the sampling mentioned in Section 7.4. That is, for each light direction $\vec{L}$, we capture images from a different viewing direction $\vec{V}$. Then change the light to another direction and capture the images from the same set of viewpoints. The process repeats until all light directions are captured.

In Section 7.4, we suggest to sample both the viewing and light directions on the spherical grid. Here, we separate the sampling patterns of viewing vectors from those of light vectors. We preserve the sampling pattern of the original light field for sampling the viewing vector. That is, placing the camera only at the rectangular grid points on the uv plane. On the other hand, we still sample the light vector over the sphere enclosing the st plane. Figure 8-9 shows the two different sampling structures for the viewing and light vectors. This arrangement allows us to re-render the desired image using hardware texture mapping capability (discussed shortly).

Each image (st image) in the light slab structure can be captured using perspective projection (Figure 8-3). Moreover, images are only captured at certain uv samples. That means, the sampling pattern of the viewing vector $\vec{V}$ will not be the same for each pixel, just like the case in Figure 7-11. One solution is to perform resampling to evaluate the values on the grid points. This will introduce errors.

A better solution that does not need resampling on the st plane, is not to record the pixel BRDFs, but to record the sets of pixel URDFs. As mentioned in Section 7.2, BRDF can also be represented

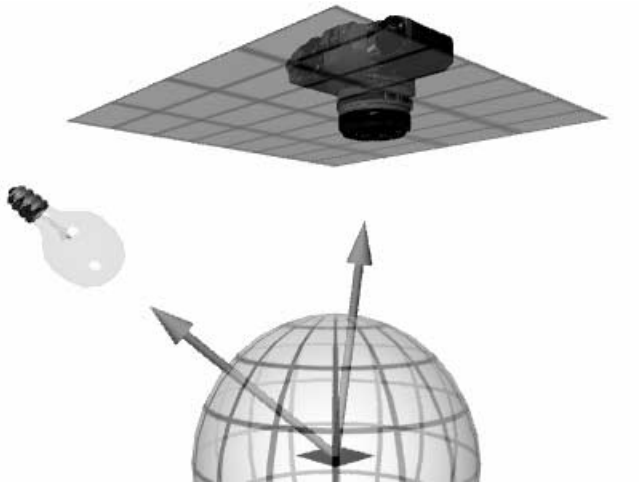


Figure 8-9: The two different sampling structures of the viewing and light vectors.

as a set of URDFs (spherical functions) (Figure 7-7). To record the set of pixel URDFs, we first fix the camera at one uv sample point (freeze the $\vec{V}$ at one direction). Then we illuminate the scene with a directional light source from a specific direction (which is equivalent to a sample point on the unit sphere) and take a photograph. Next, the light source is changed to another direction (another sample on the sphere) and another photograph is taken. The process continues until desired number of samples on the sphere are sampled. Once the sampling is finished, we have recorded a spherical function (URDF) of radiance for each pixel on the st image. Next we move the camera to another uv sample point and repeat the process. Again, for each pixel on the st image, we have captured another spherical function of radiance. If we take n uv sample points, we will have n spherical functions for each pixel.

To re-render the scene using the set of pixel URDFs, we first generates an image (may not be the desired one) for each uv sample point using Equation 7.6. That is, if there are 4 $\times$ 4 uv samples, we first generate 4 $\times$ 4 images as looking from these uv sample positions. Then these images are blended together using the constant or quadralinear blending methods (Figure 8-7) proposed by Gortler *et al.* [GORT96] to give the desired image. Figure 8-10 shows 4 images on the left are first re-rendered given the light source information and then blended to give the desired image on the right.

Therefore, we actually separate the re-rendering process into two steps. While the first step is done purely by software, the second step which involves drawing of texture mapped polygons can be accelerated by graphics hardware. This approach will give faster interactive feedback if the viewer

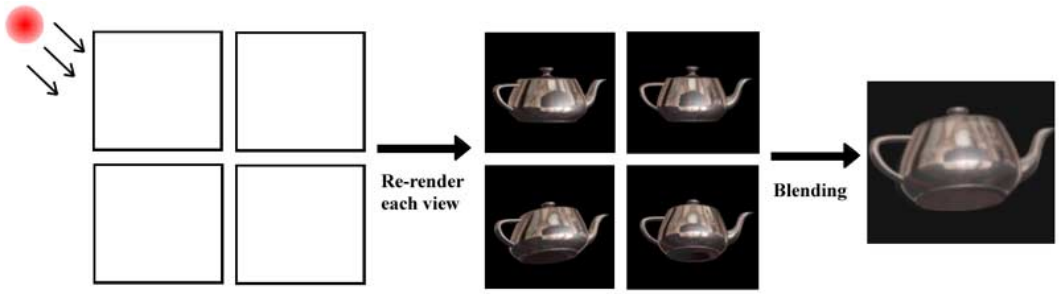


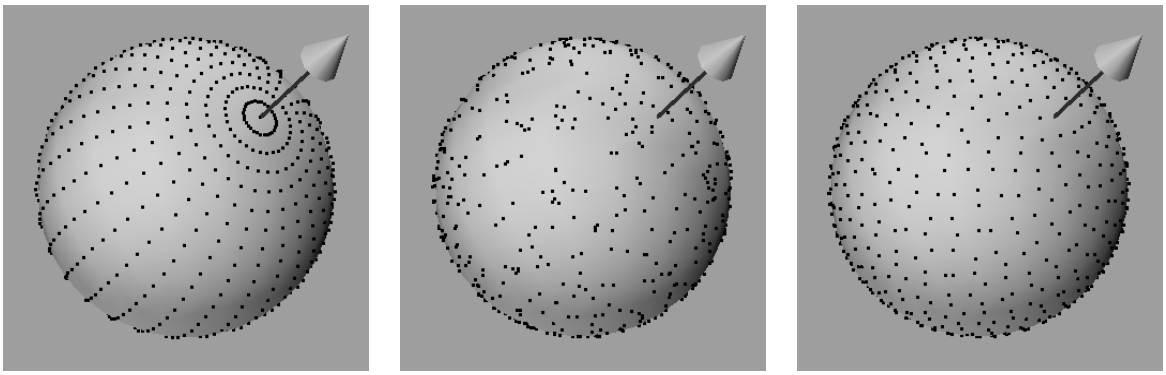
Figure 8-10: Re-render using set of pixel URDFs.

changes the viewpoint more frequent than the lighting.

8.1.4 Sampling on a Sphere

For the light direction, we still sample it over a sphere (Figure 8-9). Sampling the directional vector over a sphere is equivalent to sampling points on the surface of a unit sphere. The simplest sampling pattern is the spherical grid (Figure 8-11(a)). However, the sample points are not uniformly distributed on the sphere. More samples are located on the south and north poles while fewer samples are on the equator (see Figure 8-11(a)). Moreover, regular pattern is usually more sensitive to the problem of aliasing. An opposite approach is to sample randomly, which may give a uniformly distributed pattern and less sensitive to aliasing(Figure 8-11(b)). However, it gives noisy result. Several techniques in between random and regular sampling have been proposed. The thesis of Shirley [SHIR91b] surveys the common sampling techniques, including jittered [COOK84b], semi-jittered, Poisson disk and N-rooks sampling. Cychosz [CYCH90] generated sampling jitters using look-up tables. Chiu *et al.* [CHIU94] combined jittered and N-rooks methods to design a new multi-jittered sampling. Cross [CROS95] used a genetic algorithm to find the optimal sampling pattern for uniformly distributed edges. All these methods make tradeoffs between noisiness and aliasing.

Discrepancy analysis measures sample point equidistribution, that is, measures how uniformly distributed the point set is. Shirley [SHIR91a] first applied it to the sampling problem in computer graphics. The possible importance of discrepancy in computer graphics is also pointed out by Niederreiter [NIED92]. Dobkin *et al.* [DOBK93a, DOBK93b, DOBK96] proposed various methods to measure the discrepancy of sampling patterns and to generate the patterns [DOBK96]. Heinrich and Keller [HEIN94a, HEIN94b, KELL95] and Ohbuchi and Aono [OHBU96] applied low discrepancy sequences to Monte Carlo integration in radiosity applications. The term *quasi-Monte Carlo* is used



(a) (b) (c)

Figure 8-11: Sampling patterns on a sphere. (a) Spherical grid, (b) random sampling, (c) quasi-Monte Carlo sampling.

to describe applications which apply the low discrepancy sequences in solving the sampling problem.

We have applied a low discrepancy sequence, namely the Hammersley point set, in sampling the light vector [WONG97c]. Hammersley points have been used in numerical [PASK95, TRAU96, CASE95] and graphics [HEIN94a, HEIN94b, KELL95, OHBU96] applications, with a significant improvement in terms of error.

Hammersley point set is a uniformly distributed and stochastic point set generated by a *deterministic* equation. Let's define the Hammersley point set. Each nonnegative integer k can be expanded using a prime base p:

$$k = a_0 + a_1 p + a_2 p^2 + \ldots + a_r p^r. \tag{8.2}$$

where each a_i is an integer in $[0, p-1]$. Now define a function Φ_p of k by

$$\Phi_p(k) = \frac{a_0}{p} + \frac{a_1}{p^2} + \frac{a_2}{p^3} + \cdots + \frac{a_r}{p^{r+1}}\,. \tag{8.3}$$

If $p = 2$, the sequence of $\Phi_2(k)$, for $k = 0, 1, 2, \ldots$, is called the Van der Corput sequence [TEZU95].

Let d be the dimension of the space to be sampled. Any sequence $p_1, p_2, \ldots, p_{d-1}$ of prime numbers defines a sequence Φ_{p_1}, Φ_{p_2}, ..., $\Phi_{p_{d-1}}$ of functions, whose corresponding k-th d-dimensional Hammersley point is

$$\left(\frac{k}{n}, \Phi_{p_1}(k), \Phi_{p_2}(k), \ldots, \Phi_{p_{d-1}}(k)\right) \quad \text{for } k = 0, 1, 2, \ldots, n-1. \tag{8.4}$$

where $p_1 < p_2 < \cdots < p_{d-1}$,

n is the total number of Hammersley points.

To evaluate the function $\Phi_p(k)$, the following algorithm can be used.

$$
\begin{aligned}
&p' = p\ ,\ k' = k\ ,\ \Phi = 0 \\
&\text{while } k' > 0 \ \text{ do} \\
&\qquad a = k' \bmod p \\
&\qquad \Phi = \Phi + \tfrac{a}{p'} \\
&\qquad k' = \texttt{int}(\tfrac{k'}{p}) \\
&\qquad p' = p'p
\end{aligned}
$$

where $\texttt{int}(x)$ returns the integer part of x.

The above algorithm has a complexity of $O(\log_p k)$ for evaluating the k-th point. Hence the worst case bound of the algorithm for generating $(N+1)$ points is,

$$
\begin{aligned}
& \log_p(1) + \log_p(2) + \cdots + \log_p(N-1) + \log_p(N) \\
\leq\ & \log_p(N) + \log_p(N) + \cdots + \log_p(N) + \log_p(N) \\
=\ & N \log_p N.
\end{aligned}
$$

A Pascal implementation of this algorithm can be found in [HALT64]. In most computer graphics applications, the dimension of the sampled space is either 2 or 3. In our application, we concentrate on the generation of a uniformly distributed point set on the surface of a unit sphere. Higher dimensional sets can be similarly generated using formulæ (8.2–8.4).

To generate uniformly distributed point set on a unit sphere, we first generate uniformly distributed points on a 2D plane. Then they are mapped to the surface of the sphere. On the 2D plane, formula (8.4) simplifies to

$$
\left(\frac{k}{n}, \Phi_{p_1}(k)\right) \quad \text{for } k = 0, 1, 2, \ldots, n-1. \tag{8.5}
$$

The range of $\frac{k}{n}$ is $[0, 1)$, while that of $\Phi_{p_1}(k)$ is $[0, 1]$. For computer applications, a good choice of the prime p_1 is p_1 = 2. The evaluation of $\Phi_2(k)$ can be done efficiently with about $\log_2(k)$ bitwise shifts, multiplications and additions: no division is necessary. To generate directional vectors, or (equivalently) points on the spherical surface, the following mappings [SPAN69] is needed:

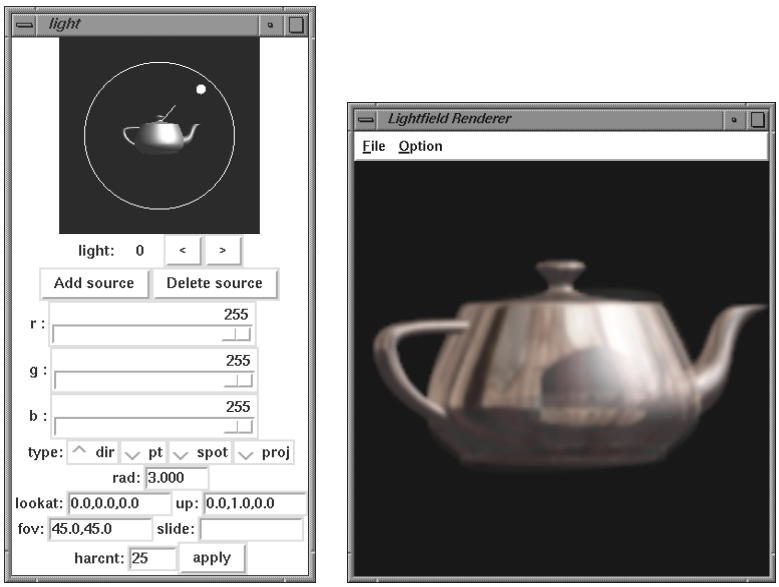
$$
\left(\frac{k}{n}, \Phi_p(k)\right) \mapsto (\phi, t) \mapsto \left(\sqrt{1-t^2}\,\cos\phi, \sqrt{1-t^2}\,\sin\phi, t\right)^T . \tag{8.6}
$$

The first, from $\left(\frac{k}{n}, \Phi_p(k)\right)$ to (ϕ, t), is simply a linear scaling to the required cylindrical domain, $(\phi, t) \in [0, 2\pi) \times [-1, 1]$. The mapping from (ϕ, t) to $(\sqrt{1-t^2}\cos\phi, \sqrt{1-t^2}\sin\phi, t)^T$ is z-preserving radial projection from the unit cylinder $C = \{(x, y, z) \mid x^2 + y^2 = 1 \ |z| \leq 1\}$ to the unit sphere.

Figure 8-11(c) shows the generated Hammersley points with $p_1 = 2$ on a sphere. Compared to the regular sampling (Figure 8-11(a)), the Hammersley point set is uniformly distributed without a perceptible pattern. Compared to the random sampling (Figure 8-11(b)), the Hammersley point set gives a pleasant, less clumped pattern. One evidence of uniformly distribution is that there is no way to tell where on the sphere are the poles or the equator. It has been recently found [CUI97] that mapping Hammersley points with base of 2 to the surface of a sphere gives the best uniformly distributed directional vectors among several common approaches.

8.1.5 A Light Field Viewer with Controllable Illumination

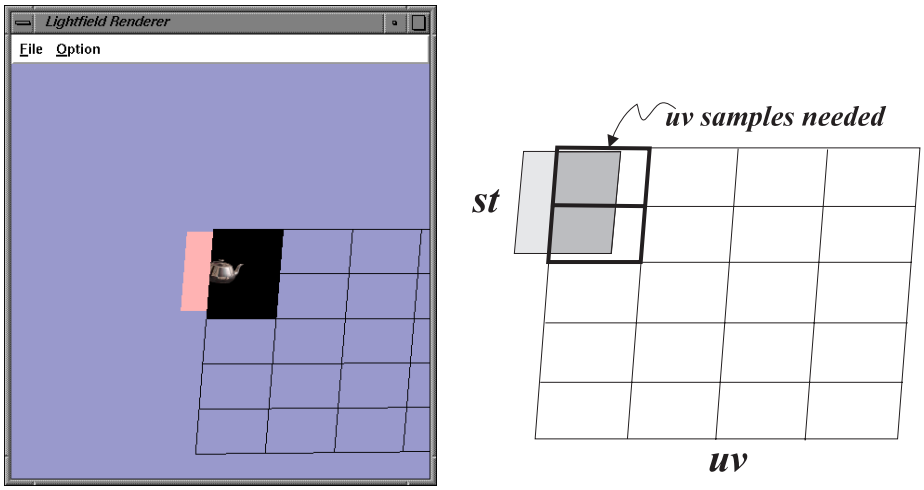
We have implemented an interactive image-based renderer, `slabview`, that displays the light field with controllable illumination. Figure 8-12 shows the interface of the viewer. The user can change the viewpoint by dragging inside the right viewing window. Through the left light source control panel, the user can control the direction, the intensity, the number and the type of the light sources.



(a) (b)

Figure 8-12: The interface of `slabview`. (a) The light source control panel, (b) the viewing window.

Figure 8-13 shows how a desired image is synthesized through drawing texture mapped polygons. In Figure 8-13(a) example, constant basis approach is used. The uv plane in the front is subdivided into several rectangles. Each rectangle is associated with a st image (the image of st plane viewed from the center of that rectangle). When the st plane (the smaller rectangle behind) overlays with uv plane, the intersection region (dark gray region in Figure 8-13(b)) will be the region that the light slab has recorded the radiances. In other word, we can see something. Therefore, we only need to draw texture mapped polygons that overlap with this intersection region, just like Figure 8-13(a). This will significantly improve the efficiency of the program.



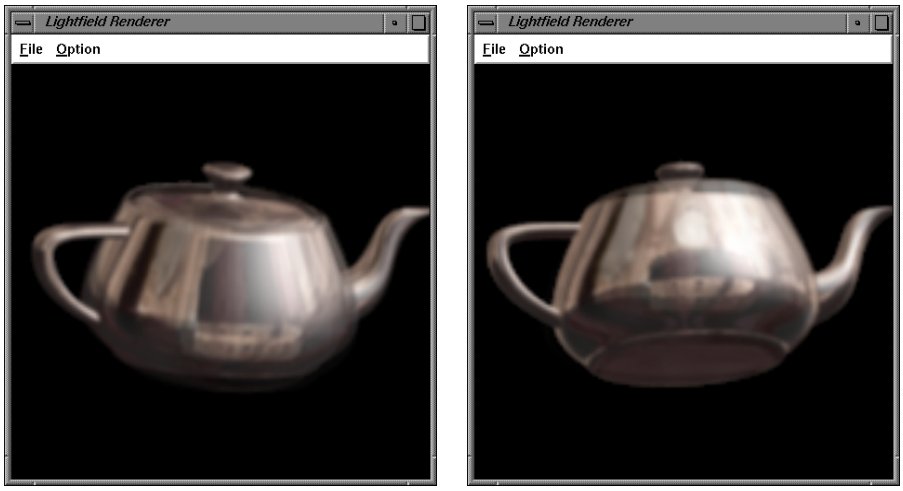
(a) (b)

Figure 8-13: Drawing only the necessary.

Whenever the user drags inside the right viewing window, suitable polygons are drawn and blended to give a smooth rotation. Since the texture mapping and blending are done by graphics hardware, rotation of the image-based object (or change of viewpoint) can be done in real time. Figures 8-14(a) and (b) show two frames from the rotation of an image-based teapot. Whenever the user drags the light source (small sphere) in the left light source control panel, pixel values are re-calculated using Equation 7.6 to re-render the image under the user-specified lighting condition. Note that the illumination in our system is *controllable*, as opposed to those uncontrollable approaches [BELH96, ZHAN98]. Only the views (st images) that are relevant (overlap with the gray intersection area in Figure 8-13(b)) are re-rendered. They are then drawn and blended by graphics hardware. Since the pixel value calculation is done purely by software, changing the illumination is slower than changing the viewpoint. The more the number of light sources used, the slower the

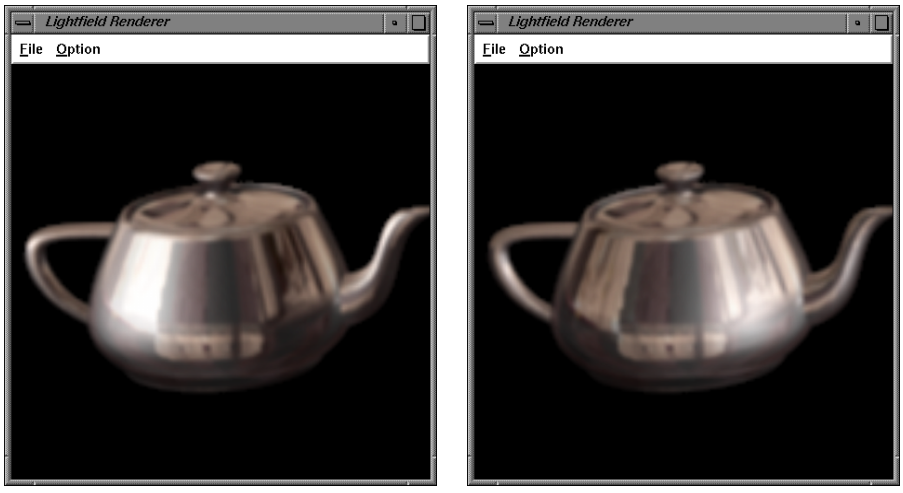
re-rendering is. Nevertheless, our prototype viewer can still re-render image-based scene illuminated by three to four light sources in real time. Figure 8-15 (a) and (b) show two frames from moving one light source from left to right.

Actually, images inside Figures 7-15 to 7-17 and Figures 7-19 to 7-21 are all generated by our prototype viewer. These results show that the concept of measuring pixel BRDF can support the illumination of variable lighting condition. If the reference images contain shadow, the re-rendered image will also include shadows (Figure 7-19).



(a) (b)

Figure 8-14: Rotation of an image-based teapot.



(a) (b)

Figure 8-15: Moving the light source from left to right.

8.2 Viewing Outward

An outward-viewing application is nearly the opposite of the inward-viewing one. Instead of looking at a fixed point in 3D, we "fix" the eye at a point in 3D and allow the eye to look around. The reason we use the quotes for the word "fix" is because the eye is not actually fixed but is allowed to move within a small region. In the case when we look at far scene, our eye can be assumed fixed, since the movement of our eyes is negligible relative to the distance from our eyes to the far scene. Figure 8-16 illustrates this action. The goal is to give an immersive feeling. One example application is in virtual reality. The CAVE [CN92] is a virtual reality application that allows the user to stand in a display chamber and looking around in order to give him/her an immersive feeling.

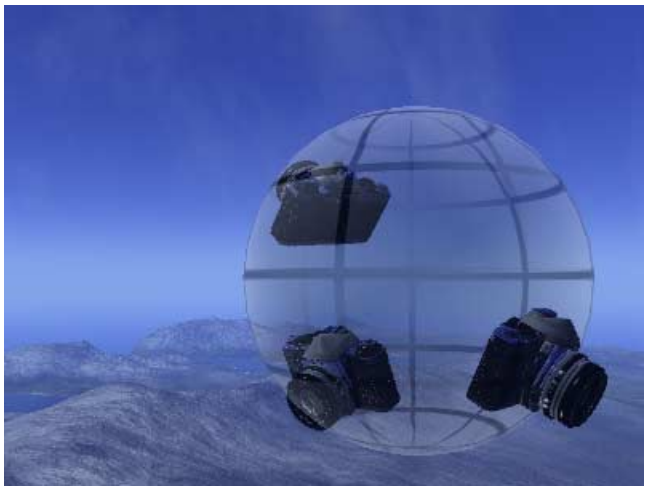


Figure 8-16: An outward-viewing example.

8.2.1 Panorama

Theoretically, the light slab organization can also be used from the outward-viewing application. In the inward-viewing applications, the st plane is usually smaller than the uv plane. However, in the outward-viewing applications, st plane has to be enlarged while the uv plane has to be shrinked. In the extreme case, uv plane shrinks to a infinitesimal point. Unfortunately, light slab representation is not very good for outward-viewing application because in most cases we want to look at far scenery. Even the st plane is enlarged to infinite size, a single light slab can only represent a field of view of 180 degree. To support a full 360 degree, at least two light slabs are needed.

A more efficient approach is to use *panoramic image*. It is an image of 360 degree as viewed

from one fix point in space. Figure 8-17(a) shows the structure of a spherical panorama. Note that a spherical panorama can record the radiances represented by a plenoptic sphere (see Section 6.1). However, storing a spherical image in computer is less natural. The image has to be either distorted or cut since a spherical surface is not developable [1]. Another type of panorama is the cylindrical panorama (Figure 8-17(b)). It composes of a cylinder and two discs. This type of panorama is more popular in computer graphics applications because a cylinder can be unfolded to a rectangular image without any distortion.

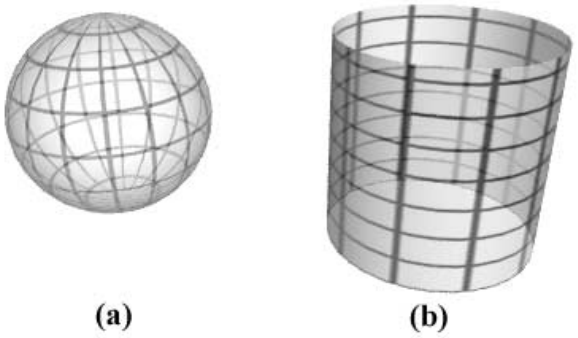


Figure 8-17: Panorama. (a) Spherical, (b) Cylindrical

For a virtual scene, a cylindrical panorama can be directly obtained by ray tracing. To shade a pixel, we fire a ray through each pixel on the cylinder. If raytracing capability is not available, six perspective images forming a cube (Figure 8-18) can also be used to resample the cylindrical panorama. Note this method is actually the environment mapping [GREE86a, GREE86b].

For a real scene, we can first capture the scene using a camera. That is capturing the scene as a set of perspective images. Each image should have a significant region of overlapping. Image registration can then be used to stitch [CHEN95a, SZEL97] them together to form a continuous panoramic image.

8.2.2 Illuminating Panoramic Image

Although a panoramic image is not an image with perspective or parallel projection, we still can record the BRDF of each pixel in the panoramic image. From now on, we focus on illumination of cylindrical panoramic image due to its efficiency and simplicity. However, the technique being described is not restricted to cylindrical panorama, but also applicable to spherical panorama.

[1] A developable surface is a surface that can be unfolded to a planar surface without infinite stretching or squeezing.

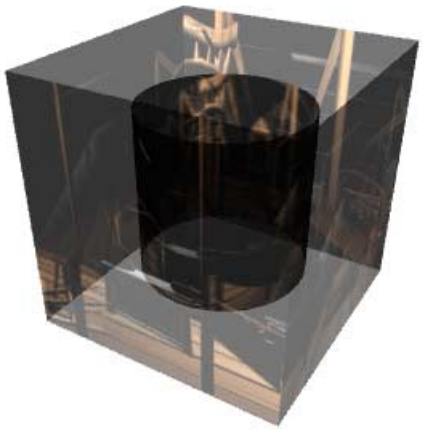


Figure 8-18: Environment mapped cube.

Again, the basic idea is to record the radiance passing through the pixel window under different illumination. One difference is that the viewpoint is always fixed for a panoramic image. That is, one vector parameter of the pixel BRDF, $\vec{V}$, is constant. The pixel BRDF reduces to a single pixel URDF (not a set). Therefore, recording pixel BRDF of a panorama is simpler than that for the light field, because we only need to record a single spherical function (URDF) for each pixel.

There is still one problem to overcome. In the case of the illumination of the light field, we record the set of pixel URDFs with a coordinate system relative to the pixel as in Figure 8-19(a). For panorama, if we still measure it on the pixel's frame like Figure 8-19(b), each pixel has a different coordinate system. That means, when querying a value in the URDF given a light vector, we need to transform the vector to the local coordinate system of each pixel before getting the answer. This is exactly the case of directional light source. Given a light vector of a directional light source, the light vector has to be transformed to the local coordinate system for finding the radiances. This is very inefficient when transformation has to be done for every pixel in the panoramic image. A more efficient approach is to use a common coordinate system for every pixel in the panorama as in Figure 8-19(c). The common coordinate system free us from transformation. For other types of light sources, the light vector for each pixel is still different. But at least it is efficient for directional lights.

8.2.3 A Panoramic Viewer with Controllable Illumination

An experimental panoramic viewer, `panoview`, that supports controllable illumination has been developed to verify the idea. Figure 8-20 shows the user interface of this program. The interface is

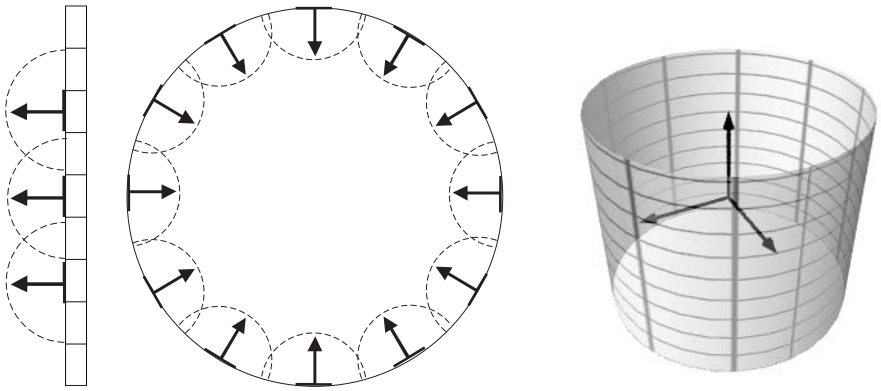
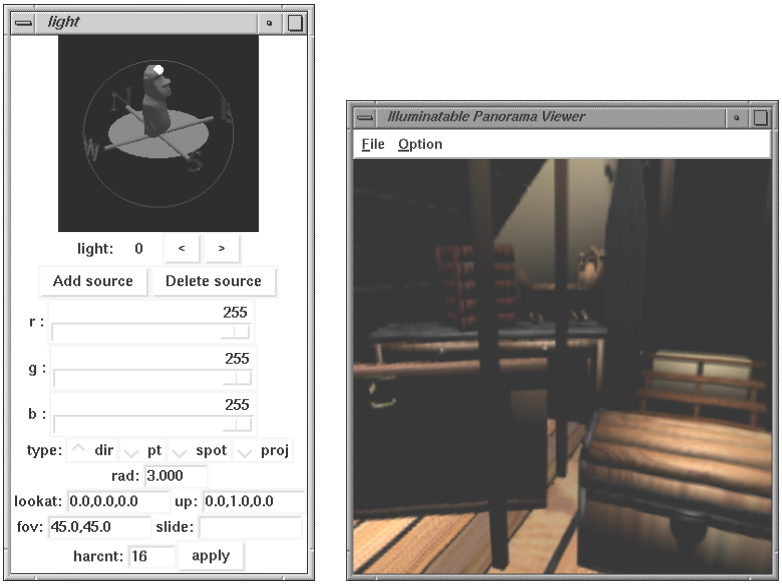


Figure 8-19: Coordinate systems. (a) Pixel coordinate system on a planar image, (b) Pixel coordinate system on cylindrical panoramic image, (top view) (c) Common coordinate system for all pixels in cylindrical panorama.

very similar to the one in Figure 8-12. But this time when the user drags inside the right viewing window, it pans to other viewing direction. The user can also zoom in and out the scene. Again the light sources are controlled by the light source control panel on the left.

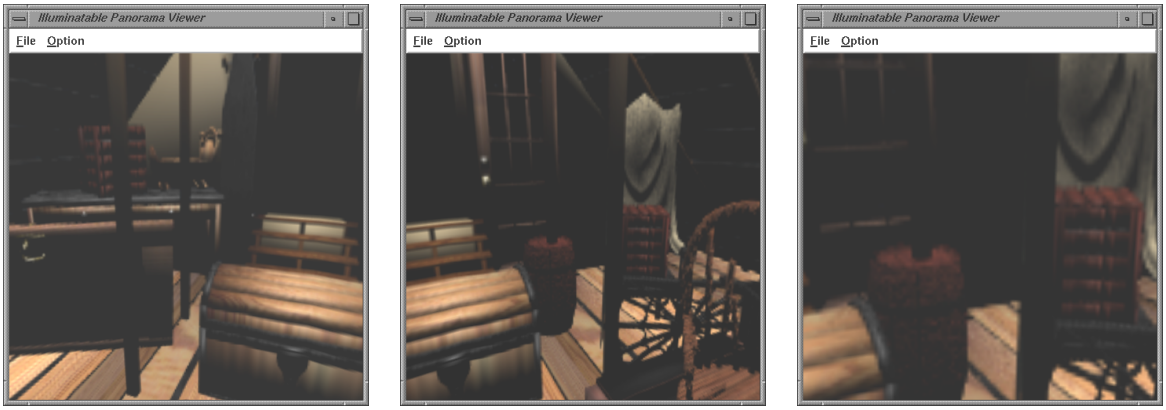
Figures 8-21(a) and (b) show two frames when panning the view from left to right while Figure 8-21(c) zoom in the view of Figure 8-21(b). To render the panorama, we map the cylindrical panorama to the surface of a cylinder. The texture mapped cylinder is then rendered by graphics hardware. Therefore, the panning and zooming can be done in real time. When the user changes the lighting, pixel values are calculated using Equation 7.6. This is again done purely by software. Figure 8-22 shows two frames from changing the direction of a light source in the image-based attic environment. Note how the illumination is done even no geometry is present. The original geometry-based attic scene requires about 2 minutes to render on SGI Octane. Using the image-based approach the scene can be rendered within a second. This demonstrates the potential of image-based rendering.

Figure 8-23(a) and 8-24(a) shows the unfolded cylindrical panoramas of a chessboard and an attic scenes respectively. The bottom three images are the perspective snapshots of the panorama, they can be obtained by warping the panorama. Figure 8-25 shows the same attic scene as in Figure 8-24, but illuminated by four spotlights, each with a different color. Note how the spotlight correctly illuminated the image-based attic. Again non-directional light sources illumination requires the depth information.



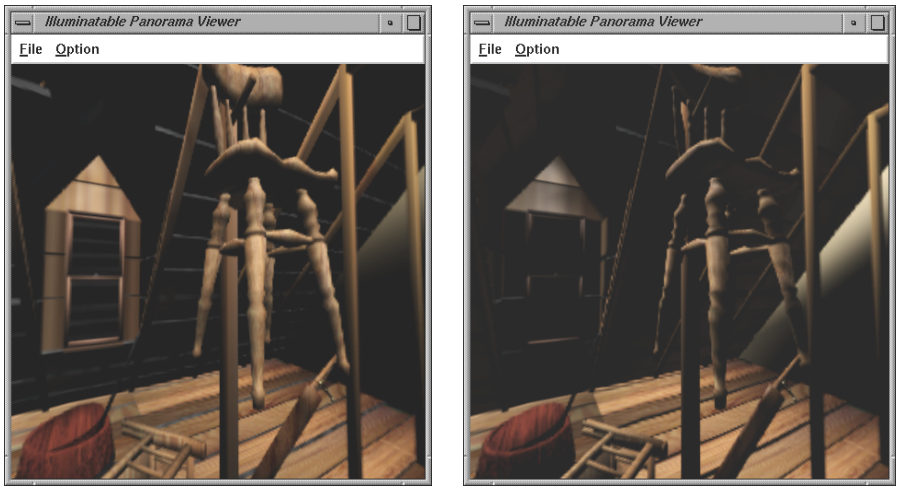
(a) (b)

Figure 8-20: A panoramic viewer with controllable illumination.



(a) (b) (c)

Figure 8-21: Panning and Zooming. (a) Original, (b) pan to right, (c) zoom in.



(a) (b)

Figure 8-22: Changing the lighting setup of a panoramic image.

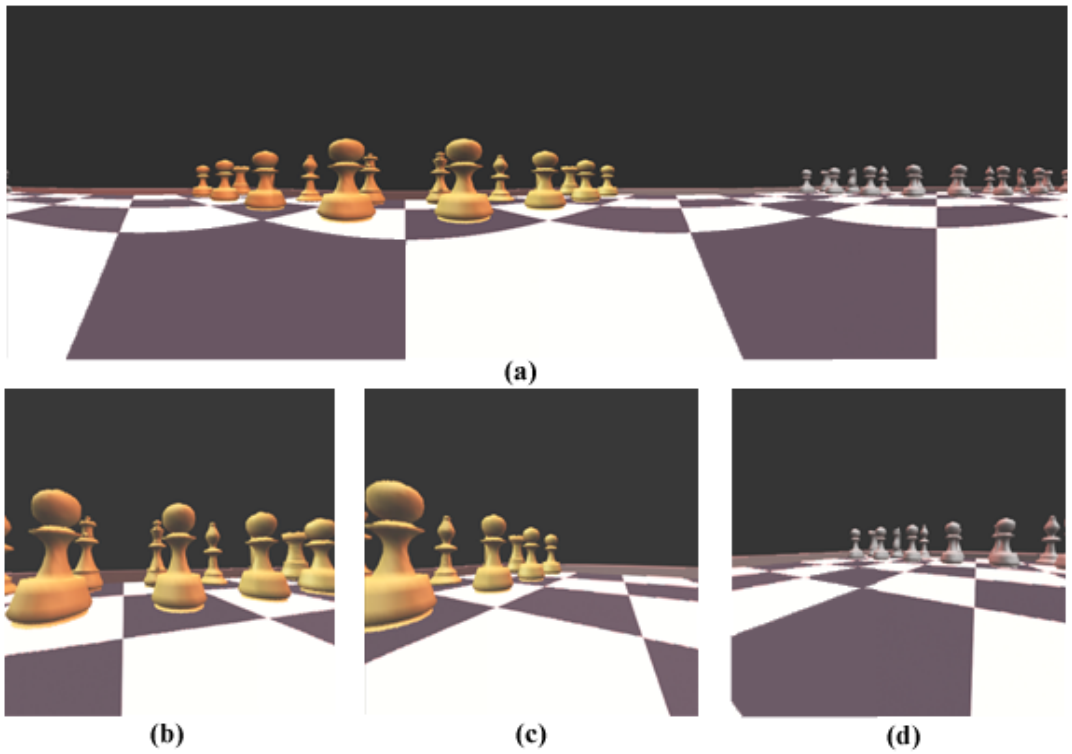


Figure 8-23: Chessboard.

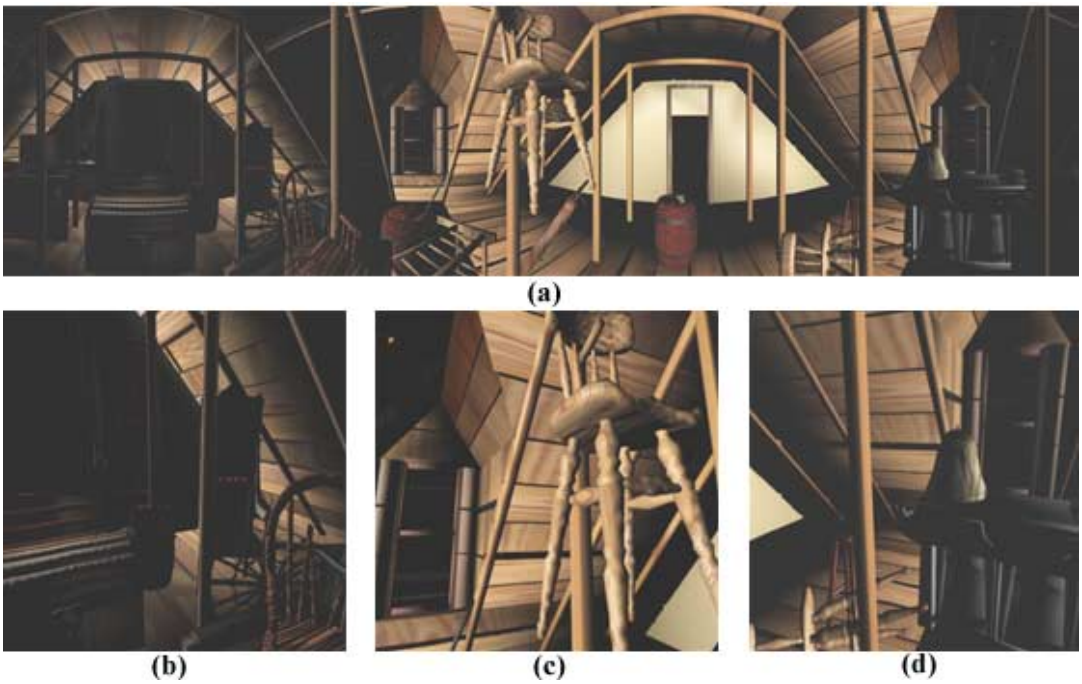


Figure 8-24: Attic.

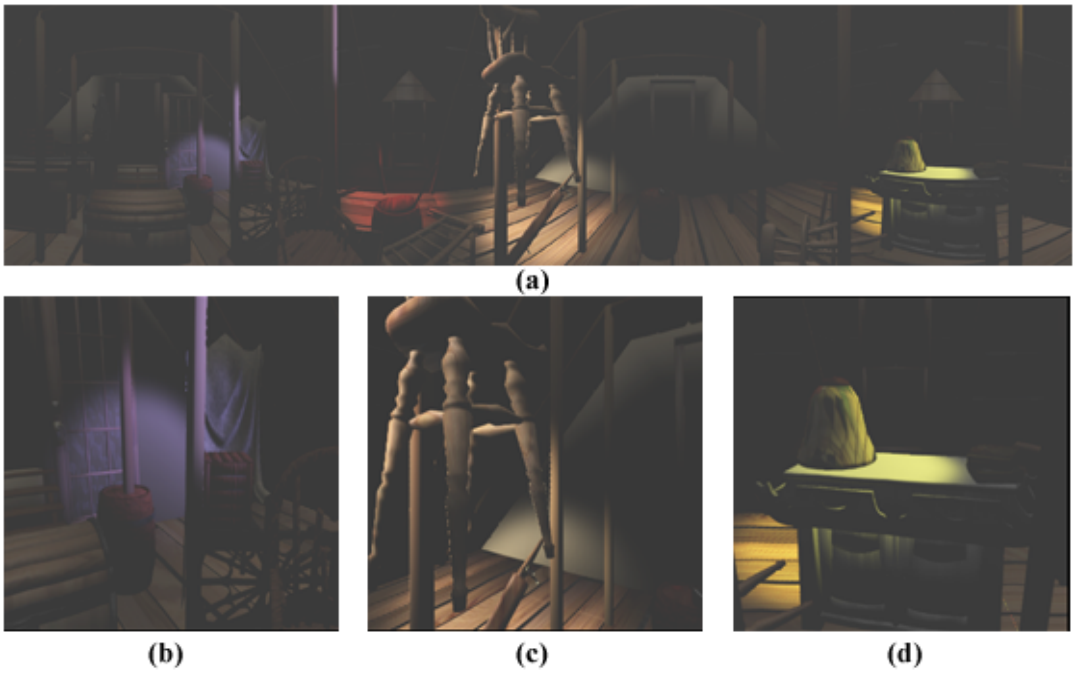


Figure 8-25: Attic illuminated by spotlights.

8.3 Summary

In this chapter, we have discussed how to apply the concept of measuring pixel BRDF to two major actions of image-based computer graphics, namely the inward-viewing and the outward-viewing actions. Although we have applied the idea on two image data representations which have substantial difference, there is not much problem in extending these two representations to include illumination. We believe the concept of pixel BRDF is general enough to be applied to other image representations, such as spherical light field [IHM97], as well.

Chapter 9

Compression

Storing the pixel BRDFs requires an enormous storage space. Assume we represent the pixel BRDF as a set of pixel URDFs. For a single pixel, if the URDF is sampled in the polar coordinate system with 20 samples along both the azimuth (θ_l) and zenith (ϕ_l) dimensions, there will be 400 floating point numbers stored for each pixel. A single view of a 256 $\times$ 256 image plane will require 100Mb of storage.

In this chapter, we will investigate various approaches to compression of pixel BRDF data. All compression techniques identify some forms of coherence among data in order to compress it. We will first explore the data coherence of the radiance values associated with a single pixel. Then, we will explore the data coherence between adjacent pixels. Utilizing these two types of data coherence, we can compress the pixel BRDFs with a compression ratio of about 100 to 1.
There are two main criteria in choosing a good compression scheme:

- High compression ratio.
- Fast decoding algorithm.

Both criteria are equally important. A scheme with high compression ratio but cannot decode quickly is not very useful for our application. Since the decoding has to be done purely by software, a fast decoding algorithm is necessary for an interactive application.

9.1 Coherence Within a Pixel

In the following discussions, we will concentrate on compressing sets of pixel URDFs, instead of pixel BRDFs. As we have mentioned in the previous chapter, storing pixel URDF allows the separation of the re-rendering process into two steps. The first step is done purely by software while the second step can be accelerated by graphics hardware, hence speedup the user interaction. Although we believe compressing the complete pixel BRDF will significantly improve the compression ratio, the decoding will be much slower since it cannot utilize the existing graphics hardware.

To represent the URDFs more efficiently, the tabular data is transformed to the frequency domain and quantization is performed to reduce storage. We have tested two types of transforms, spherical harmonic transform and discrete cosine transform.

9.1.1 Spherical Harmonics

Spherical harmonics [COUR53] are analogous to Fourier series, but in the spherical domain. Cabral *et al.* [CABR87] proposed the representation of BRDF using spherical harmonics. The work is further extended by Sillion *et al.* [SILL91] to model the entire range of incident angle. It is especially suitable for representing smooth spherical functions. Appendix A will give a more detail description of the spherical harmonics. In our approach, the viewing direction $\vec{V}$ for each pixel is actually fixed. Hence, the unidirectional function ρ can be transformed to spherical harmonics domain using the following equations directly, without considering how to represent a bidirectional function described by Sillion *et al.* [SILL91].

$$C_{l,m} = \int_0^{2\pi} \int_0^{\pi} \rho(\theta_l, \phi_l) Y_{l,m}(\theta_l, \phi_l) \sin\theta_l d\theta_l d\phi_l, \tag{9.1}$$

where

$$Y_{l,m}(\theta_l, \phi_l) = \begin{cases} N_{l,m} P_{l,m}(\cos\theta_l) \cos(m\phi_l) & \text{if } m > 0 \\ N_{l,0} P_{l,0}(\cos\theta_l)/\sqrt{2} & \text{if } m = 0 \\ N_{l,m} P_{l,|m|}(\cos\theta_l) \sin(|m|\phi_l) & \text{if } m < 0, \end{cases}$$

$$N_{l,m} = \sqrt{\frac{2l+1}{2\pi} \frac{(l-|m|)!}{(l+|m|)!}},$$

and

$$P_{l,m}(x) = \begin{cases} (1-2m)\sqrt{1-x^2} P_{m-1,m-1}(x) & \text{if } l = m \\ x(2m+1) P_{m,m}(x) & \text{if } l = m+1 \\ x\frac{2l-1}{l-m} P_{l-1,m}(x) - \frac{l+m-1}{l-m} P_{l-2,m}(x) & \text{otherwise.} \end{cases}$$

where the base case is $P_{0,0}(x) = 1$.

Functions $Y_{l,m}$'s are the basis functions (or spherical harmonics). Figure 9-1 shows the first few spherical harmonics. The first basis function is a sphere. It has equal magnitude in any direction, hence no directional preference. All other basis functions have directional preferences.

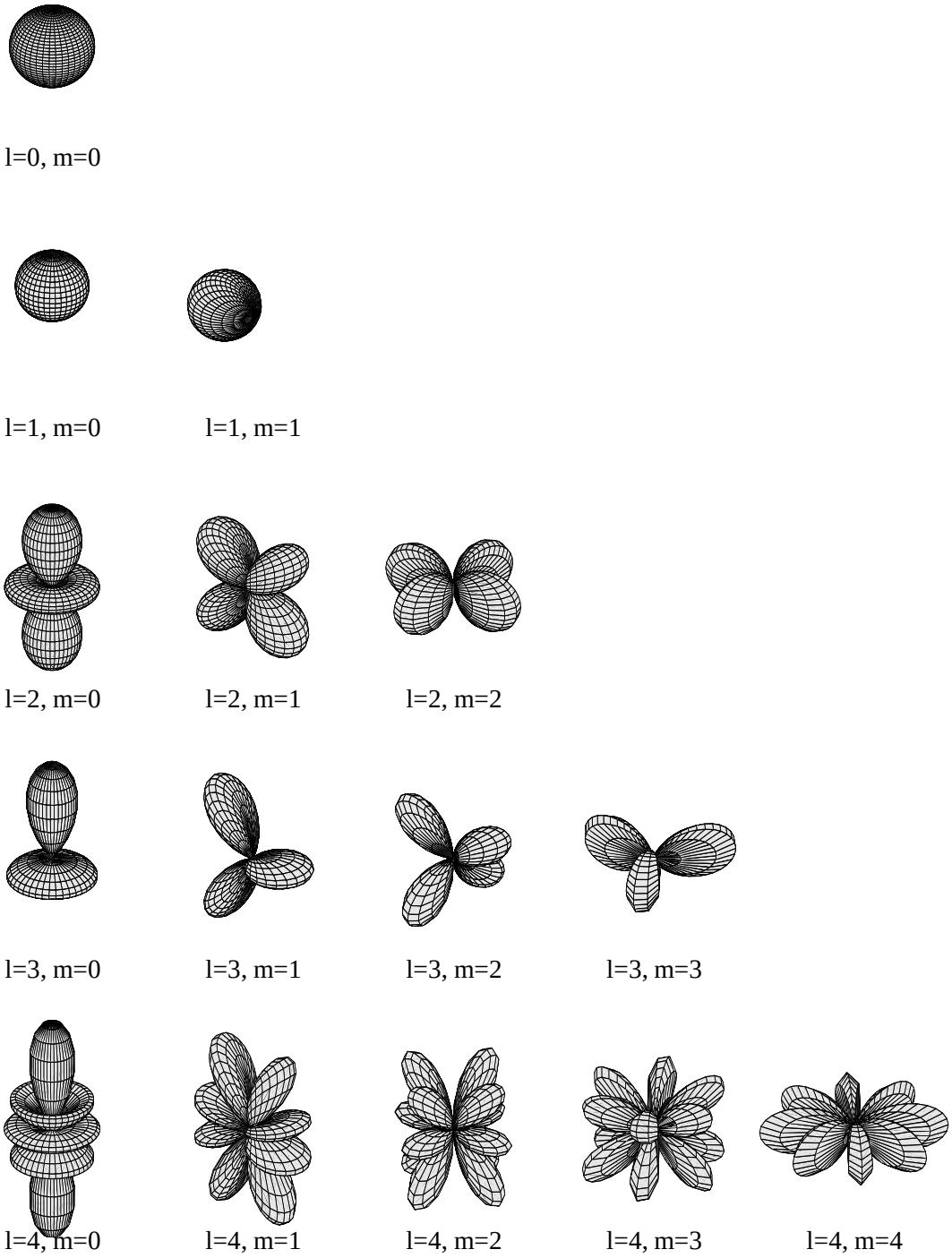


Figure 9-1: Spherical harmonics.

$C_{l,m}$'s are the coefficients of the spherical harmonics which are going to be stored for each pixel. The more coefficients are used, the more accurate the spherical harmonics representation is. Accuracy also depends on the number of samples in the (θ_l, ϕ_l) space. We found 9 to 16 spherical harmonic coefficients are sufficient in tested examples containing no shadow. More coefficients are needed to accurately represent scenes containing shadows.

To reconstruct the reflectance given the light vector (θ_l, ϕ_l), the following equation is solved for each pixel in each view.

$$\rho(\theta_l, \phi_l) = \sum_{l=0}^{l_{max}} \sum_{m=-l}^{l} C_{l,m} Y_{l,m}(\theta_l, \phi_l). \tag{9.2}$$

where $(l_{max})^2$ is the number of spherical harmonic coefficients to be used.

Figure 9-2 shows the original tabular reflectance distribution of a pixel on the left and its corresponding reconstructed distribution on the right. There are 1800 samples (30 along θ_l in the range $[0, \frac{\pi}{2}]$ and 60 along ϕ) in the left original distribution. The reconstructed distribution on the right is represented by 25 spherical harmonics coefficients only.

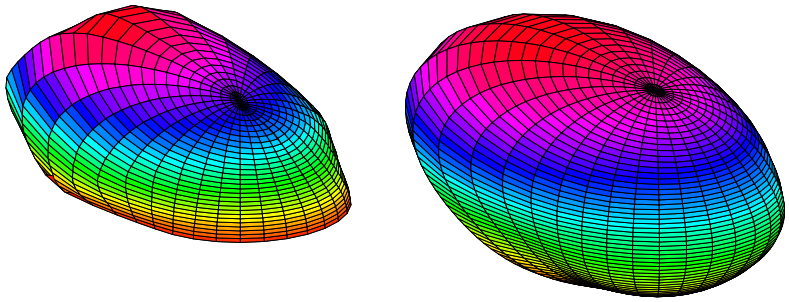


Figure 9-2: Original sampled (left) and reconstructed (right) distribution. Note the lower hemisphere of the reconstructed distribution is interpolated to prevent discontinuity.

The more coefficients are used, the more accurate the reconstructed images are. Figure 9-3 shows the visual difference of representing the same pixel URDFs using different number of spherical harmonic coefficients. From left to right, the number of coefficients used are 1, 4 and 25. All images are re-rendered under the *same* lighting condition. It seems that as the number of coefficients increases the teapot becomes more shiny. Since the recorded teapot is highly reflective, the correct teapot should be shiny. One interesting observation is that as the number of coefficient decreases, the shiny object becomes duller. It seems that the specular component is represented by the high-order coefficients. This can be explained by the shape of the spherical harmonics basis functions mentioned before. The

first basis function is a sphere with no directional preference (Figure 9-1). It actually captures the diffuse component in the illumination model. Other higher order basis functions contain directional bumps. The higher the order is, the sharper the bumps are. These high-order basis functions hence capture the directional specular component.

(a) (b) (c)

Figure 9-3: Specularity difference of using different number of spherical harmonic coefficients. (a) 1 coefficient, (b) 4 coefficients, (c) 25 coefficients.

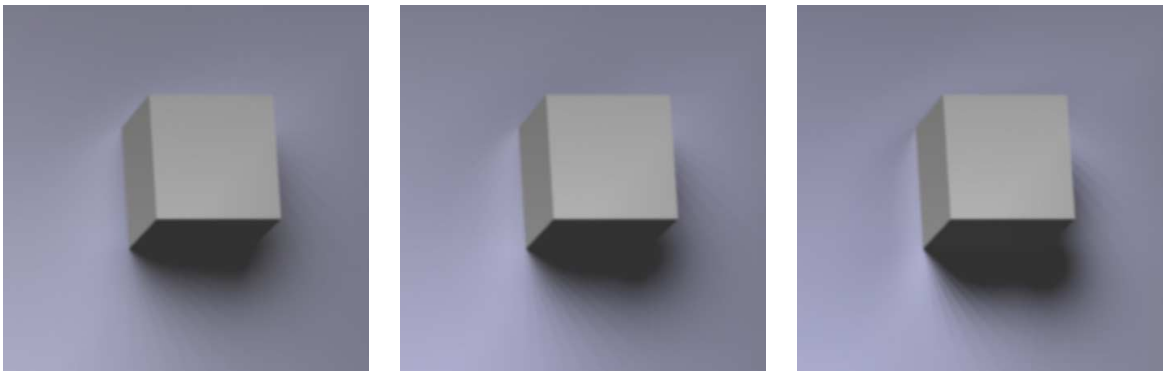
Another visual artifact when less coefficients are used is the smoothing-out of shadow. If the reference images contain shadow (especially the hard shadow), the URDFs are discontinuous. A discontinuous signal requires infinite number of spherical harmonic coefficients to represent. Of course, we can only afford finite number of coefficients. Any finite representation will certainly smooth out the signal. Figure 9-4 shows three reconstructed images using three different number of coefficients. As the number of coefficient increases from left to right, the reconstructed images become more accurate in term of shadow representation.

Using the example mentioned in the beginning of this chapter, a 256×256 image which originally requires 100 Mb of storage can now be compressed to 18.75 Mb if 25 spherical harmonics coefficients are used for encoding one URDF. The compression ratio is roughly 5 to 1.

9.1.2 Discrete Cosine Transform

Although spherical harmonics can efficiently represent smooth spherical functions, it is inferior in representing discontinuous function which is quite common if the scene contains shadow. This phenomenon motivates us to try another compression scheme.

The second compression scheme we have tested is the discrete cosine transform (DCT). One reason to choose DCT is that hardware DCT codec is becoming widely available. Same as before, we



(a) (b) (c)

Figure 9-4: Shadow difference of using different number of spherical harmonic coefficients. (a) 16 coefficients, (b) 25 coefficients, (c) 49 coefficients.

do not compress the four dimensional BRDFs. Instead, the two dimensional URDFs are compressed. Since the URDF is a spherical function, it is first mapped to a 2D disc (Figure 9-5), before applying the standard 2D discrete cosine transform to the disc image.

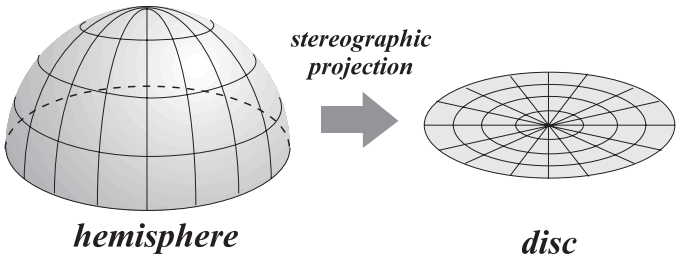


Figure 9-5: Mapping a hemisphere to a disc.

To map a spherical function to a plane, the mapping should be done in two passes, namely, one for the upper hemisphere and one for the lower half. The mapping from a hemisphere to a disc is done by stereographic projection [HILB52]. To project the lower hemisphere, the point of projection $\dot{C}$ is first placed at the pole of upper hemisphere and the plane is placed underneath the lower hemisphere (Figure 9-6). A point $\dot{S}$ on the hemisphere is mapped to point $\dot{P}$ on the plane by firing a ray from $\dot{C}$ through point $\dot{S}$ and intersects the plane at point $\dot{P}$. The upper hemisphere can be mapped to plane similarly. The polar coordinate (θ_l, ϕ_l) on a hemisphere is mapped to the 2D coordinate (x, y), within a unit square by

$$x = \frac{1}{2}[\tan(\theta_l/2)\cos(-\phi_l) + 1] \tag{9.3}$$

$$y = \frac{1}{2}[\tan(\theta_l/2)\sin(-\phi_l) + 1] \tag{9.4}$$

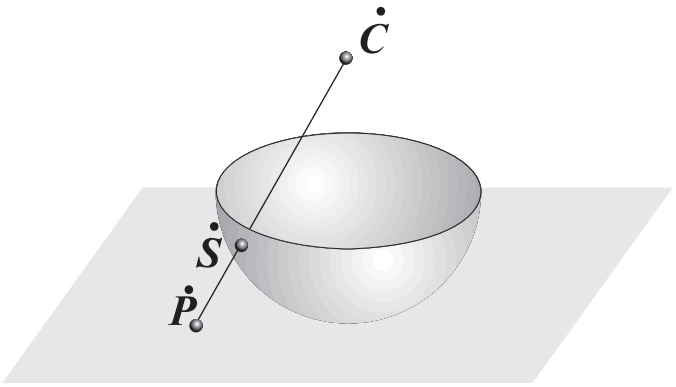


Figure 9-6: Stereographic projection.

The resultant disc image after projecting the upper hemisphere of the URDF of an example pixel is shown in Figure 9-7(a). The example pixel is extracted from the test scene in Figure 9-8(a). In Figure 9-7(a), the white region near the image center indicates the specular highlight. The polygonal black hole on the right is due to the shadow cast by the box in Figure 9-8(a).

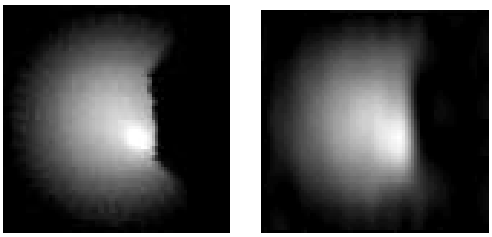
Once the spherical function is projected to 2D image, discrete cosine transform (DCT) can be applied to transform the image and the resulting DCT coefficients are zonal sampled and quantized. The $N \times N$ cosine transform matrix, $c(i,j)$ is defined as,

$$c(i,j) = \begin{cases} \frac{1}{\sqrt{N}}, & i = 0, 0 \le j \le N-1, \\ \sqrt{\frac{2}{N}}\cos\frac{\pi(2j+1)i}{2N}, & 1 \le i \le N-1, 0 \le j \le N-1. \end{cases} \tag{9.5}$$

Figure 9-7 shows the images before (a) and after (b) the quantization in DCT domain. Only 64 coefficients are retained for the image in Figure 9-7(b), while the image in Figure 9-7(a) is represented by 50×50 floating point data.

9.1.3 Comparison

Figure 9-8 visually compares the reconstructed images of different compression schemes. Figure 9-8(a) shows the test scene containing a box which cast shadow on a plane. The square region in Figure 9-8(a) is enlarged for visual comparison. The ideal result is generated by looking up the original tabular BRDFs (Figure 9-8(b)). Note that the hard shadow is preserved. After the stereographic



(a) (b)

Figure 9-7: Before (a) and after (b) quantizing the disc image in frequency domain. Original data in (a) is represented by 50 x 50 floating point data. The number of coefficients to represent the image (b) is 64.

projection, some errors are introduced. It is because the mapping process is actually a resampling process. Blurring is found around the shadow in Figure 9-8(c). This error can be reduced by increasing the resolution of the disc image, *i.e.* increasing the number of samples. However, the storage size will also be increased. Figure 9-8(d) shows the reconstructed image generated from data compressed using spherical harmonics. The error in this image is purely due to the quantization taken place in the spherical harmonic domain. Figure 9-8(e) shows the reconstructed image generated from data compressed using DCT. The errors in this image include quantization error in frequency domain and the resampling error during stereographic projection.

In order to have a fair comparison, equal number of coefficients (64 floating point coefficients) are used to compress the data in both compressed cases (Figures 9-8(d) & (e)). Comparing Figure 9-8(d) to Figure 9-8(e), the image generated from DCT is noisier than that of spherical harmonics. However, the shadow in the image generated from DCT is a better approximation of the true shadow in Figure 9-8(b). The sharp corner of the shadow becomes a round corner in the case of spherical harmonics, while the corner is still observable in the case of DCT. This is also confirmed by the RMS of error statistics. The RMS of error of image generated from spherical harmonic data is 0.1043 while that of image generated from DCT data is 0.0865. From this experiment, DCT compression scheme is preferred if the scene contains hard shadows and a close approximation to the true image is needed. On the other hand, spherical harmonics is preferred if the scene contains not much hard shadow and a pleasant (smooth) visual result is a main concern.

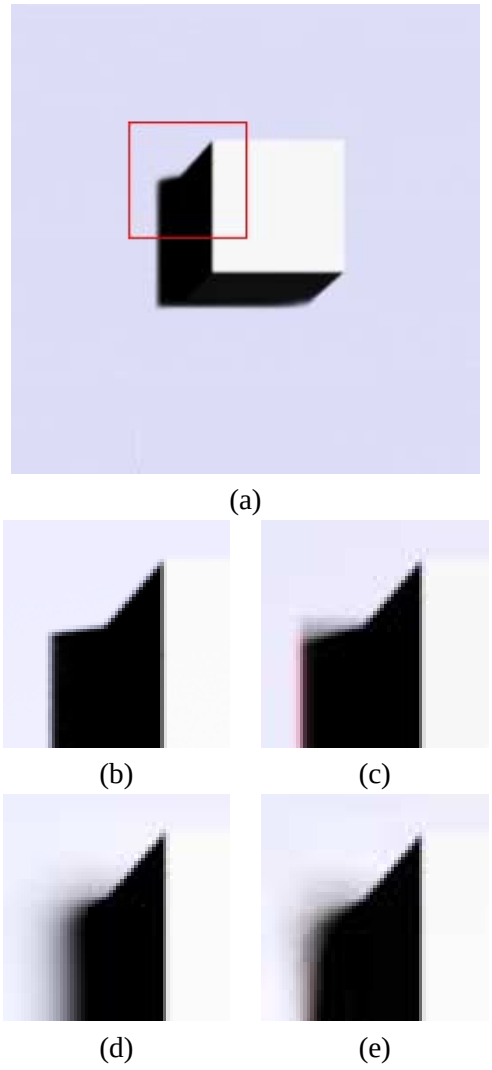


Figure 9-8: Visual comparison of reconstructed images. (a) Test scene. (b) Image generated using original tabular BRDFs. (c) Result after projecting spherical function to a disc, also uncompressed. RMS(err.)=0.0979. (d) Result from data compressed using spherical harmonics. RMS(err.)=0.1043. (e) Result from data compressed using DCT. RMS(err.)=0.0865.

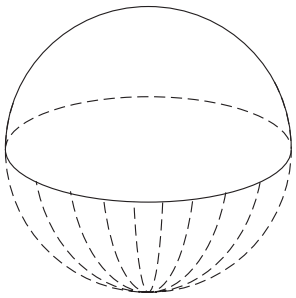


Figure 9-9: The boundary value along the equator is linearly interpolated to prevent equatorial discontinuity in the sampled BRDF.

9.1.4 Preventing Discontinuity

Truncating the spherical harmonic series or discrete cosine series gives persistent Gibb's ringing artifacts. One source of discontinuity is the incomplete sampling of light directions (boundary discontinuity). Incomplete sampling is sometimes necessary for fast scene capture. From our experience, there is no need to sample the whole range of θ_l, *i.e.*, $[0, \pi]$. Usually the range $[0, \frac{\pi}{2}]$ is sufficient. Zeroing all the unsampled entries introduces discontinuity along the equator of the sampling sphere. To avoid this sharp change, the boundary value along the equator is linearly interpolated to a constant value at the south pole (see Figure 9-9 and the right diagram in Figure 9-2). Another source of discontinuity is shadowing (Figure 9-7(a)), which is unavoidable. Hard shadows will be smoothed out if represented by a finite sum of harmonics (Figure 9-4(a), (b) & (c)).

9.2 Coherence Among the Pixels

Although DCT may produce more accurate result in term of root-mean-square error, it generates more noisy image which is annoying. On the other hand spherical harmonic transform provides more pleasant imagery result even though the error is larger. For the computer graphics applications, we prefer to use spherical harmonics transform to compress the pixel URDFs. From now on, we will only discuss how to further compress the data which have been compressed by spherical harmonic transform.

Up to this moment, we only utilize the data coherence within a single pixel BRDF. We have not yet utilized the coherence between data of adjacent pixels.

9.2.1 Vector Quantization

By investigating the spherical harmonic coefficient vectors of adjacent pixels, we find that the values of the coefficients of the neighbor pixels are usually very close. That means we can compress further by utilizing this coherence.

One approach to do so is to use *vector quantization* (VQ) [GERS82]. The basic idea of vector quantization is like the following. To compress the data, say n vectors, we first find out a set of k representative vectors (this vector set is usually called the *codebook* and the representative vector is called the *code vector*), such that $k \ll n$. Then we represent each of the n vectors by the closest code vector inside the codebook and memorize only the index of that code vector. The final compressed data only contains the codebook (k code vectors) and all indexes. Since k is much smaller than n, hence we can compress.

The main problem is how to find the k representative code vectors. There are many algorithms proposed. We have chosen a well-known algorithm, known as the LBG vector quantizer (LBG stands for Linde, Buzo and Gray [LIND80]) or k means algorithm, due to its simplicity. Figure 9-10 shows the LBG algorithm.

Given the size k of the codebook, the algorithm first randomly selects k vectors as the initial code vectors. In each iteration, the algorithm subdivides the n sample vectors into k sets based on the *distance* between the sample vector and the code vectors. The mean of the vectors inside each set is then chosen to be the code vector for the next iteration. The process of finding the mean vectors continues until the convergence occurs.

Figure 9-11(b) shows the reconstructed image after using vector quantization. Figure 9-11(a) shows the reconstructed image if no vector quantization is used. The major difference is that inside the reconstructed image from VQ compressed data, there exists contours in the regions of smooth ramp (circled in Figure 9-11(b)). Note this is the common artifact of compressing images using VQ. One way to improve it is to increase the size of the codebook.

Using VQ, we can significantly reduce the data size. The usual compression ratio which still preserves an acceptable quality is 10 to 1.

Step 1: Select k initial representative vectors. A good choice is to choose them randomly from the n vectors.

$$C_i^1 = X_{\text{random}}$$

where C_i^1 is the i-th code vector out of k initial code vectors,
$1 \leq i \leq k$,
X_{random} is the a sample vector chosen randomly from the n sample vectors.

Moreover, we also set up k vector sets $\mathbf{S}_i$ and initial each set to an empty set.

Step 2: In the m-th iteration, assign the sample vector X_l to the the set $\mathbf{S}_i$,

$$\mathbf{S}_i = \mathbf{S}_i \cup X_l\,,$$

if

$$\|X_l - C_i^m\| < \|X_l - C_j^m\|,$$

for all $i \neq j$.

Step 3: Update the code vectors to the means of the sample vectors inside the vector set $\mathbf{S}_i$,

$$C_i^{m+1} = \frac{1}{|\mathbf{S}_i|} \sum_{X \in \mathbf{S}_i} X,$$

where $|\mathbf{S}_i|$ is the number of vector in the set $\mathbf{S}_i$.

Step 4: Goto Step 2 until convergence is achieved.

Figure 9-10: Algorithm of LBG vector quantizer

(a) (b)

Figure 9-11: Contour artifact of VQ compression. (a) Without VQ compression, (b) with VQ compression.

9.3 Summary

In this chapter, we have discussed one practical aspect in using pixel BRDFs, the compression. To compress the URDF associating with a single pixel, we have tested two methods, namely the spherical harmonic and the discrete cosine transforms. Spherical harmonic transform usually gives smooth and pleasant imagery result, although the re-rendered images may not be as accurate as that compressed by discrete cosine transform. We further compress the data using vector quantization to utilize the data coherence among the adjacent pixels. After applying a series of compression algorithms, we can achieve an overall compression ratio of about 100 to 1.

Chapter 10

Conclusions and Future Directions

Anyone who attempts to simulate complex physical phenomena soon realizes that literal simulation is far beyond the capabilities of today's hardware or software.

— James F. Blinn, 1988.

Geometry-based computer graphics has been practiced for a long period of time in the history of computer graphics. By approximating the real world or imaginary world using geometry models, geometry-based computer graphics gives us a visual experience of the world being modeled. Geometry representation contains not just enough information to provide us the visual experience, but also extra information that human may not perceive. In principle, a goal of geometry-based time-critical modeling and rendering is to minimize or simplify this extra (visually imperceptible) information in order to speed up the simulation process. Ideally, the graphics system should contain no extra information which slows down the simulation if the system's only goal is to provide the visual experience.

Image representation can be thought as an extreme case that contains only information to give us the visual experience and no extra information is present. Instead of modeling the world, image-based computer graphics directly models the radiance energy impinging on the human eye. Therefore, it fulfills what the human perception needed without storing any extra information.

10.1 Synopsis

In this thesis, we have contributed a collection of concepts and algorithms for speeding up the modeling and rendering. Firstly, we have introduced a new geometry-based simplification algorithm that generates simplified triangular meshes directly from the volume data. The flexibility in partitioning the volume allows further simplification of the triangular mesh. The algorithm has the ability of generating isosurface representation in multiple resolutions which can be used for level-of-detail rendering. It is a fast heuristic algorithm, rather than a path to a strict optimum. Instead of the optimality of the generated mesh, the speed of the algorithm is our main concern.

Then we proposed a new concept of measuring the apparent BRDF of a pixel in the field of image-based time-critical modeling and rendering. Without the geometry representation, we no longer have

access to the geometry models and surface properties. Hence no way to learn the surface reflectance which determines the amount of reflected radiance from the surface element. The abstraction of measuring pixel BRDF provides us the fundamental model in image-based computer graphics which can be manipulated to synthesize image viewed from any desired direction under any desired illumination. To verify the concept, we apply it to two major image representations. Without much difficulty, both of them can be extended to include illumination. Finally, we also provide a series of compression schemes which compress the huge pixel BRDF data to a manageable size.

In the rest of this section, we will list the pros and cons of both the geometry-based and image-based approaches.

10.1.1 Pros and Cons of Geometry-based Approach

Geometry-based computer graphics is a *computation-intensive* approach. Object/scene is represented in a very compact form, geometry representation. However, rendering of geometry models usually requires more computation than that in image-based computer graphics. More importantly, the rendering time depends on the scene complexity. This dependence makes the real-time rendering of arbitrarily complex scene nearly impossible. No matter how fast the rendering softwares and hardwares are, the scene still can be complex enough to slow down any geometry-based modeling and rendering algorithms.

Modeling is also a major problem of geometry-based computer graphics. Most of the early geometry models are constructed by hand. Even though the three-dimensional digitizers are becoming more popular, constructing the geometry representation of an object/scene is still labor intensive. The geometry models acquired from these digitizers usually contain noise and are in high resolution format. They required further modification and simplification. Moreover, not all objects can be scanned due to their size, weight and rigid location. Imaginary objects/scenes still have to be constructed manually. Therefore, modeling will still be one of *time-consuming* and *labor-intensive* step in the geometry-based computer graphics.

Besides these disadvantages of geometry-based computer graphics, geometry representation provides extra information that can be used in *simulation* (not just visual simulation). For instance, a computer program can be used to physically simulate the air dynamics surrounding the modeled wing of a flight. In this case, the visual result becomes a visualization tool, not the final goal. In entertainment industry, if the computer-animated movie characters are geometrically represented, the

geometry models can be directly (or with little modification) used for toy manufacturing.

10.1.2 Pros and Cons of Image-based Approach

The emergence of image-based computer graphics is due to the demand of real-time display of arbitrarily complex scene. The approach detaches the dependency of the modeling and rendering time on the scene complexity. Rendering speed is now only dependent on the resolution of the image representation. Image-based approach successfully transforms an infinite problem (infinite scene complexity) to a finite problem (finite image resolution). Once the computer power and capacity reach a threshold, image-based computer graphics will allow us to display arbitrarily complex scene in real time due to the finite nature of the problem.

Image-based computer graphics is a *data-intensive* approach. It needs more memory, more storage and more bandwidth in order to store and transfer the image data. It is our belief that *there is still space to improve the storage and bandwidth of current computer systems while the speed of computer processor tends to a limit in the near future.*

Modeling real world object/scene in image-based computer graphics is usually easier since modeling becomes taking photographs. However, object/scene not exist will still need the help of geometry-based computer graphics. One more problem of image-based modeling is the *flexibility*. For example, if two objects are being modeled, we have to make a decision whether to model them in one single image representation or to separate them into two image representations. The former will reduce the storage but remove the flexibility, since we cannot change the relative distance of these objects. If we choose the later one, we can have more flexibility in controlling their relative position. But the later approach will also imply that the rendering time will associate with the scene complexity again.

Since the image representation only models the radiance impinging on our eyes and no geometry model exists, there is no way to perform simulation as in the geometry-based approach. Therefore, image-based computer graphics can only be used for providing the visual experience.

10.2 Future Work and Discussions

10.2.1 Hybrid Approach

As discussed above, we cannot say any one of the two approaches is superior. Instead, we believe both approaches will continue to be used in different computer graphics systems. A hybrid approach

will probably become more popular in the future. There are already some graphics systems developed using both approaches. Nimeroff *et al.* [NIME96] accelerated the radiosity software, *RADIANCE* [WARD94], using depth image with radiance values. In another application, Lengyel and Snyder [LENG97] applied the image warping techniques to warp a rendered image fragment in order to make it looks affinely transformed. Their idea is to substitute the portion of the image with a previously rendered image fragment and try not to render that portion unless the error exceeds the given threshold. This image caching approach significantly improves the rendering speed of the animation sequences.

The mentioned systems are still geometry-based systems in nature. They only use image as a temporary representation of the geometry object. We believe that in the future there will be an increase in modeling object/scene with image representations. Hence a hybrid system, which renders image-based entities as well as geometry-based entities, is needed.

10.2.2 View Dependence

View-dependence simplification techniques [LUEB97, HOPP97, XIA96] has been shown useful in representing portion of the object by dense mesh while the rest of the object are represented by coarse mesh. They have an application in speeding up the rendering while preserving the image quality. The well known time-consuming problem of the radiosity applications will be benefited by applying the view-dependence techniques.

The geometry-based simplification technique proposed in Part I is a view-independent algorithm. No viewing information is used in guiding the simplification process. Only the geometry complexity of the enclosed isosurface is used. We believe that a view-dependent approach will further improve the algorithm in term of triangle count. However, the algorithm will be complicated to support the feature of selective refinement. Moreover, a specially designed renderer is also needed.

10.2.3 Capturing Real Life Data

We have demonstrated the usefulness of measuring pixel BRDF for synthetic data. The next step will be the capturing of pixel BRDF from the real life data. Developing a capturing system for real world object/scene is a challenge. It relies on many computer vision techniques which are sensitive to noise.

One of the important information in measuring pixel BRDF is the direction information, both the light and viewing vectors. A method that can extract accurate direction information is needed in order

to prevent blurry image result. Standard pose estimation techniques can be used to extract the pose of the camera. For the light vector, a camera can be tied with the spotlight to extract the direction of the light source.

With extra depth information, the image-based object/scene can be correctly illuminated with light source other than the directional light. The depth information can be converted from the correspondence information which is able to be extracted from images by employing various vision techniques [BESL88, BOLL87].

Another problem is our current approach is that we need a dense image set in order to accurately record the pixel BRDF. Acquiring a dense image set from real life data is labor intensive. Therefore, one future direction is to construct the pixel BRDF using sparse image set.

10.2.4 Global Illumination

In this thesis, we proposed a local illumination model (Equation 7.6) for re-rendering of image-based object/scene. The natural extension is to design a global illumination model for the image-based object/scene.

The basic difference between the local and global illumination models is that global illumination accounts for the radiance contribution of each element in the environment. A simple extension is to treat every element in the environment as a light source and sum their contributions using the Equation 7.6. However, this approach will be prohibitively slow. Moreover, occlusion is another missing information in image-based approach. Without it, the final radiance calculated will not be correct.

Global illumination in image-based computer graphics is a difficult problem since one of the major component, form factor [COHE93, SILL94], of the radiosity equation is missing. Form factor is basically a geometry factor. Without the geometry model, there is no enough information in any projected images to allow us to calculate the crucial component. One possible solution is to calculate the form factor using the depth or the correspondence information extracted from images.

Bibliography

[ADEL91] ADELSON, E. H. AND BERGEN, J. R. The plenoptic function and the elements of early vision. In LANDY, M. S. AND MOVSHON, J. A., editors, *Computational Models of Visual Processing*, chapter 1, pp. 3–20. MIT Press, 1991.

[AKEL93] AKELEY, K. Reality engine graphics. In VALASTYAN, L. AND WALSH, L., editors, *Proceedings of the Annual Conference on Computer Graphics*, pp. 109–116, New York, NY, USA, August 1993. ACM Press.

[BEIE92] BEIER, T. AND NEELY, S. Feature-based image metamorphosis. In CATMULL, E. E., editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pp. 35–42, July 1992.

[BELH96] BELHUMEUR, P. N. AND KRIEGMAN, D. J. What is the set of images of an object under all possible lighting conditions. In *IEEE Conference on Computer Vision and Pattern Recognition*, 1996.

[BENT75] BENTLEY, J. L. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.

[BENT83] BENTON, S. A. Survey of holographic stereograms. In *Processing and Display of Three-Dimensional Data*, volume 367, 1983.

[BESL88] BESL, P. J. Active optical range imaging sensors. In SANZ, J. L. C., editor, *Advances in Machine Vision: Architectures and Applications*. Springer-Verlag, 1988.

[BLIN76] BLINN, J. F. AND NEWELL, M. E. Texture and reflection in computer generated images. *Communications of the ACM*, 19(10):542–546, October 1976.

[BLIN78a] BLINN, J. F. *Computer display of curved surfaces*. Ph.d. thesis, University of Utah, 1978.

[BLIN78b] BLINN, J. F. Simulation of wrinkled surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, volume 12, pp. 286–292, August 1978.

[BLOO97] BLOOMENTHAL, J., editor. *Introduction to Implicit Surfaces*. Morgan Kaufmann, 1997.

[BOLL87] BOLLES, R. C., BAKER, H. H., AND MARIMONT, D. H. Epipolar-plane image anlysis: an approach to determining structure from motion. *International Journal of Computer Vision*, 1:7–55, 1987.

[BUSB60] BUSBRIDGE, I. W. *The Mathematics of Radiative Transfer*. Cambridge University Press, 1960.

[CABR87] CABRAL, B., MAX, N., AND SPRINGMEYER, R. Bidirectional reflection functions from surface bump maps. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pp. 273–281, July 1987.

[CASE95] CASE, J. Wall street's dalliance with number theory. *SIAM News*, pp. 8 – 9, December 1995.

[CATM74] CATMULL, E. E. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Dept. of CS, U. of Utah, December 1974.

[CHEN93] CHEN, S. E. AND WILLIAMS, L. View interpolation for image synthesis. In KAJIYA, J. T., editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pp. 279–288, Aug. 1993.

[CHEN95a] CHEN, S. E. QuickTime VR - an image-based approach to virtual environment navigation. In Computer Graphics *Proceedings, Annual Conference Series, SIGGRAPH'95*, pp. 29–38, August 1995.

[CHEN95b] CHEN, S. E. AND MILLER, G. S. P. Cylindrical to planar image mapping using scanline coherence. United States Patent number 5,396,583, March 7 1995.

[CHIU94] CHIU, K., SHIRLEY, P., AND WANG, C. Multi-jittered sampling. In *Graphics Gems IV*, pp. 370–374. AP Professional, 1994.

[CLAR76] CLARK, J. H. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, October 1976.

[CN92] CRUZ-NEIRA, C., SANDIN, D. J., DEFANTI, T. A., KENYON, R. V., AND HART, J. C. The cave: Audio visual experience automatic virtual environment. *Communications of the ACM*, 35(6):65–72, June 1992.

[COHE93] COHEN, M. F. AND WALLACE, J. R. *Radiosity and Realistic Image Synthesis*. Academic Press Professional, San Diego, CA, 1993.

[COHE96] COHEN, J., VARSHNEY, A., MANOCHA, D., AND TURK, G. Simplification envelopes. *Computer Graphics*, 30:119–128, 1996.

[COOK81] COOK, R. L. AND TORRANCE, K. E. A reflectance model for computer graphics. In *Computer Graphics (SIGGRAPH '81 Proceedings)*, volume 15, pp. 307–316, August 1981.

[COOK84a] COOK, R. L. Shade trees. In CHRISTIANSEN, H., editor, *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pp. 223–231, July 1984.

[COOK84b] COOK, R. L., PORTER, T., AND CARPENTER, L. Distributed ray tracing. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, pp. 137–145, July 1984.

[COUR53] COURANT, R. AND HILBERT, D. *Methods of Mathematical Physics*. Interscience Publisher, Inc., New York, 1953.

[CROS95] CROSS, R. A. Sampling patterns optimized for uniform distribution of edges. In *Graphics Gems V*, pp. 359–363. AP Professional, 1995.

[CUI97] CUI, J. AND FREEDEN, W. Equidistribution on the sphere. *SIAM Journal on Scientific Computing*, 18(2):595–609, March 1997.

[CYCH90] CYCHOSZ, J. M. Efficient generation of sampling jitter using look-up tables. In *Graphics Gems*, pp. 64–74. AP Professional, 1990.

[DEBE96] DEBEVEC, P. E., TAYLOR, C. J., AND MALIK, J. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In RUSHMEIER, H., editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pp. 11–20. ACM SIGGRAPH, Addison Wesley, August 1996.

[DEBE97] DEBEVEC, P. E. AND MALIK, J. Recovering high dynamic range radiance maps from photographs. In Computer Graphics *Proceedings, Annual Conference Series, SIGGRAPH'97*, pp. 369–378, August 1997.

[DEHA91] DEHAEMER, M. J. AND ZYDA, M. J. Simplification of objects rendered by polygonal approximations. *Computer & Graphics*, 15(2):175–184, 1991.

[DOBK93a] DOBKIN, D. P. AND EPPSTEIN, D. Computing the discrepancy. In *Proceedings of the 9th ACM Symposium on Computational Geometry*, pp. 47–52, 1993.

[DOBK93b] DOBKIN, D. P. AND MITCHELL, D. P. Random-edge discrepancy of supersampling patterns. In *Graphics Interface*, pp. 62–69, 1993.

[DOBK96] DOBKIN, D. P., EPPSTEIN, D., AND MITCHELL, D. P. Computing the discrepancy with applications to supersampling patterns. *ACM Transactions on Graphics*, 15(4):354–376, October 1996.

[FAUG93] FAUGERAS, O. AND ROBERT, L. What can two images tell us about a third one? Technical report, INRIA, July 1993.

[FOLE90] FOLEY, T. A., LANE, D. A., AND NIELSON, G. M. Towards animating raytraced volume visualization. *The Journal of Visualization and Computer Animation*, 1(1):2–8, 1990.

[FUNK93] FUNKHOUSER, T. A. AND SÉQUIN, C. H. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In KAJIYA, J. T., editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pp. 247–254, August 1993.

[GARD84] GARDNER, G. Y. Simulation of natural scenes using textured quadric surfaces. In CHRISTIANSEN, H., editor, *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pp. 11–20, July 1984.

[GARD85] GARDNER, G. Y. Visual simulation of clouds. In BARSKY, B. A., editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pp. 297–303, July 1985.

[GARL97] GARLAND, M. AND HECKBERT, P. S. Surface simplification using quadric error metrics. In WHITTED, T., editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pp. 209–216. ACM SIGGRAPH, Addison Wesley, Aug. 1997.

[GERS39] GERSHUN, A. The light field. *Journal of Mathematics and Physics*, XVIII:51–151, 1939. Translated by P. Moon and G. Timoshenko.

[GERS82] GERSHO, A. On the structure of vector quantizers. *IEEE Transactions on Information Theory*, 28:157–165, March 1982.

[GOLD89] GOLDSTEIN, E. B. *Sensation and Perception*. Wadsworth Publishing Company, 3rd edition, 1989.

[GORA84] GORAL, C. M., TORRANCE, K. E., GREENBERG, D. P., AND BATTAILE, B. Modelling the interaction of light between diffuse surfaces. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pp. 212–22, July 1984.

[GORT96] GORTLER, S. J., GRZESZCZUK, R., SZELISKI, R., AND COHEN, M. F. The lumigraph. In Computer Graphics *Proceedings, Annual Conference Series, SIGGRAPH'96*, pp. 43–54, August 1996.

[GREE86a] GREENE, N. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, 6(11), November 1986.

[GREE86b] GREENE, N. AND HECKBERT, P. S. Creating raster omnimax images from multiple perspective views using the elliptical weighted average filter. *IEEE Computer Graphics and Applications*, 6(6):21–27, June 1986.

[GREE93] GREENE, N. AND KASS, M. Hierarchical Z-buffer visibility. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pp. 231–240, 1993.

[GREE96] GREENE, N. Hierarchical polygon tiling with coverage masks. In RUSHMEIER, H., editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pp. 65–74. ACM SIGGRAPH, Addison Wesley, August 1996.

[GU97] GU, X. AND STEVEN J. GORTLER, M. F. C. Polyhedral geometry and the two-plane parameterization. In *Proceedings of the 8th Eurographics Rendering Workshop*, pp. 1–12, June 1997.

[HAEB92] HAEBERLI, P. Synthetic lighting for photography. available on `http://www.sgi.com/grafica/synth/index.html`, January 1992.

[HALL89] HALL, R. *Illumination and Color in Computer Generated Imagery*. Springer-Verlag, New York, 1989.

[HALT64] HALTON, J. H. AND SMITH, G. B. Radical-inverse quasi-random point sequence. *Communications of the ACM*, 7(12):701–702, December 1964.

[HECK86] HECKBERT, P. S. Survey of texture mapping. *IEEE Computer Graphics and Applications*, 6(11):56–67, November 1986.

[HECK87] HECKBERT, P. S. Ten unsolved problems in rendering. In *Workshop on Rendering Algorithms and Systems, Graphics Interface'87*, April 1987.

[HEIN94a] HEINRICH, S. AND KELLER, A. Quasi-monte carlo methods in computer graphics, part i: The qmc buffer. Technical report, University of Kaiserslautern, 1994. 242/94.

[HEIN94b] HEINRICH, S. AND KELLER, A. Quasi-monte carlo methods in computer graphics, part ii: The radiance equation. Technical report, University of Kaiserslautern, 1994. 243/94.

[HILB52] HILBERT, D. AND COHN-VOSSEN, S. *Geometry and the Imagination*. Chelsea Publising Company, New York, 1952.

[HOPP93] HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. Mesh optimization. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, pp. 19–26, August 1993.

[HOPP96] HOPPE, H. Progressive meshes. *Computer Graphics*, 30:99–108, August 1996.

[HOPP97] HOPPE, H. View-dependent refinement of progressive meshes. In WHITTED, T., editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pp. 189–198. ACM SIGGRAPH, Addison Wesley, August 1997.

[IHM97] IHM, I., PARK, S., AND LEE, R. K. Rendering of spherical light fields. In *Proceedings of Pacific Graphics '97*, pp. 59–68, October 1997.

[KAJI85] KAJIYA, J. T. Anisotropic reflection models. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pp. 15–21, July 1985.

[KELL95] KELLER, A. A quasi-monte carlo algorithm for the global illumination problem in the radiosity setting. In *Proceedings of Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, pp. 239–251. Springer-Verlag, June 1995.

[LEE77] LEE, D. T. AND WONG, C. K. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica*, 9(23):23–29, 1977.

[LENG97] LENGYEL, J. AND SNYDER, J. Rendering with coherent layers. In Computer Graphics *Proceedings, Annual Conference Series, SIGGRAPH'97*, pp. 233–242, August 1997.

[LEVO96] LEVOY, M. AND HANRAHAN, P. Light field rendering. In Computer Graphics *Proceedings, Annual Conference Series, SIGGRAPH'96*, pp. 31–42, August 1996.

[LIND80] LINDE, Y., BUZO, A., AND GRAY, R. M. An algorithm for vector quantizer design. *IEEE Transactions on Communcations*, 28(1):84–95, January 1980.

[LIND96] LINDSTROM, P., KOLLER, D., RIBARSKY, W., HODGES, AND F., L. Real-time, continuous level of detail rendering of height fields. *Computer Graphics*, 30:109–118, 1996.

[LIVE97] LIVE PICTURE. Realspace viewer 1.0, 1997. A software to view image-based objects.

[LIVN96] LIVNAT, Y., SHEN, H.-W., AND JOHNSON, C. R. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, March 1996.

[LORE87] LORENSEN, W. E. AND CLINE, H. E. Marching cubes: A high resolution 3D surface construction algorithm. In STONE, M. C., editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pp. 163–169, July 1987.

[LUEB97] LUEBKE, D. AND ERIKSON, C. View-dependent simplification of arbitrary polygonal environments. In WHITTED, T., editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pp. 199–208. ACM SIGGRAPH, Addison Wesley, August 1997.

[MAX95] MAX, N. AND OHSAKI, K. Rendering trees from precomputed Z-buffer views. In *Eurographics Rendering Workshop 1995*. Eurographics, June 1995.

[MC90] MURRAY-COLEMAN, J. F. AND SMITH, A. M. The automated measurement of BRDFs and their application to luminaire modeling. *Journal of the Illuminating Engineering Society*, Winter, 1990.

[MCMI95] MCMILLAN, L. AND BISHOP, G. Plenoptic modeling: An image-based rendering system. In Computer Graphics *Proceedings, Annual Conference Series, SIGGRAPH'95*, pp. 39–46, August 1995.

[MCMI97] MCMILLAN, L. *An Image-based Approach to Three-Dimensional Computer Graphics*. PhD thesis, University of North Carolina at Chapel Hill, 1997.

[MILL84] MILLER, G. S. AND HOFFMAN, C. R. Illumination and reflection maps: Simulated objects in simulated and real environments. In *SIGGRAPH '84 Advanced Computer Graphics Animation seminar notes*. July 1984.

[MONT97] MONTRYM, J. S., BAUM, D. R., DIGNAM, D. L., AND MIGDAL, C. J. Infinite Reality: a real-time graphics system. In *Computer Graphics*, volume 31, pp. 293–302, August 1997.

[NEWE77] NEWELL, M. E. AND BLINN, J. F. The progressiong of realism in computer generated images. In *ACM 77 Proceedings*, pp. 444–448, October 1977.

[NIED92] NIEDERREITER, H. Quasirandom sampling computer graphics. In *Proceedings of the 3rd International Seminar on Digital Image Processing in Medicine*, pp. 29–33, 1992.

[NIME94] NIMEROFF, J. S., SIMONCELLI, E., AND DORSEY, J. Efficient re-rendering of naturally illuminated environments. In *Fifth Eurographics Workshop on Rendering*, pp. 359–373, Darmstadt, Germany, June 1994.

[NIME96] NIMEROFF, J. S., DORSEY, J., AND RUSHMEIER, H. Implementation and analysis of an image-based global illumination framework for animated environments. *IEEE Transactions on Visualziation and Computer Graphics*, 2(4):283–298, December 1996.

[NISH85] NISHITA, T. AND NAKAMAE, E. Continuous tone representation of three-dimensional objects taking account of shadows and interreflection. In BARSKY, B. A., editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pp. 23–30, July 1985.

[OHBU96] OHBUCHI, R. AND AONO, M. Quasi-monte carlo rendering with adaptive sampling. Technical report, Tokyo Research Laboratory, IBM Japan Ltd., 1996.

[PASK95] PASKOV, S. H. AND TRAUB, J. F. Faster valuing of financial derivatives. *Journal of Portfolio Management*, 22:113–120, 1995.

[PHON75] PHONG, B.-T. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, June 1975.

[POLI93] POLIS, M. F. AND MCKEOWN, D. M. Issues in iterative TIN generation to support large scale simulations. In *Proceedings of 11th International Symposium on Computer Assisted Cartography (AUTOCARTO11)*, pp. 267–277, November 1993.

[POPO97] POPOVIĆ, J. AND HOPPE, H. Progressive simplicial complexes. In WHITTED, T., editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pp. 217–224. ACM SIGGRAPH, Addison Wesley, August 1997.

[POST97] POSTON, T., NGUYEN, H., HENG, P.-A., AND WONG, T.-T. Skeleton climbing : Fast isosurfaces with fewer triangles. In *Pacific Graphics'97*, pp. 117–126, October 1997.

[POST98] POSTON, T., WONG, T.-T., AND HENG, P.-A. Multiresolution isosurface extraction with adaptive skeleton climbing. *Computer Graphics Forum*, 1998. Accepted for publication.

[PULL97] PULLI, K., COHEN, M., DUCHAMP, T., HOPPE, H., SHAPIRO, L., AND STUETZLE, W. View-based rendering: Visualizing real objects from scanned range and color data. In *Proceedings of the 8th Eurographics Rendering Workshop*, pp. 23–34, June 1997.

[REEV87] REEVES, W. T., SALESIN, D. H., AND COOK, R. L. Rendering antialiased shadows with depth maps. In STONE, M. C., editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pp. 283–291, July 1987.

[ROSS93] ROSSIGNAC, J. AND BORREL, P. Multi-resolution 3D approximation for rendering complex scenes. In FALCIDIENO, B. AND KUNII, T. L., editors, *Modeling of Computer Graphics (Second Conference on Geometric Modelling in Computer Graphics)*, pp. 455–465. Springer-Verlag, June 1993. Genova, Italy.

[SCAR92] SCARLATOS, L. AND PAVLIDIS, T. Hierarchical triangulation using cartographic coherence. *CVGIP: Graphical Models and Image Processing*, 54(2):147–161, March 1992.

[SCHR92] SCHROEDER, W. J., ZARGE, J. A., AND LORENSEN, W. E. Decimation of triangle meshes. In CATMULL, E. E., editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pp. 65–70, July 1992.

[SEGA92] SEGAL, M., KOROBKIN, C., VAN WIDENFELT, R., FORAN, J., AND HAEBERLI, P. E. Fast shadows and lighting effects using texture mapping. In CATMULL, E. E., editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pp. 249–252, July 1992.

[SEIT96] SEITZ, S. M. AND DYER, C. R. View morphing. In *Computer Graphics Proceedings, Annual Conference Series (Proc. SIGGRAPH '96)*, pp. 21–30, 1996. Available from ftp.cs.wisc.edu.

[SEIT97] SEITZ, S. M. AND DYER, C. R. Photorealistic scene reconstruction by voxel coloring. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 1067–1073, 1997.

[SEIT98] SEITZ, S. M. AND KUTULAKOS, K. N. Plenoptic image editing. In *Proceedings 6th International Conference on Computer Vision (ICCV'98)*, 1998.

[SHEK96] SHEKHAR, R., FAYYAD, E., YAGEL, R., AND CORNHILL, J. Octree-based decimation of marching cubes surfaces. In *IEEE Visualization '96 Proceedings*, pp. 335–342, Oct 1996.

[SHIR91a] SHIRLEY, P. Discrepancy as a quality measure for sample distributions. In *Proceedings of Eurographics*, pp. 183–193, 1991.

[SHIR91b] SHIRLEY, P. *Physically Based Lighting Calculations for Computer Graphics*. PhD thesis, University of Illinois at Urbana-Champaign, 1991.

[SHU95] SHU, R., CHEN, Z., AND KANKANHALLI, M. S. Adaptive marching cubes. *The Visual Computer*, 11:202–217, 1995.

[SIEG81] SIEGEL, R. AND HOWELL, J. R. *Thermal Radiation Heat Transfer*. Hemisphere Publishing Corp., Washington, DC, 1981.

[SILL91] SILLION, F. X., ARVO, J. R., WESTIN, S. H., AND GREENBERG, D. P. A global illumination solution for general reflectance distributions. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pp. 187–196, July 1991.

[SILL94] SILLION, F. AND PUECH, C. *Radiosity and Global Illumination*. Morgan Kaufmann, San Francisco, 1994.

[SPAN69] SPANIER, J. AND GELBARD, E. M. *Monte Carlo Principles and Neutron Transport Problems*. Addison-Wesley, New York, N.Y., 1969.

[SUTH63] SUTHERLAND, I. E. Sketchpad: a man-machine graphical communication system. *SJCC*, 1963.

[SZEL97] SZELISKI, R. AND SHUM, H.-Y. Creating full view panoramic image mosaics and environment maps. In Computer Graphics *Proceedings, Annual Conference Series, SIGGRAPH'97*, pp. 251–258, August 1997.

[TEZU95] TEZUKA, S. *Uniform Random Numbers: Theory and Practice.* Kluwer Academic Publishers, 1995.

[TORR67] TORRANCE, K. E. AND SPARROW, E. M. Theory for off-specular reflection from roughened surfaces. In *Journal of Optical Society of America*, volume 57, pp. 1105–1114, September 1967.

[TRAU96] TRAUB, J. In math we trust. *What's Happening in the Mathematical Sciences*, 3:101–111, 1996.

[TURK92] TURK, G. Re-tiling polygonal surfaces. In CATMULL, E. E., editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pp. 55–64, July 1992.

[WARD92] WARD, G. J. Measuring and modeling anisotropic reflection. In Computer Graphics *Proceedings, Annual Conference Series, SIGGRAPH'92*, pp. 265–272, July 1992.

[WARD94] WARD, G. J. The RADIANCE lighting simulation and rendering system. *Computer Graphics*, 28:459–472, 1994.

[WHIT80] WHITTED, T. An improved illumination model for shaded display. In *Communications of the ACM*, volume 23, pp. 343–349, June 1980.

[WILH92] WILHELMS, J. AND GELDER, A. V. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.

[WILL83] WILLIAMS, L. Pyramidal parametrics. In *Computer Graphics (SIGGRAPH '83 Proceedings)*, volume 17, pp. 1–11, July 1983.

[WONG97a] WONG, T.-T., HENG, P.-A., OR, S.-H., AND NG, W.-Y. Illuminating image-based objects. In *Pacific Graphics'97*, pp. 69–78, Seoul, Korea, October 1997.

[WONG97b] WONG, T.-T., HENG, P.-A., OR, S.-H., AND NG, W.-Y. Image-based rendering with controllable illumination. In *Eighth Eurographics Workshop on Rendering (Rendering Techniques'97)*, pp. 13–22, Saint Etienne, France, June 1997.

[WONG97c] WONG, T.-T., LUK, W.-S., AND HENG, P.-A. Sampling with hammersley and halton points. *ACM Journal of Graphics Tools*, 2(2):9–24, 1997.

[WONG98] WONG, T.-T., HENG, P.-A., OR, S.-H., AND NG, W.-Y. Illumination of image-based objects. *Journal of Visualization and Computer Animation*, 1998. Accepted for publication.

[WYVI90] WYVILL, B. AND JEVANS, D. Table driven polygonisation. SIGGRAPH *1990 course notes* **23**, pp. 7–1–7–6, 1990.

[XIA96] XIA, J. C. AND VARSHNEY, A. Dynamic view-dependent simplification for polygonal models. In *IEEE Visualization '96*. IEEE, October 1996. ISBN 0-89791-864-9.

[ZHAN98] ZHANG, Z. Modeling geometric structure and illumination variation of a scene from real images. In *Proceedings of the International Conference on Computer Vision (ICCV'98)*, Bombay, India, January 1998.

Appendix A

Spherical Harmonics

Spherical harmonics have been used in solving various physical problems. The spherical harmonics $Y_{l,m}(\theta, \phi)$ are the solution to Laplace's equation in spherical coordinates. They are functions of two angular parameters, the zenith angle θ and the azimuth angle ϕ, specifying a position on the surface of the sphere. Each harmonic is represented by two indices, l and m. Index l is known as the spherical degree while m as the azimuthal order. The spherical harmonics are formulated to associate with the Legendre polynomials [1].

$$Y_{l,m} = \sqrt{\frac{2l+1}{2\pi}\frac{(l-m)!}{(l+m)!}} P_{l,m}(\cos\theta) e^{im\phi}, \tag{A.1}$$

where $P_{l,m}(x)$ are the associated Legendre polynomials,

and $m = -l, (-l+1), \cdots, 0, \cdots, (l-1), l$.

Here the first term of the equation on the right is the normalization coefficient which normalizes the spherical harmonics. The integration of a normalized spherical harmonic over a sphere is unity.

$$\int_0^{2\pi} \int_{-1}^{1} Y^*_{l',m'}(\theta, \phi) Y_{l,m}(\theta, \phi) d\cos\theta d\phi = \delta_{mm'}\delta_{ll'}, \tag{A.2}$$

where δ_{mn} is called the Kronecker Delta, The asterisk denotes the complex conjugation.

Spherical harmonics obey the following three properties,

$$Y_{l,-l}(\theta, \phi) = \frac{1}{2^l l!}\sqrt{\frac{(2l+1)!}{4\pi}} \sin^l\theta e^{-il\phi}, \tag{A.3}$$

$$Y_{l,0}(\theta, \phi) = \sqrt{\frac{2l+1}{4\pi}} P_{l,0}(\cos\theta), \tag{A.4}$$

$$Y_{l,-m}(\theta, \phi) = (-1)^m Y^*_{l,m}(\theta, \phi), \tag{A.5}$$

[1] After the French mathematician Adrien-Marie Legendre (1752-1833)

The third property in Equation A.5 can be used so that the spherical harmonics always related to an associated Legendre polynomial with $m \geq 0$.

There are three modes for the spherical harmonics. The mode is depending on the order of m. When $m = 0$, it is called the zonal mode. The harmonics with $m = 0$ are called zonal harmonics and are in the following form,

$$k_{\text{zonal}} P_{l,0}(\cos\theta), \tag{A.6}$$

where k_{zonal} is a coefficient.

When $m = l$, the harmonics are called sectoral harmonics and are in the following form,

$$k_{\text{sectoral}} \sin(m\phi) P_{l,l}(\cos\theta) \;\text{ or }\; k'_{\text{sectoral}} \cos(m\phi) P_{l,l}(\cos\theta), \tag{A.7}$$

In other cases, $m \neq 0$ and $m \neq l$, the harmonics are called tesseral harmonics and in the form of

$$k_{\text{sectoral}} \sin(m\phi) P_{l,m}(\cos\theta) \;\text{ or }\; k'_{\text{sectoral}} \cos(m\phi) P_{l,m}(\cos\theta), \tag{A.8}$$

where $l \neq m$.

The spherical harmonics form a complete orthonormal basis. Hence any real spherical function $f(\theta, \phi)$ can be expanded in term of complex spherical harmonics.

$$f(\theta, \phi) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} C^c_{l,m} Y_{l,m}(\theta, \phi), \tag{A.9}$$

where $C^c_{l,m}$ are the coefficients.

Moreover, the real spherical function $f(\theta, \phi)$ can also be expanded in term of real spherical harmonics using the same summation equation as in Equation A.9. The real spherical harmonics can be expressed as follow,

$$Y^r_{l,m}(\theta, \phi) = \begin{cases} N_{l,m} P_{l,m}(\cos\theta) \cos(m\phi) & \text{if } m > 0 \\ N_{l,0} P_{l,0}(\cos\theta)/\sqrt{2} & \text{if } m = 0 \\ N_{l,m} P_{l,|m|}(\cos\theta) \sin(|m|\phi) & \text{if } m < 0, \end{cases} \tag{A.10}$$

where,

$$N_{l,m} = \sqrt{\frac{2l+1}{2\pi} \frac{(l-|m|)!}{(l+|m|)!}},$$

Note this is the formulation used in Chapter 9.

The major computation of evaluating spherical harmonics is the evaluation of the Legendre polynomials. For simplicity, let's substitute $x = \cos\theta$, the ordinary Legendre polynomials are defined by,

$$P_{l,m}(x) = (-1)^m (1-x^2)^{m/2} \frac{d^m}{dx^m} P_l(x). \tag{A.11}$$

The Legendre polynomials can be evaluated numerically using the explicit expression. However, this a bad approach in writing computer program. A better approach is to express the Legendre in recurrence form.

$$P_{l,m}(x) = \begin{cases} (1-2m)\sqrt{1-x^2}P_{m-1,m-1}(x) & \text{if } l = m \\ x(2m+1)P_{m,m}(x) & \text{if } l = m+1 \\ x\frac{2l-1}{l-m}P_{l-1,m}(x) - \frac{l+m-1}{l-m}P_{l-2,m}(x) & \text{otherwise.} \end{cases} \tag{A.12}$$

The above expression of the Legendre polynomial can be efficiently coded in computer languages.