國立交通大學

資訊科學系

碩 士 論 文

一個事先計算輻射傳輸為基礎的即時全域照度演算法

A Real-Time Global Illumination Algorithm based on Pre-Computed Radiance Transfer

研 究 生：陳志豪

指導教授：施仁忠　教授

中 華 民 國 九 十 四 年 六 月

一 個 事 先 計 算 輻 射 傳 輸 為 基 礎 的 即 時 全 域 照 度 演 算 法
A Real-Time Global Illumination Algorithm based on Pre-Computed Radiance Transfer

研　究　生：陳志豪	Student：Chin-Hao Chen
指導教授：施仁忠	Advisor：Zen-Chung Shih

國 立 交 通 大 學
資 訊 科 學 系
碩 士 論 文

A Thesis
Submitted to Institute of Computer and Information Science
College of Electrical Engineering and Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer and Information Science

June 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年六月

國　立　交　通　大　學

博碩士論文全文電子檔著作權授權書

（提供授權人裝訂於紙本論文書名頁之次頁用）

本授權書所授權之學位論文，為本人於國立交通大學資訊科學系所
____________組， 93 學年度第 2 學期取得碩士學位之論文。

論文題目：一個事先計算輻射傳輸為基礎的即時全域照度演算法
指導教授：施仁忠

■ 同意

本人茲將本著作，以非專屬、無償授權國立交通大學與台灣聯合大學系統圖書館：基於推動讀者間「資源共享、互惠合作」之理念，與回饋社會與學術研究之目的，國立交通大學及台灣聯合大學系統圖書館得不限地域、時間與次數，以紙本、光碟或數位化等各種方法收錄、重製與利用；於著作權法合理使用範圍內，讀者得進行線上檢索、閱覽、下載或列印。

論文全文上載網路公開之範圍及時間：

本校及台灣聯合大學系統區域網路	■ 立即公開
校外網際網路	■ 立即公開

■ **全文電子檔送交國家圖書館**

授 權 人：陳志豪

親筆簽名： 陳志豪

中華民國 94 **年** 6 **月** 27 **日**

國 立 交 通 大 學

博碩士紙本論文著作權授權書

（提供授權人裝訂於全文電子檔授權書之次頁用）

本授權書所授權之學位論文，為本人於國立交通大學資訊科學系所 ＿＿＿＿＿＿＿組， 93 學年度第 2 學期取得碩士學位之論文。

論文題目：一個事先計算輻射傳輸為基礎的即時全域照度演算法
指導教授：施仁忠

■ 同意

本人茲將本著作，以非專屬、無償授權國立交通大學，基於推動讀者間「資源共享、互惠合作」之理念，與回饋社會與學術研究之目的，國立交通大學圖書館得以紙本收錄、重製與利用；於著作權法合理使用範圍內，讀者得進行閱覽或列印。

本論文為本人向經濟部智慧局申請專利(未申請者本條款請不予理會)的附件之一，申請文號為：＿＿＿＿＿＿＿＿＿＿，請將論文延至＿＿年＿＿月＿＿日再公開。

授 權 人：陳志豪

親筆簽名：陳志豪

中華民國 94 **年** 6 **月** 27 **日**

國家圖書館
博碩士論文電子檔案上網授權書

（提供授權人裝訂於紙本論文本校授權書之後）

ID:GT009223529

本授權書所授權之論文為授權人在國立交通大學資訊科學系所 93 學年度第 2 學期取得碩士學位之論文。

論文題目：一個事先計算輻射傳輸為基礎的即時全域照度演算法
指導教授：施仁忠

茲同意將授權人擁有著作權之上列論文全文（含摘要），非專屬、無償授權國家圖書館，不限地域、時間與次數，以微縮、光碟或其他各種數位化方式將上列論文重製，並得將數位化之上列論文及論文電子檔以上載網路方式，提供讀者基於個人非營利性質之線上檢索、閱覽、下載或列印。

※ 讀者基於非營利性質之線上檢索、閱覽、下載或列印上列論文，應依著作權法相關規定辦理。

授權人：陳志豪

親筆簽名：陳志豪

民國94**年**6 **月**27**日**

一個事先計算輻射傳輸為基礎的即時全域照度演算法

研究生：陳志豪　　　　　　　　　　指導教授：施仁忠 教授

國立交通大學資訊科學系

摘　　要

在全域照明當中，我們可以使用事先計算輻射傳輸函數，來儲存光線照明的果。由於事先計算輻射傳輸函數的資料量龐大，光線照明的效果並不能即時的描繪出來。在過去的研究中，利用分類的概念來事先計算輻射傳輸函數資料的壓縮，來加速描繪光線照明效果的效率。但是資料分類會導致了多邊形重繪的問題。多邊形重繪會直接的降低描繪的效率。在本論文當中，我們利用了位置一致性來減少多邊形重繪。另外我們也提出了一個選擇分類為超分類的演算法來減少多邊形重繪。

A real-time global illumination algorithm based on pre-computed radiance transfer

Student: Chin-Hao Chen Advisor: Dr. Zen-Chung Shih

Department of Computer and Information Science

National Chiao-Tung University

ABSTRACT

In global illumination, we can use Pre-computed Radiance Transfer (PRT) Function to store the lighting effects. Since the required storage for the values of PRT Function is huge, the result of lighting can not be rendered in real-time. In Peter-Pike Sloan's research, they compress the required storage by clustering PRT data to accelerate the rendering performance. But clustering data may result the polygon overdraw problem. Polygon overdraw will decrease the rendering performance directly. In this thesis, we apply the Geometry Coherence to reduce overdraw. Besides, we also introduce a cluster selection algorithm for super-cluster to decrease triangle overdraws.

Acknowledgements

First of all, I would like to express much respect to my advisor, Prof. Zen-Chung Shih. I greatly appreciate his patient and guidance. Moreover, I want to dedicate the achievement of this work to my family. They encouraged me and gave me support during these days. Besides, I am grateful to all members in Computer Graphics and Virtual Reality Laboratory for their help in these days. I deeply treasure the friendship among us. Last, many thanks are gave to all other people who have ever helped me on my research.

Contents

List of Tables

List of Figures

Chapter 1 Introduction

In global illumination, it is a challenge to render the global light transport in real-time, especially when the light source is an area light source which needs to integrate over many samples to get the lighting effects. Traditional global illumination methods, such as ray tracing [26], radiosity [4], and photon mapping [9], are too expensive for real-time applications.

1.1 Motivation

Sloan et al. [22] introduced the pre-computed radiance transfer (PRT) algorithm to store the lighting effects, such as shadowing and inter-reflection. For example, the point p, as shown in Figure 1.1. Source radiance is from the infinite spherical environment map. Transferred incident radiance equals to the source radiance but increased by inter-reflection and decreased by shadow. Exiting radiance is resulted from that the bi-directional reflectance distribution function (BRDF) times the transferred incident radiance. For each vertex in the entire scene, the PRT function is measured and stored in the pre-process.

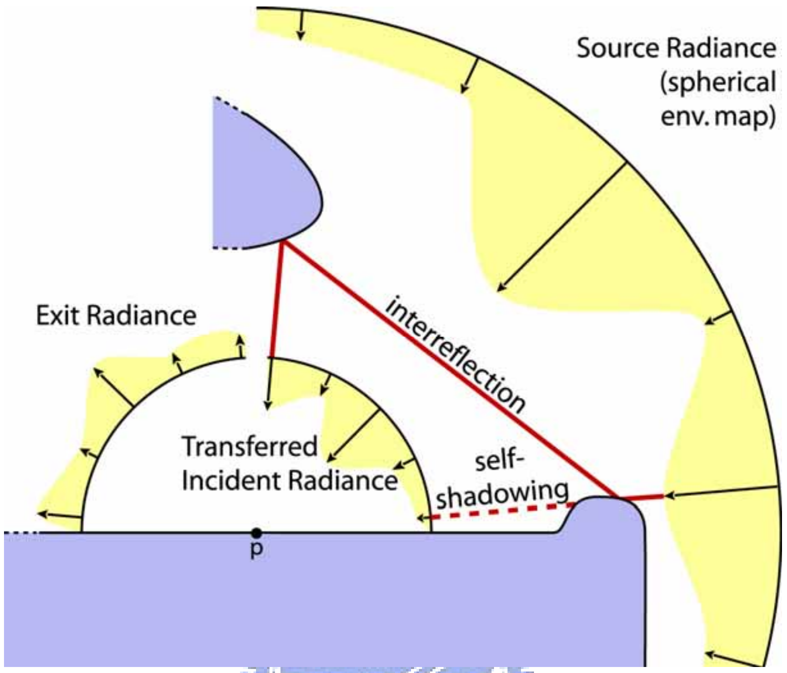


Figure 1.1: Radiance Transfer at point p from source radiance
to transferred incident radiance to exit radiance.

Sloan et al. [22] also project the radiance transfer function of each vertex to spherical harmonic (SH) basis. The projected coefficients of each vertex form a vector, called the PRT vector. The source radiance (spherical environment map) is also projected to the SH basis. Similarly, the projected coefficients of source radiance form a vector, called the lighting vector. When rendering, we only need to calculate the dot product of PRT vector and lighting vector for each vertex. For example, a diffuse object is considered and used the order 6 SH basis. It means that only the dot product of two 36D vectors is calculated for each vertex in the whole scene. Although the

PRT algorithm increases the rendering speed, it does not render the scene interactively. It is because that the computation of rendering is still large.

Sloan et al. [21] introduced the clustered principle component analysis (CPCA) algorithm to improve the PRT algorithm. The main idea is to cluster the data. The PRT vectors of the whole scene are clustered into several clusters and then apply the principle component analysis (PCA) to compress every cluster. The principle component analysis is a traditional method in statistics. When rendering object, they only need to calculate the weighted sum of the product of principle transfer vectors and lighting vector. The computation of rendering object is decreased. The rendering speed is increased.

1.2 Overviews

Although Sloan et al. [21] improved the rendering speed for PRT based illumination, but they also arose the overdraw problem. The overdraw problem means that some triangles will draw over one time when rendering object. This problem will decrease the rendering speed directly.

In this thesis, our objective is to increase the rendering speed by decreasing polygon overdraw. We describe two techniques to decrease the polygon overdraw.

First, we introduce the geometry coherence. It is used when we cluster the PRT vectors. Second, we propose a clusters selection algorithm for super-cluster.

1.3 Thesis Organization

This thesis is organized as follows. In chapter 2, we review the previous related researches in global illumination, pre-computed radiance transfer, and principle component analysis. In chapter 3, we introduce our propose system for compression the PRT vectors and rendering based on CPCA method. In chapter 4, we introduce the geometry coherence and the cluster selection algorithm for super-cluster to reduce the triangle overdraws. In chapter 5, we demonstrate the implementation results and compare to the original CPCA method. Finally, conclusions and future work are given in chapter 6.

Chapter 2
Related Works

In this chapter, we are going to discuss some related researches. First in section 2.1, we survey several illumination methods proposed in the past ten years. In section 2.2, we review researches which extend the concept of traditional PRT method. In section 2.3, we discuss researches about approximating the radiance transfer metric. Finally in section 2.4, we talk about researches in computer graphics using principle component analysis.

2.1 Illumination methods

There are various techniques proposed to encapsulate and capture the global illumination effects. Levoy and Hanrahan [12] proposed the *light field rendering* method. The light field records radiance samples in a set of images. The set of images are used for rendering views from arbitrary camera positions.

Gortler et al. [5] proposed a similar image-based rendering technique to record and capture the appearance of objects, which is called the *lumigraph* technique. Besides, there are some extended researches about the previous two techniques, such as Chen et al. [1], Miller et al. [17], and Wood et al. [27]. However, these techniques only support the static lighting environments when render objects. It means that although they can support arbitrary views but they should fix the lighting statement of objects when rendering.

Sloan et al. [22] proposed the pre-computed radiance transfer (PRT) method which captures and records the lighting effects, such as self-shadow and self-inter-reflection, in dynamic lighting environment. By using spherical harmonic (SH) basis to represent the radiance transfer function and lighting function, it can render a scene by dot products. For non-diffuse objects, they need to calculate dot product of radiance transfer matrices and lighting vector. For diffuse objects, they only need to calculate the dot product of radiance transfer vectors and lighting vector.

2.2 Survey of PRT Extensions

Up to now, there are various PRT extensions proposed. Lehtinen and Kautz [11] compress the PRT raw data by principle component analysis (PCA) to increase the rendering performance. Sloan et al. [21] proposed the clustered principle component analysis (CPCA) method. First, they cluster the PRT raw data into several clusters before compressing. It can decrease the variance of the radiance transfer matrices. Next, they used the PCA to compress the data of each cluster. Thus, the required storage is decreased and the rendering performance is increased.

Besides, Sloan et al. [23] also proposed another technique, *bi-scale radiance transfer*, to improve the rendering quality of the PRT data. They modeled the radiance transfer functions at two scales. The first, macro-scale, captures the global light transport and measures the ordinary PRT technique. The second, meso-scale, modeled a better structure of the surface using 4D radiance transfer texture (RTT). They also project RTTs to SH basis, but differ from bidirectional texture functions (BTFs) proposed by Dana et al. [3].

2.3 PRT approximation methods

In traditional PRT methods, they applied the SH basis to approximate the radiance transfer function. But there are some related works using wavelet basis to approximate radiance transfer function instead of using SH basis. Liu et al. [14] proposed a PRT-based method. They factored the glossy BRDF into purely view-dependent part and purely light-dependent part. The radiance transfer matrices are transformed onto the wavelet basis and compressed by CPCA method similarly.

Ng et al. [18] also projected the radiance transfer function to wavelet basis. But they select a subset of the wavelet basis for rendering the current frame. Furthermore, Ng et al. [19] extended their concept into the triple product wavelet integrals for relighting scenes. The triple product includes the incident lighting, visibility, and the BRDF functions.

Unlike the SH basis projected methods, the wavelet basis projected methods can handle the high-frequency lighting. But they still can not archive the real-time rendering performance on current graphic hardware. Their rendering performance is below 7 FPS for median-size models.

2.4 PRT compress method

The principle component analysis [10] (PCA) method is a traditional method in statistics. PCA is also used in computer graphics in some researches such like James et al. [8], Matusik et al. [15], Vasilescu et al. [24], and Wood et al. [27].

PCA can find the major components of the high dimensional sample data and transform the sample data into low dimensional data set. In this thesis, we also apply the PCA to approximate the radiance transfer matrices.

Chapter 3
System overview

Our system consists of two major processes, one is the pre-computed process and the other is rendering process. The input of pre-computed process is a 3D model. After the pre-computed process, the outputs are the principle component of radiance transfer matrices. The principle component of radiance transfer matrices is also one of the inputs of rendering process. After the rendering process, the lighting transport results of the 3D model are displayed on the screen. In this chapter, we will discuss these two processes individually.

3.1 Pre-computed process

The pre-computed process consists of three modules; the PRT computation module, LBG clustering module, and PCA compressing module. Figure 3.1 shows the flow chart of the pre-computed process.

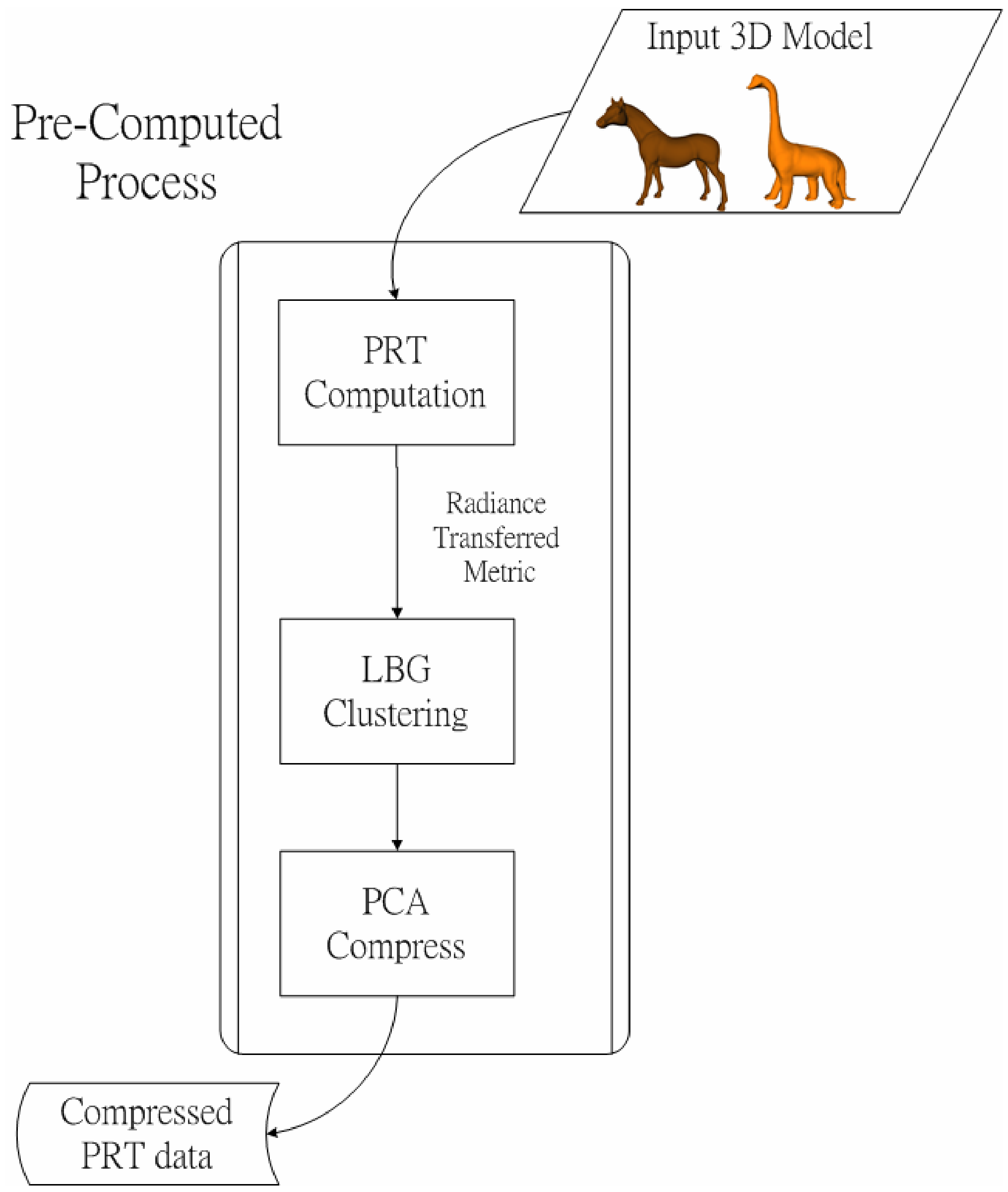


Figure 3.1: Pre-computed process flow chart

A 3D model is the input of the pre-computed process. The outputs of the PRT computation module are the PRT vector of each vertex. The PRT vector of each vertex is given by projecting the radiance transfer matrix to the SH basis domain. The PRT vectors of all vertices form the inputs of the LBG cluster module. The LBG cluster module classifies the inputs into several clusters. After LBG cluster module, for every cluster, we apply the PCA compressing module to compress the data. The output compressed data is saved for rendering process. We will describe the three modules in the following.

3.1.1 PRT computation module

The PRT computation of radiance transfer matrices is proposed by Sloan et al. [22]. They separated the measurement process into two passes: the shadow pass and the light transport pass. The shadow pass measures the direct shadow effect. The light transport pass measures the lighting effects, such as reflections and transmissions.

First, in the shadow pass, we get the visibility map $\boldsymbol{V_p}$ for each vertex. The visibility map is measured by setting view position at vertex $\boldsymbol{p}$ and render mesh into the cubic map. Next, sample the visibility map in direction $\boldsymbol{s_d}$. $\boldsymbol{s_d}$ only needs the directions around the vertex normal above the hemi-sphere. The shadow transfer matrix is measured by integrating all directions above the hemi-sphere. The equation is shown below Equation 1.

$$(M_p)_{ij}^0 = \int_{s\in S,\ s\bullet n_p \geq 0} V_p(s) y_l^m(s) y_{l'}^{m'}(s) ds \qquad (1)$$

$\boldsymbol{(M_p)_{ij}^0}$ is an element of the shadow transfer matrix in row $\boldsymbol{i}$ of column $\boldsymbol{j}$. The matrix element index is given by ***m***, ***l***, ***m'***, and ***l'***, where "***i***" equals to ***l(l+1)+m+1*** and "*j*" equals to **l'(l'+1)+m'+1**. $y_l^m(s)$ means that the ***m***-th projected SH coefficient of order ***l*** in direction ***s***. $\boldsymbol{n_p}$ is the normal of the vertex ***p***.

Before measuring light transport pass, we need to project BRDF to SH basis domain. Westin et al. [25] proposed a Monte Carlo technique to compute the BRDF matrix, B. The z-axis is the vertex normal and the y-axis is the tangent vector of the vertex. Then we project the BRDF matrix to SH basis.

In the visibility map, there are three situations: entirely shadowed, entirely un-shadowed, and partially shadowed. When measuring light transport pass, we only need to measure the directions which are partially shadowed. We update the transfer matrix from $(M_p)_{ij}^0$ iteratively, as shown below Equation 2 and 3.

$$(M_p)_{ij}^b = (M_p)_{ij}^{b-1} + \int_{s \in S} (1 - Vp(s)) y_l^m(s) IR_p(s) ds \tag{2}$$

$$IR_p(s) = \sum_{k=1}^{l''^2} \left\{ \left(R_q(B)(M_q)^{b-1} \right)_{kj} y_{l''}^{m''}(-s) \right\} \tag{3}$$

$(M_p)_{ij}^b$ is the final radiance transfer matrix of vertex ***p***. "***q***" is the hitting point of sample ray from vertex ***p*** to another triangle in direction ***s***. $R_q(B)$ denotes the rotated BRDF matrix where the coordinate is aligned the local coordinate of the vertex. If the object does not consider the light transmission effect, we only need to integrate over the hemi-sphere. "***b***" is a user defined value which is the maximum number of iterations for measuring radiance transfer matrix.

3.1.2 LBG clustering module

The PRT computation module measures the radiance transfer matrix of each vertex and projects to the SH basis. The inputs of the LBG clustering module are the projected coefficients. The projected coefficients of each vertex are called the PRT vector. For all vectors, we cluster the data using the LBG algorithm [13]. The LBG clustering algorithm is also called *k-means* clustering algorithm [20], where "***k***" means the data will be clustered into k clusters. The pseudo code of LBG algorithm is shown in Figure 3.2.

```
Step 0: Initial pass:
        For each cluster, randomly give the mean value.
Step 1: Cluster pass:
        For each vector, find the nearest cluster by calculating distance of mean.
Step 2: Update pass:
        For each cluster, update the mean by average the vectors which belong to the
        cluster.
Step 3: Check pass:
        Summation the total energy of all vertices $E_i$
        Compare to the total energy of the last iteration $E_{i-1}$
        If the converged ratio small than a pre-given threshold, then cluster done;
        Else jump to the step 1 and do next iteration.
```

Figure 3.2: The pseudo code of LBG algorithm

The total energy of all vertices $\boldsymbol{E_i}$ is defined as following Equation 4. The "$\boldsymbol{i}$" means $\boldsymbol{i}$-th iteration. The distance of mean is the Gaussian distance.

$$d_v = \|\vec{V} - Mean\|^2$$
$$Ei = \sum_{v \in all\ vertex} d_v \tag{4}$$

The converged ratio is defined in Equation 5. When the ratio is less than a user defined threshold, the LBG clustering algorithm is done. In our system, another converged state is occurred when the total energy function equals to zero. It may occur when all data is the same.

$$R = (E_{i-1} - E_i)/E_{i-1}$$
$$E_0 = \infty \tag{5}$$

The traditional LBG algorithm initializes means by randomly sampled values. It may take unnecessary iterations to converge the state. So, in our system, we first scan all vertices to find the hyper-box of all data. The idea of hype-box is similar to the bounding box, where the difference is that the dimension of hyper-box is greater than 3. Then we randomly pick up several points in hyper-box as initial cluster means.

Furthermore, the LBG algorithm may result another problem, called "Null Cluster", even though the state converged. The "Null Cluster" means that no data belongs to this cluster after clustering algorithm done. It is because that we pick up points randomly. To solve this in our system, we will find a cluster, which includes more data than any other clusters, when the null cluster occurred. Then we divide the data into two clusters. The total energy is surely less than the original energy. The "Null Cluster Avoidance" is still applied until there are none "Null Cluster". After LBG clustering module, the original data are partitioned into several clusters.

3.1.3 PCA compressing module

After LBG clustering module, we apply PCA to each cluster. The PCA compressing module will analyze the data and find the principle components. The number of principle components is given by user. In practice, we often use 8 or 12 principle components for each cluster. After PCA for each cluster, we will get the principle components for a cluster and get the weighting coefficients vector for each vertex in the cluster. In rendering time, we will reconstruct the PRT vectors by weighting coefficients vector and principle components. The equation is shown as following Equation 6.

$$\boldsymbol{R}_p \approx \boldsymbol{M}_k + \sum_j \boldsymbol{w}_{pj} \cdot \boldsymbol{B}_j \tag{6}$$

The "$\boldsymbol{R}_p$" is the approximated PRT vector of vertex $\boldsymbol{p}$. The distortion of the "$\boldsymbol{R}_p$" and the PRT vector is effected by the number of the principle components. The more number of the principle components the less approximating distortion. The "$\boldsymbol{M_k}$" is the mean of the cluster. The "$\boldsymbol{w_{pj}}$" is the weighting coefficients vector of vertex $\boldsymbol{p}$. The "$\boldsymbol{B_j}$" is the principle components of the cluster. For the vertices in each cluster, "$\boldsymbol{M_k}$" and "$\boldsymbol{B_j}$" is the same. For each vertex, it only need restore "$\boldsymbol{w_{pj}}$" and which cluster it belongs to.

After this module, the required storage is decreased. The required storage of the PRT vectors is (NumberOfVertice * VectorSize). After the PCA compressing module, the required storage is (NumberOfVertice * NumberOfPC + NumberOfCluster * (NumberOfPC +1)* VectorSize).

3.2 Rendering process

In this section, we describe the rendering process. The flow chart of rendering process is shown as Figure 3.3.

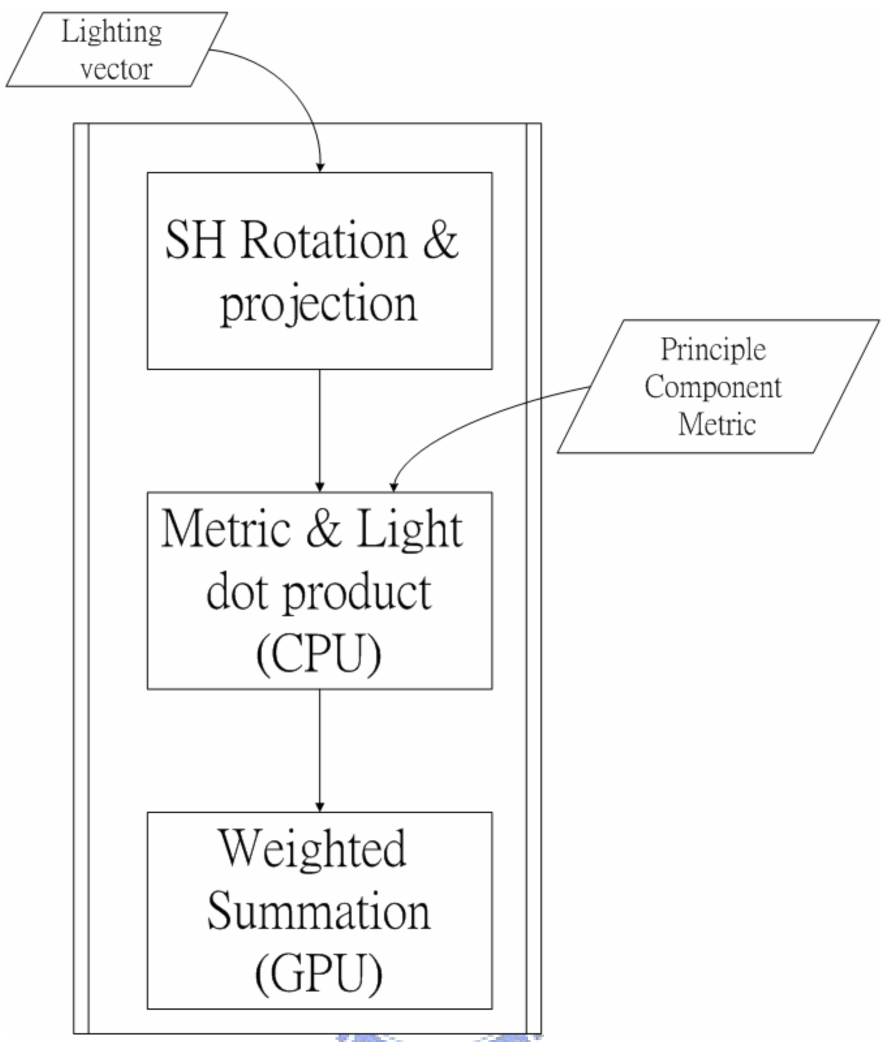


Figure 3.3: The rendering process flow chart

There are also three components in this process. The inputs of the process are lighting vector and principle component matrices. The lighting vector is obtain by projecting the light source, environment lighting or several light sources, to the SH basis domain. The principle component matrices is obtained by the pre-computed process. In this process, we will calculate the lighting effect for each vertex.

Before calculating the dot product of vectors and lighting vector, we need to rotate the light vector. It is because that the lighting environment is dynamic. The environment can be rotated arbitrarily. Since the light source came from infinite, the movement of environment need not concern. Although we can re-project light source to SH basis again when the lighting environment changed, but it takes expensive computation for rendering. Ivanic et al. [6][7] proposed a method to rotate the SH coefficients in the SH domain. The computation cost is less than re-projecting lighting vector to SH basis. We also apply their method in our system.

After pre-computed process, we get the principle components of clusters and the weighting vectors of all vertices. In the original PRT, we need to calculate the dot product of the PRT vectors and lighting vector to get the lighting effect. After compression, we only need to calculate the dot product of the principle components and the lighting vector, and weighted summation by weighting vector. The equation is shown as follows: Equation 7.

$$\begin{aligned} &\boldsymbol{R}_p \approx \boldsymbol{M}_k + \sum_j w_{pj} \cdot \boldsymbol{B}_j \\ &\Rightarrow \boldsymbol{R}_p \bullet \boldsymbol{L} \approx \boldsymbol{M}_k \bullet \boldsymbol{L} + \sum_j w_{pj} \cdot \boldsymbol{B}_j \bullet \boldsymbol{L} \end{aligned} \tag{7}$$

In our system, we calculate the dot product $\boldsymbol{M}_k \cdot \boldsymbol{L}$ and $\boldsymbol{B}_j \cdot \boldsymbol{L}$ in CPU and the weighting summation in GPU using fragment shader. We use the new features of the fragment shader model 3.0 and do per-vertex computation. The principle component metric are placed in the hardware float constant registers. The weighting vector of each vertex is placed as the input of the shader. We use the High Level Shading Language (HLSL) to utilize the shader to compute the weighted summation. The other points on the mesh surface except mesh vertices will shade by hardware interpolation.

Chapter 4
Triangles Overdraw Reduction

In this chapter, we discuss the triangles overdraw problem and introduce two methods to reduce it. First in Section 4.1, we introduce the triangles overdraw problem. In Section 4.2, we apply the Geometry Coherence to decrease the triangles overdraw. In Section 4.3, we introduce a cluster selection algorithm for super-cluster to reduce triangles overdraw.

4.1 The triangles overdraw problem

The triangles overdraw problem means that triangles are drawn over more than one time when rendering. This problem is occurred as a result of the current graphic hardware restriction. Recalling in Chapter 3, we cluster the PRT vectors into several clusters and place the principle components into the graphic card registers. If the number of clusters is large enough, the registers requirement of all principle components may be larger than the number of graphic hardware registers. Therefore the objects are not rendered in one pass. Some triangles could be rendered in the second or third passes. For example, the modern graphic hardware equips with 256 float4 registers (i.e. float4 register equals 4 float registers). If we use order 6 SH basis for glossy objects, 8 principle components per cluster. The PRT vectors are classified into 60 clusters. The requirement of registers is ((6*6)*(8+1)*60)/4, i.e. 4860, float4 registers is larger than 256 float4 registers.

In our system, we will split the mesh into several partitions by distribution of clusters. When rendering, we accumulate all rendering result of partitions to get the final result. For example in Figure 4.1, the mesh is divided into three clusters and represented by three colors, red, blue, and green respectively.

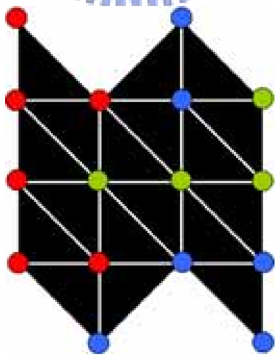


Figure 4.1: The overdraw mesh

The mesh will be split into three partitions, as shown in Figure 4.2.

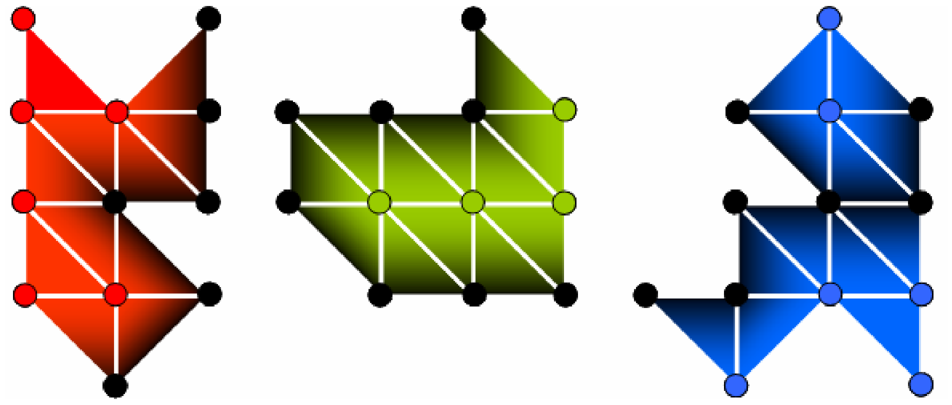


Figure 4.2: Mesh Partitions

When rendering vertices of triangles in a cluster, there are two situations occurred. One is the vertex belong to current cluster. We will calculate the lighting effect of the vertex. The other is the vertex belong to another cluster. We will set the output lighting effect result to zero (i.e. black color). After all clusters are rendered, all vertices lighting effect are calculated. The triangles overdraw occurs. Figure 4.3 shows all triangles overdraw of the example mesh.

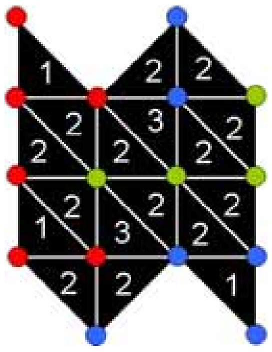


Figure 4.3: Triangles overdraw.

There are three situations of triangle overdraw. First, the triangle overdraws equals to one, it means that the vertices of triangle belong to the same cluster. Second, the triangle overdraws equal to two, it means that the vertices of the triangle belong to two clusters. The triangle will be drawn in two times. Third, the triangle overdraws equals to three, it means that the vertices of the triangle belong to three clusters, and the triangle will draw in three times. The accumulated result is shown in Figure 4.4.

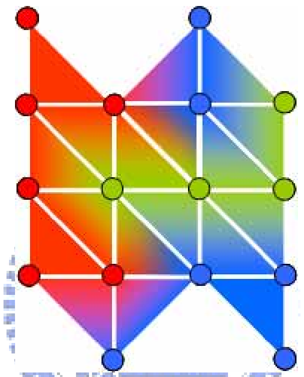


Figure 4.4: Accumulated mesh result

In our system, we apply alpha blending function in graphic card to accumulate the result of partitions. We measure the "average triangle overdraws" in our system to represent the triangles overdraw instead of total rendering triangles. The average triangles overdraw equals to divide the total triangles rendered by total triangles of the objects. The equation is as following.

$$\boldsymbol{Avg\ OD = \frac{Total\ rendered\ triangles}{Total\ ojbects\ triangles}} \qquad (8)$$

In the result, we compare the rendering speed by listing the average triangles overdraw and frames per second (FPS). The average triangles overdraw will affect the rendering speed directly.

4.2 The geometry coherence

In Chapter 3, we cluster the PRT vectors by calculating the Gaussian distances between the PRT vector and mean vectors of clusters. Some vertices may cluster into a clusters but not close in 3D space. It will increase the extra triangle overdraws. The geometry coherence is an idea to reduce the extra triangle overdraws. The main concept of geometry coherence is that" When the vertices are closer, the PRT vectors are similar". In our system, we add the geometry coherence in the LBG clustering algorithm module. When clustering the PRT vectors, we not only calculate the Gaussian distances of vectors but also calculate the Gaussian distances of positions.

For example in Figure 4.5, the purple point is the data point we want to cluster. The red points are in one cluster and blue points are in another cluster.

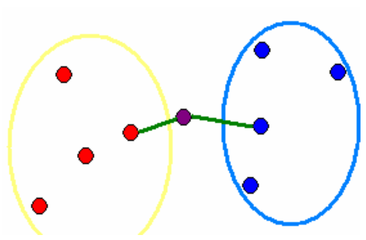


Figure 4.5: The Geometry Coherence example

In our system, we cluster the data by calculating the Gaussian distances of PRT vector plus the Gaussian distances of position vector. The equation is shown as following.

$$\boldsymbol{E_{ij}} = \left\| \vec{V}_i - \boldsymbol{mean}_j \right\| + \alpha \cdot \boldsymbol{GeoCoherence}_j \tag{9}$$

The "***i***" means point ***i***-th and the "***j***" means the ***j***-th cluster. "***α***" is a user given value to control the effect of geometry coherence. The "***GeoCoherence***" is defined as finding the minimum distance of the data and the points in a cluster. The equation is shown as following.

$$\boldsymbol{GeoCoherence}_j = \min_{\forall point\ k \in cluster\ j} \{ \left\| \vec{P}_i - \vec{P}_k \right\| \} \tag{10}$$

After applying the geometry coherence when clustering, the triangles overdraw is reduced.

4.3 The clusters selection algorithm

Sloan et al. [21] introduces the idea of super-clusters to reduce the triangles overdraw. But they don’t give a proper method for selecting the clusters for super-clusters. In the following, we discuss a clusters selection algorithm for super-clusters.

In the Section 4.1, we discuss the reason of triangles overdraw. Although we can not place all clusters in one pass, but we can place partial clusters of all in one pass. Recalling the example in Section 4.1, if we want to place all clusters into registers, we need 4860 float4 registers. The registers requirement of one cluster is 81 float4 registers. So we can place 256/81, 3 clusters in one pass rendering. This is the main concept of super-clusters. Our goal is to establish an algorithm to select clusters for a super-cluster. This will reduce the triangle overdraw again.

The pseudo code of our algorithm is shown in following Figure 4.6.

```
Step 1: Find the cluster which has the most data and put in cluster C.
Step 2: Find the cluster which can reduce the most triangles overdraws of cluster C.
Step 3: Merge the cluster selected by step 2 into cluster C.
Step 4: If all clusters are selected, then exit.
        Else if the registers are full, then go to step 1 and empty the cluster C.
        Else go to step 2 to find next cluster.
```

Figure 4.6: The clusters selecting algorithm for super-clusters

In step 3, “merge the cluster” does not mean merge the principle component data of two clusters. It means that we will concern both triangle overdraw reduction effect in the next time selecting another cluster, i.e. Step 2.

For example, in Figure 4.7, every circle represents a cluster. The intersection of two circles means the reduced triangle overdraws when they are selected into the same super-cluster. First, we select the red cluster as the initial cluster. Next, we select the green cluster and merge these two clusters to form a super-cluster ***C***. The reason is that the green cluster can reduce 8 triangle overdraws which is greater than any other clusters. Next, we select the purple cluster and merge into ***C***. We do not select blue cluster because the purple cluster will reduce more triangle overdraws. So, the "merge" means merging the triangle overdraw reduction effects when we select another cluster into a super-cluster in the next iteration.

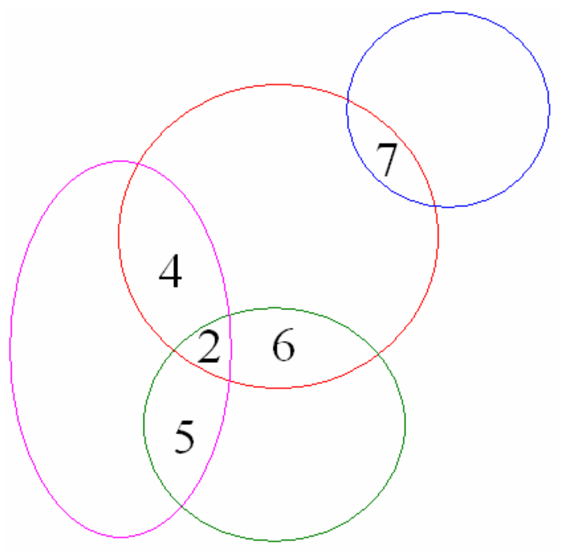


Figure 4.7: An example of clusters selection

Chapter 5
Experimental Results

In this chapter, we demonstrate experimental results of our purposed approach. Our system is implemented in C++ language and DirectX 9.0c [16]. It is working on a Pentium IV 3.4GHz CPU and an NVIDIA GeForce 6800GT graphic card.

In the testing examples, we compare our results with two previous methods which are the original CPCA method and the CPCA triangle overdraws reduced method. The resolution of image size is 1280 $\times$ 948 pixels. All the testing models are shown in the Figure 5.1. Table 5.1 shows the details of these models. The first row refers to the number of vertices of them. The second row refers to the number of polygons of them. The third row refers to the number of principle components we apply in the following experimental results. The fourth row refers to the total PRT data size before compression.

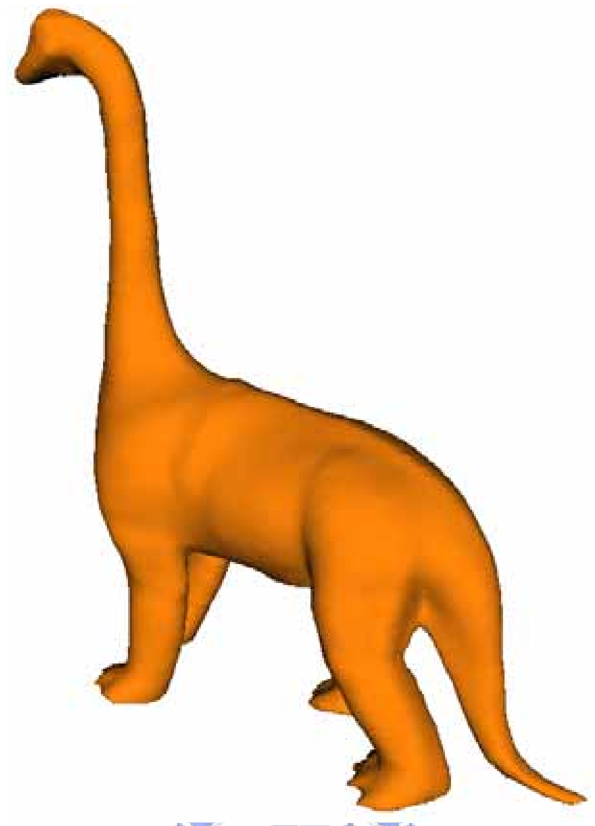


Figure 5.1 (a): The “Dino” model

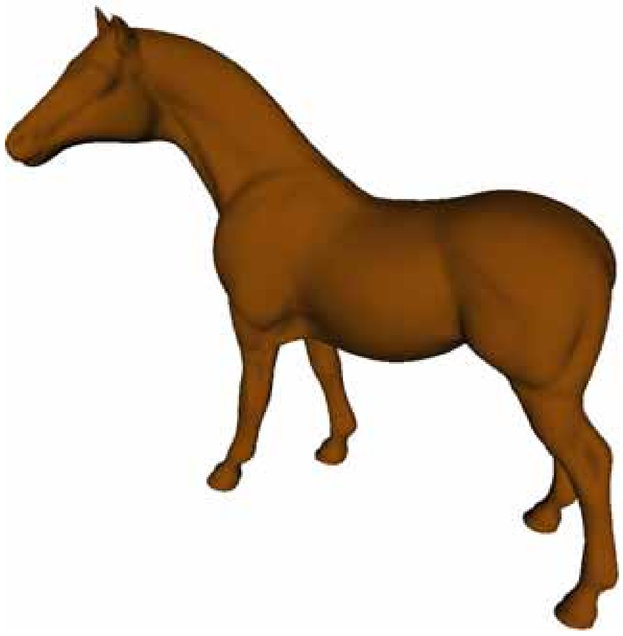


Figure 5.1 (b): The “Horse” model

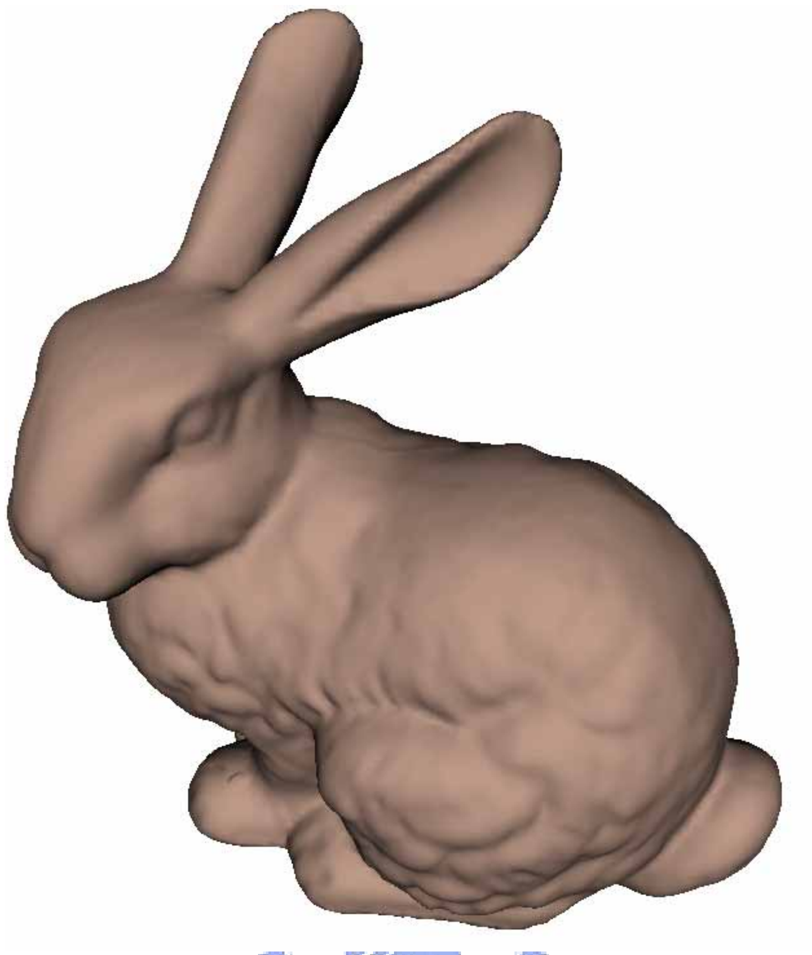


Figure 5.1(c): The “Bunny” model

	Dino	Horse	Bunny
Vertices	23984	19851	34834
Triangles	47904	39698	69451
# of Principle Components	8	8	8
PRT Data Size	355 MB	294 MB	516 MB

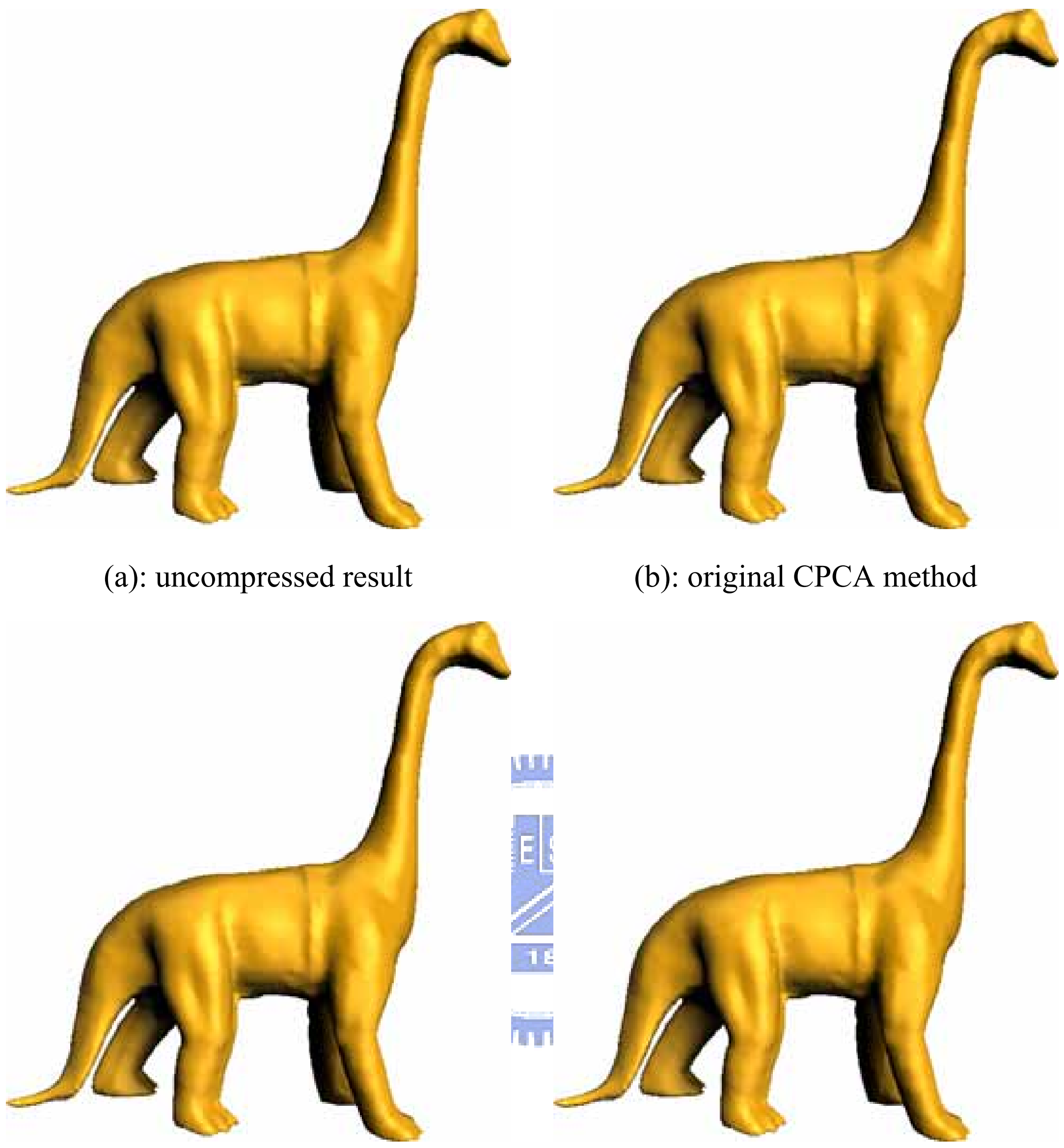
Table 5.1: The models comparisons

In the following tables, the first column refers to the methods. The second column refers to the clustering time in LBG algorithm module. The third column refers to the average triangles overdraw. The last column refers to the frames per second (FPS). The fifth column refers to the average squared error (SE).

First, the measurement of the "Dino" model is shown as follows. The material is applying Phong BRDF model. Table 5.2 shows the comparison with the other methods where the number of clusters is 64. Figure 5.2 shows the rendering result. Figure 5.2 (a) shows the uncompressed rendered result. Figure 5.2 (b) shows the rendered result of the original CPCA method without applying triangle overdraws reduction. Figure 5.2 (c) shows the result with applying triangle overdraws reduction. Figure 5.2 (d) shows the rendered result which applies our method.

	Clustering time	Average triangles overdraw	FPS	Average SE
Original CPCA	3 m 53 sec	1.427584	27.57	0.033846
CPCA overdraw red.	5 m 42 sec	1.383141	28.12	0.032895
Our method	2 m 18 sec	1.309619	32.37	0.047968

Table 5.2: Dino model, 64 clusters, α=0.05



(a): uncompressed result

(b): original CPCA method

(c): CPCA triangles overdraw red.

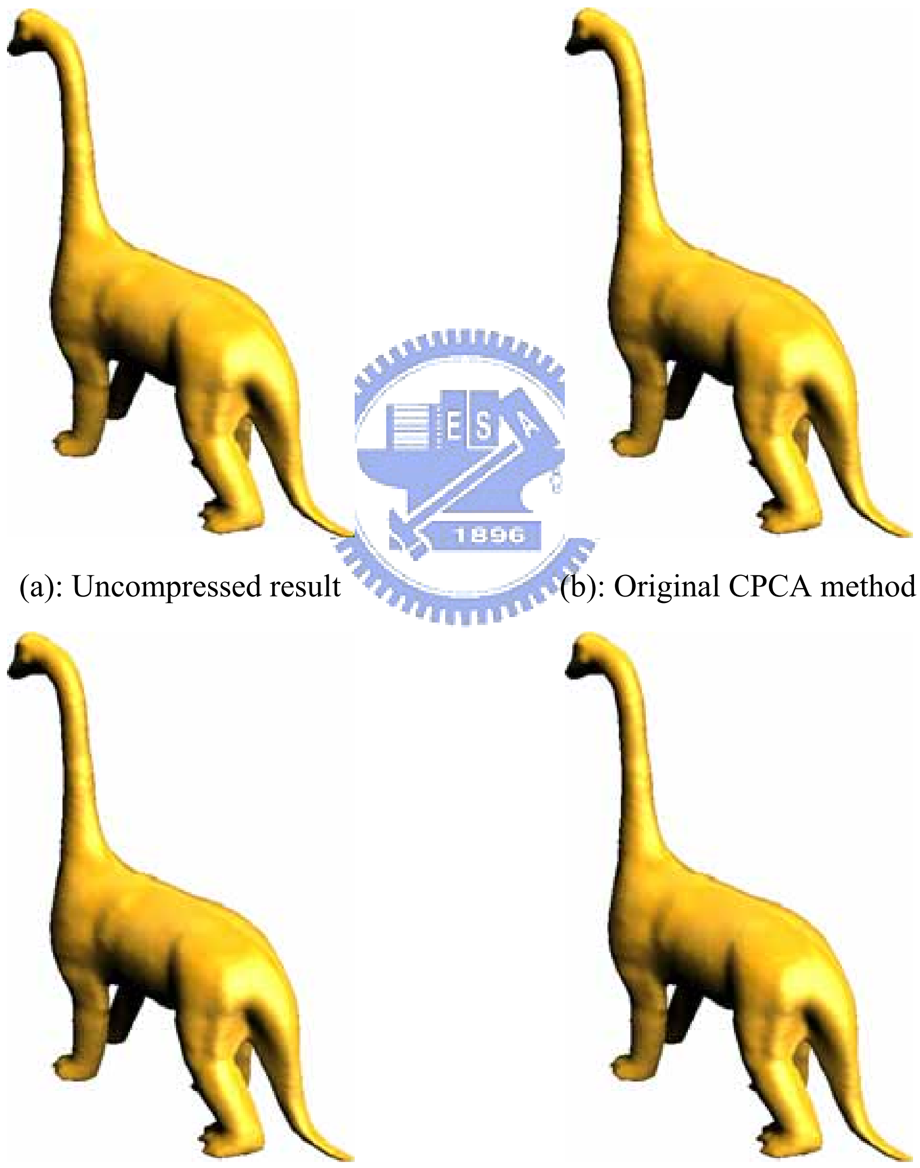
(d): our method

Figure 5.2: The “Dino” model comparisons, 64 clusters

Table 5.3 shows the results that the number of clusters is 256. Figure 5.3 demonstrates the rendering results of methods. We change both the view and lighting directions.

	Clustering time	Average triangles overdraw	FPS	Average SE
Original CPCA	27 m 4 s	1.733425	22.93	0.005371
CPCA overdraw red.	27 m 27 s	1.646063	24.43	0.006344
Our method	7 m 31 s	1.404810	30.52	0.012216

Table 5.3: The "Dino" model, 256 clusters, α=0.05



(a): Uncompressed result (b): Original CPCA method

(c): CPCA triangles overdraw red. (d): Our method

Figure 5.3: The "Dino" model comparisons, 256 clusters

Similarly the result for 128 and 192 clusters are shown in Tables 5.4 and 5.5 respectively.

	Clustering time	Average triangles overdraw	FPS	Average SE
Original CPCA	12 m 38 s	1.556363	26.58	0.014452
CPCA overdraw red.	12 m 17 s	1.505114	27.15	0.014756
Our method	4 m 10 s	1.359051	31.91	0.027246

Table 5.4: The "Dino" model, 128 clusters, $\boldsymbol{\alpha}$=0.05

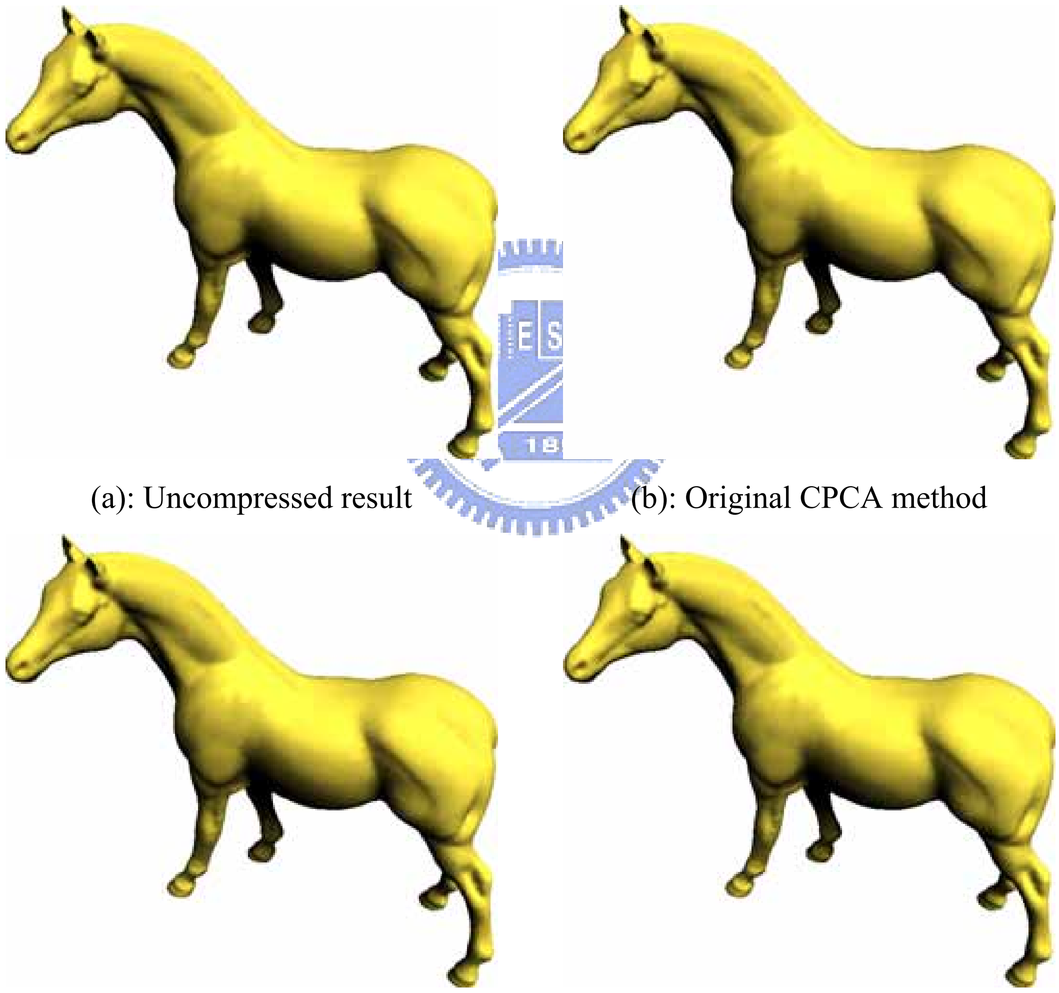
	Clustering time	Average triangles overdraw	FPS	Average SE
Original CPCA	15 m 15 s	1.637316	24.62	0.007938
CPCA overdraw red.	19 m 46 s	1.593019	25.60	0.008764
Our method	4 m 55 s	1.382953	30.89	0.017335

Table 5.5: The "Dino" model, 192 clusters, $\boldsymbol{\alpha}$=0.05

Next example is a "Horse" model. The material is applying the Phong BRDF model. Table 5.6 shows the comparing results, where the number of clusters is 64. Figure 5.4 displays the rendering results.

	Clustering time	Average triangles overdraw	FPS	Average SE
Original CPCA	3 m 46 s	1.510882	27.70	0.029795
CPCA overdraw red.	4 m 19 s	1.449997	29.38	0.029944
Our method	5 m 44 s	1.417578	30.51	0.029789
CPCA overdraw red. + Our method	4 m 30 s	1.344148	32.11	0.030263

Table 5.6: The "Horse" model, 64 clusters, $\boldsymbol{\alpha}$=0.05



(a): Uncompressed result (b): Original CPCA method

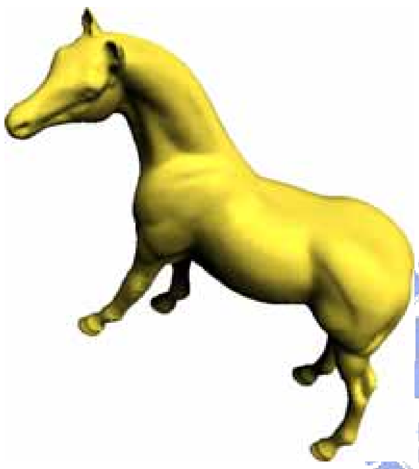
(c): CPCA triangles overdraw red. (d): Our method

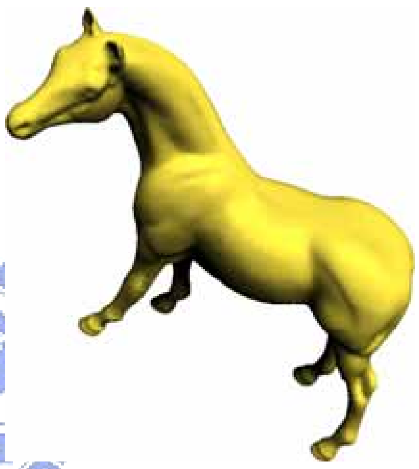
Figure 5.4: The "Horse" model comparisons, 64 clusters

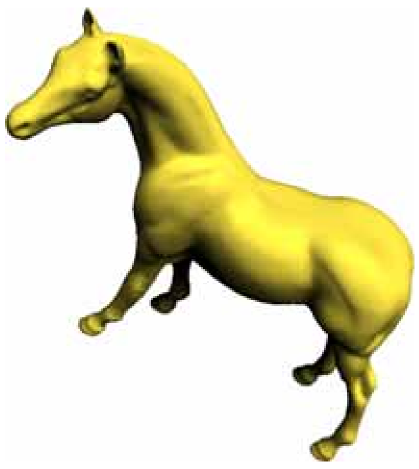
Then we increase the number of clusters to 256. The comparing results are shown in Table 5.7 and the rendering results are shown in Figure 5.5.

	Clustering time	Average triangles overdraw	FPS	Average SE
Original CPCA	16 m	1.825407	25.18	0.006340
CPCA overdraw red.	16 m 56 s	1.724545	27.70	0.007333
Our method	20 m 17 s	1.632803	30.03	0.006301
CPCA overdraw red. + Our method	13 m 40 s	1.574815	32.54	0.007226

Table 5.7: The "Horse" model, 256 clusters, α=0.05


(a): Uncompressed result


(b): Original CPCA method


(c): CPCA triangles overdraw red.

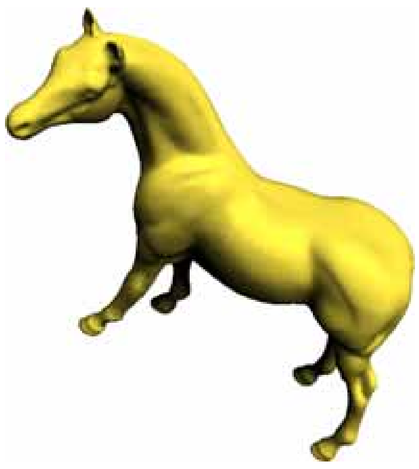

(d): Our method

Figure 5.5: The "Horse" model comparisons, 256 clusters

For 128 and 192 clusters the comparisons are shown in Tables 5.8 and 5.9.

	Clustering time	Average triangles overdraw	FPS	Average SE
Original CPCA	11 m 40 s	1.664971	26.75	0.014491
CPCA overdraw red.	7 m 46 s	1.599350	28.49	0.016006
Our method	15 m 58 s	1.506927	31.23	0.014509
CPCA overdraw red. + Our method	6 m 9 s	1.474205	32.47	0.015547

Table 5.8: The "Horse" model, 128 clusters, α=0.05

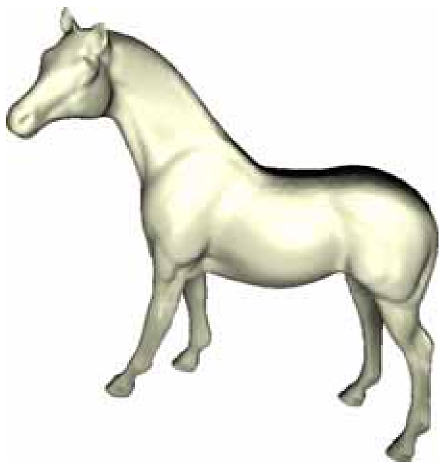
	Clustering time	Average triangles overdraw	FPS	Average SE
Original CPCA	13 m 51 s	1.739609	25.43	0.009147
CPCA overdraw red.	12 m 46 s	1.673963	27.06	0.009875
Our method	17 m 57 s	1.570583	30.32	0.009132
CPCA overdraw red. + Our method	13 m 54 s	1.511764	31.16	0.010184

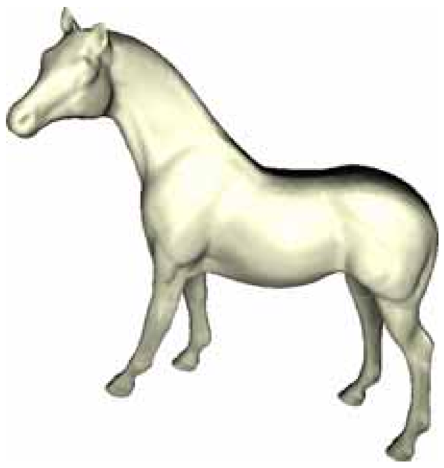
Table 5.9: The "Horse" model, 192 clusters, α=0.05

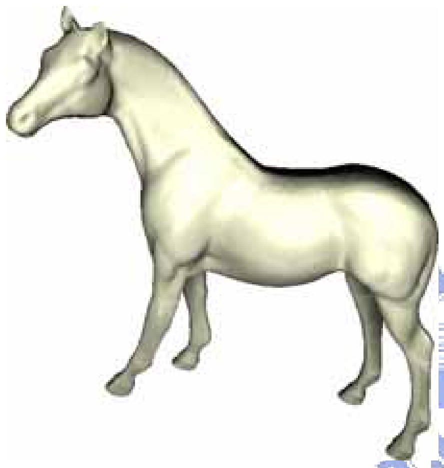
Next, we apply the Cook-Torrance BRDF model [2]. Table 5.10 shows the result where the number of clusters is 64. Figure 5.6 shows the rendering results.

	Clustering time	Average triangles overdraw	FPS	Average SE
Original CPCA	3 m 35 s	1.518691	27.75	0.258778
CPCA overdraw red.	4 m 18 s	1.469621	29.06	0.255928
Our method	5 m 19 s	1.408963	31.02	0.258831
CPCA overdraw red. + Our method	3 m 4 s	1.378961	31.59	0.259388

Table 5.10: The "Horse" model, 64 clusters, α=0.05


(a): Uncompressed result


(b): Original CPCA method


(c): CPCA triangles overdraw red.

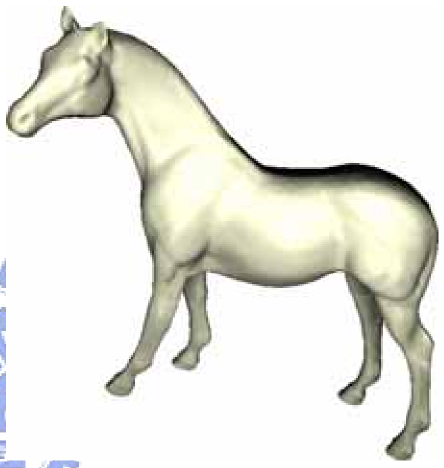
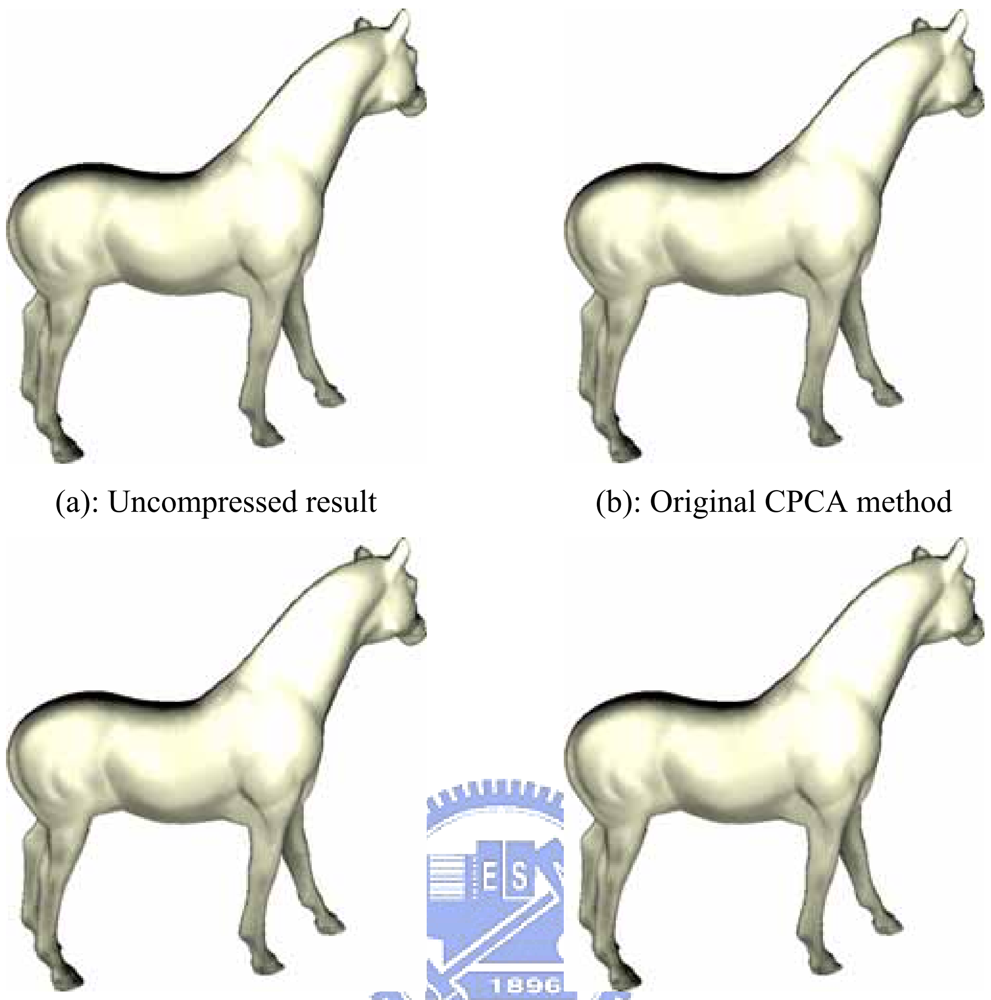

(d): Our method

Figure 5.6: The "Horse" model comparisons, 64 clusters

Next for 256 clusters, the results are shown in Table 5.11 and Figure 5.7.

	Clustering time	Average triangles overdraw	FPS	Average SE
Original CPCA	14 m 23 s	1.799159	25.32	0.064937
CPCA overdraw red.	16 m 56 s	1.755126	26.41	0.069018
Our method	18 m 5 s	1.640612	30.01	0.064994
CPCA overdraw red. + Our method	19 m 17 s	1.592372	32.33	0.064323

Table 5.11: The "Horse" model, 256 clusters, α=0.05



(a): Uncompressed result (b): Original CPCA method

(c): CPCA triangles overdraw red. (d): Our method

Figure 5.7: The “Horse” model comparisons, 256 clusters

For 128 and 192 clusters, Tables 5.12 and 5.13 show the results.

	Clustering time	Average triangles overdraw	FPS	Average SE
Original CPCA	7 m 14 s	1.643282	26.64	0.136315
CPCA overdraw red.	8 m 9 s	1.608796	27.45	0.137434
Our method	10 m 21 s	1.519724	30.04	0.136325
CPCA overdraw red. + Our method	9 m 22 s	1.457983	31.62	0.136876

Table 5.12: The “Horse” model, 128 clusters, α=0.05

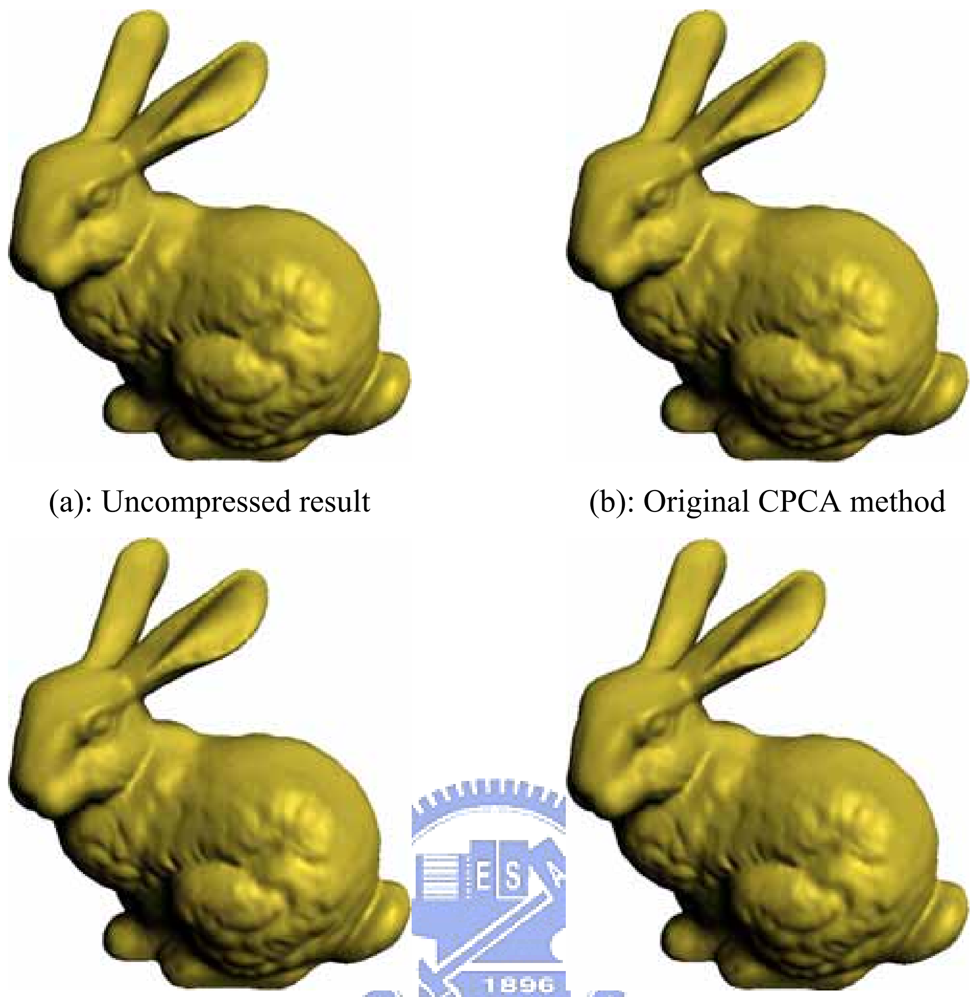
	Clustering time	Average triangles overdraw	FPS	Average SE
Original CPCA	10 m 48 s	1.729105	25.76	0.089433
CPCA overdraw red.	10 m 24 s	1.696131	26.58	0.093385
Our method	14 m 3 s	1.584186	30.01	0.089484
CPCA overdraw red. + Our method	12 m 4 s	1.560885	30.24	0.092139

Table 5.13: The “Horse” model, 192 clusters, ***α***=0.05

Finally, we use the “Bunny” model for testing. We apply the Cook-Torrance BRDF model. Table 5.14 shows the results for 256 clusters. Figure 5.8 shows the rendering results.

	Clustering time	Average triangles overdraw	FPS	Average SE
Original CPCA	26 m 49 s	1.529308	23.05	0.054466
CPCA overdraw red.	31 m 26 s	1.467294	24.45	0.058319
Our method	38 m 13 s	1.413759	25.89	0.054501
CPCA overdraw red. + Our method	34 m 47 s	1.378627	26.77	0.055894

Table 5.14: The “Bunny” model, 256 clusters, ***α***=0.05



(a): Uncompressed result (b): Original CPCA method

(c): CPCA triangles overdraw red. (d): Our method

Figure 5.8: The “Bunny” model comparisons, 256 clusters

Tables 5.15, 5.16, and 5.17 show the results of 64 clusters, 128 clusters, and 196 clusters.

	Clustering time	Average triangles overdraw	FPS	Average SE
Original CPCA	5 m 40 s	1.305410	25.86	0.221043
CPCA overdraw red.	7 m 18 s	1.275172	26.67	0.224016
Our method	10 m 6 s	1.239781	27.72	0.221096
CPCA overdraw red. + Our method	7 m 14 s	1.207974	28.53	0.225144

Table 5.15: The “Bunny” model, 64 clusters, $\boldsymbol{\alpha}$=0.05

	Clustering time	Average triangles overdraw	FPS	Average SE
Original CPCA	12 m 46 s	1.412708	25.07	0.118658
CPCA overdraw red.	15 m 8 s	1.345726	26.65	0.117374
Our method	19 m 39 s	1.300917	27.61	0.118617
CPCA overdraw red. + Our method	17 m 17 s	1.285856	28.16	0.113184

Table 5.16: The “Bunny” model, 128 clusters, ***α***=0.05

	Clustering time	Average triangles overdraw	FPS	Average SE
Original CPCA	22 m 19 s	1.471411	24.65	0.073271
CPCA overdraw red.	19 m 25 s	1.420354	25.89	0.076507
Our method	31 m 52 s	1.356309	27.49	0.073280
CPCA overdraw red. + Our method	22 m 35 s	1.337691	27.83	0.077189

Table 5.17: The “Bunny” model, 192 clusters, ***α***=0.05

Next, we change the lighting condition to the environment lighting. We show the rendering results in the following figures.

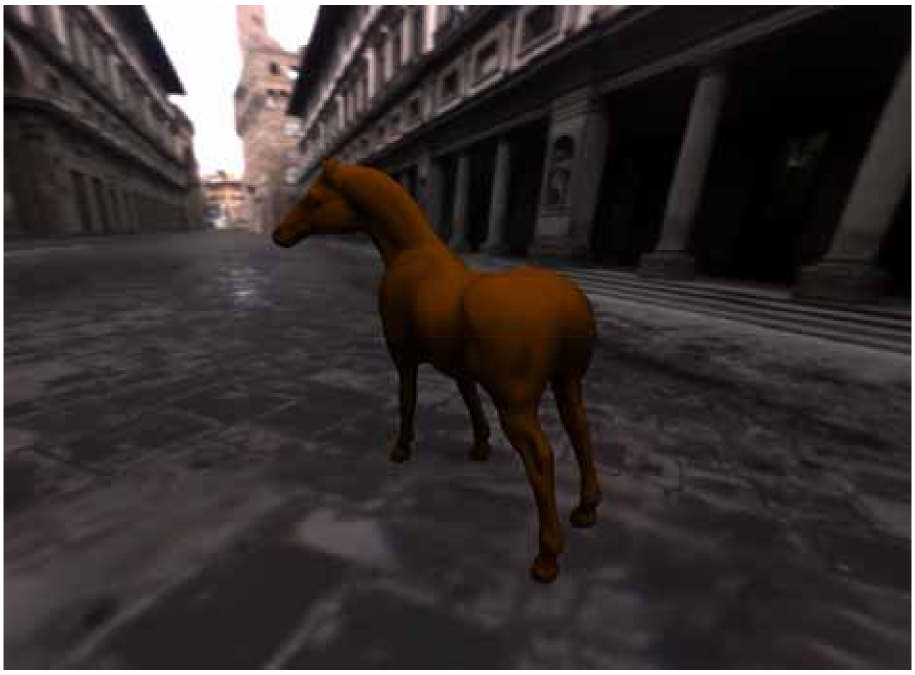


Figure 5.9: The “Horse” model with environment lighting

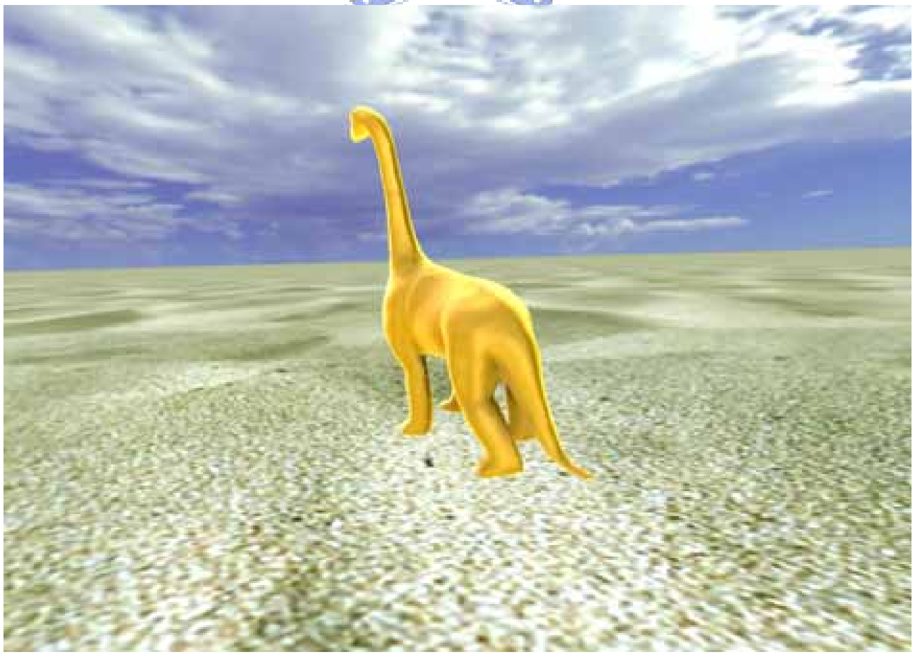


Figure 5.10: The “Dino” model with environment lighting

Chapter 6
Conclusions and Future Works

In this thesis, we propose two methods to reduce the triangles overdraw. One considers Geometry Coherence and the other uses the clusters selection algorithm.

Geometry Coherence may reduce the triangle overdraws, but increase the average square errors. Although the average square errors increase, the rendered images do not differ very much perceptually.

The clusters selection algorithm may also reduce the triangle overdraws. The number of clusters for a super-cluster will affect the number of reduced triangle overdraws. When the number of clusters for a super-cluster is smaller, the number of reduced triangle overdraws is smaller. Oppositely, when the number of clusters for a super-cluster is larger, the number of reduced triangle overdraws is larger.

Our proposed approaches increase the real-time performance of global illumination. Our approaches can be applied to 3D games or virtual reality applications.

However, there are still some issues left for future research.

(1) There could be other coherences may enhance the performance further.

(2) The Geometry Coherence may increase the average squared error. We may try another approach to control the average squared error.

(3) The LBG algorithm is a simple clustering method. We may try to find another clustering algorithm to decrease the clustering time.

References

[1] CHEN, W.-C., BOUGUET, Y.-V., CHU, M. H., AND GRZESZCZUK, R. 2002. Light Field Mapping: Efficient Representation and Hardware Rendering of Surface Light Fields. ACM Transactions on Graphics, 21(3), 447-456.

[2] COOK, R. AND TORRANCE, K. 1982. A Reflectance Model for Computer Graphics. ACM Transactions on Graphics, 1(1), 7–24.

[3] DANA, K, VAN GINNEKEN, B, NAYAR, S, AND KOENDERINK, J. 1999. Reflectance and Texture of Real World Surfaces. ACM Transactions on Graphics, 18(1), 1-34.

[4] GORAL, C. M., TORRANCE, K. E., GREENBERG, D. P., AND BATTAILE, B. 1984. Modeling the Interaction of Light between Diffuse Surfaces. Proceedings of SIGGRAPH '84, 213-222.

[5] GORTLER, S. J., GRZESZCZUK, R., SZELISKI, R., AND COHEN, M. F. 1996. The Lumigraph. Proceedings of SIGGRAPH '96, 43-54.

[6] IVANIC, J. AND RUEDENBERG, K. 1996. Rotation Matrices for Real Spherical Harmonics, Direct Determination by Recursion. Journal of Physical Chemistry A, 100(15), 6342-6347.

[7] IVANIC, J. AND RUEDENBERG, K. 1998. Additions and Corrections: Rotation Matrices for Real Spherical Harmonics. Journal of Physical Chemistry A, 102(45), 9099-9100.

[8] JAMES, D. L. AND FATAHALIAN, K. 2003. Precomputing Interactive Dynamic Deformable Scenes. ACM Transactions on Graphics, 22(3), 879- 887.

[9] JENSEN H. W. 1996. Global Illumination Using Photon Maps. Proceedings of the Eurographics Workshop on Rendering '96, 21-30.

[10] JOLLIFFE, I. T. 2002. Principle Component Analysis, 2nd Edition. Springer-Verlag.

[11] LEHTINEN, J. AND KAUTZ, J. 2003. Matrix Radiance Transfer. Proceedings of the ACM Symposium on Interactive 3D Graphics 2003, 59-64.

[12] LEVOY, M. AND HANRAHAN, P. 1996. Light Field Rendering. Proceedings of SIGGRAPH '96, 31-41

[13] LINDE, Y., BUZO, A., AND GRAY, R. 1980. An Algorithm for Vector Quantizer Design, IEEE Transactions on Communication, 28(1), 84-95.

[14] LIU, X., SLOAN, P.-K., SHUM, H.-Y., AND SNYDER, J. 2004. All-Frequency Precomputed Radiance Transfer for Glossy Objects. Proceedings of the Eurographics Symposium on Rendering 2004, 337-344.

[15] MATUSIK, W., PFISTER, H., NGAN, A., BEARDSLEY, P., ZIEGLER, R., AND MCMILLAN L. 2002. Image-Based 3D Photography using Opacity Hulls. ACM Transactions on Graphics, 21(3), 427- 437.

[16] MICROSOFT. 2005. DirectX 9.0c SDK (April 2005). http://www.microsoft.com/DirectX.

[17] MILLER, G., RUBIN, S., AND PONCELEN, D. 1998. Lazy Decompression of Surface Light Fields for Pre-computed Global Illumination. Proceedings of the Eurographics Workshop on Rendering '98, 281- 292.

[18] NG, R., RAMAMOORTHI, R., AND HANRAHAN, P. 2003. All-Frequency Shadows Using Non-Linear Wavelet Lighting Approximation. ACM Transactions on Graphics, 22(3), 376-381.

[19] NG, R., RAMAMOORTHI, R., AND HANRAHAN, P. 2004. Triple Product Wavelet Integrals for All-Frequency Relighting. ACM Transactions on Graphics, 23(3), 477–487

[20] SELIM, S. Z. AND ISMAIL, M. A. 1984. K- Means-Type Algorithms: A Generalized Convergence Theorem and Characterization of Local Optimality. IEEE Transactions on Pattern Analysis and Machine Intelligence, 6(1), 81-87.

[21] SLOAN, P., HALL, J., HART, J., AND SNYDER, J. 2003. Clustered Principal Components for Pre- computed Radiance Transfer. ACM Transactions on Graphics, 22(3), 382-391.

[22] SLOAN, P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. ACM Transactions on Graphics, 21(3), 527-536.

[23] SLOAN, P., LIU, X., SHUM, H., AND SNYDER, J. 2003. Bi-Scale Radiance Transfer. ACM Transactions on Graphics, 22(3), 370-375.

[24] VASILESCU, M. A. O. AND TERZOPOULOS, D. 2004. TensorTextures: Multilinear Image-Based Rendering. ACM Transactions on Graphics, 23(3), 336-342.

[25] WESTIN, S., ARVO, J., AND TORRANCE, K. 1992. Predicting Reflectance Functions from Complex Surfaces. Proceedings of SIGGRAPH '92, 255-264.

[26] WHITTED, T. 1980. An Improved Illumination Model for Shaded Display. Communications of the ACM, 23(6), 343–349.

[27] WOOD, D., AZUMA, D., ALDINGER, K., CURLESS, B., DUCHAMP, T., SALESIN, D., AND STUETZLE, W. 2000. Surface Light Fields for 3D Photography. Proceedings of SIGGRAPH 2000, 287- 296.