Faculty of Mathematics, Physics and Informatics, Comenius University, Bratislava

Real-time friendly representation of arbitrary BRDF with appearance industry measurements

Ivan Šeďo

2007

Real-time friendly representation of arbitrary BRDF with appearance industry measurements

Ivan Šeďo

Department of Applied Informatics Faculty of Mathematics, Physics and Informatics, Comenius University, Bratislava

Thesis advisor:
Doc. RNDr. Roman Ďurikovič, PhD.

May 7, 2007

Bratislava

Assignment: Study available papers about real-time visualization of complex (*e.g.* glossy) BRDFs. Choose a method allowing dynamic change of lighting, BRDF and viewing position. Implement chosen method in form of a visualization tool. Try to use GPU for computations if it would bring significant speed up. Implement virtual glossmeter, compare and validate its measurement to measurements of a real-world glossmeter.

Hereby I honestly declare that I have worked on this thesis independently using only the listed literature.
Bratislava, 5. May 2007

Acknowledgements

Ďakujem môjmu školiteľovi, Doc. RNDr. Romanovi Ďurikovičovi, za motiváciu.

Ďakujem mojim rodičom za podporu počas celého môjho štúdia tak ďaleko od domova.

Ďakujem Janke za jej trpezlivosť počas písania tejto práce.

Abstract: This thesis is devoted to the problem of representing bidirectional reflectance distribution functions (BRDF) as ordinary functions and also representing precomputed radiance transfer data of a static scene, because these data (light transfer operator) is among other the derivative carrier of informations about BRDF used. Later we discuss the need and principles of industrial apearance measurement attributes (mainly gloss). Another part of our work is the implementation of the environment mapping using efficient wavelet rotation. This technique allows dynamic change of lighting, viewpoint and BRDF while preserving all-frequency details. Other implemented feature is the virtual glossmeter. As a feedback to the user it provides gloss unit value measured from the used parametrizable BRDF. In the end we experiment with possible speedup by means of GPU computations and presence of mult-core CPUs.

Keywords: GPU, spherical harmonics, real-time rendering, BRDF, wavelets, appearance, gloss, virtual glossmeter

Abstrakt: Táto práca sa venuje problémom reprezentácie obojsmerných distribučnýchch funkcií (BRDF) ako funkcií takých i reprezentáciou dát predpočítaného prenosu radiancie v statickej scéne, pretože tieto dáta (operátor prenosu svetla) sú okrem iného sekundárnymi nositeľmi informácií o použitej BRDF. Neskôr sa v texte zaoberáme potrebou a princípmi priemyselných meraní vizuálnych atribútov (predovšetkým lesku) skúmaných materiálov. Súčasťou práce je implementácia pohľadovo závislého mapovania s využitím rotačných transformácií vlnkových koeficientov. Táto metóda umožňuje dynamickú zmenu osvetlenia, polohy pozorovateľa, BRDF objektov scény a zachováva vo všetkých. Súčasťou softvérového diela je aj implementácia virtuálnho leskomera, ktorý nám ako spätnú väzbu poskytuje hodnotu lesku nameranú z použitej používateľom parametrizovateľnej BRDF. V experimentálnej časti práce sa zaoberáme možnosťami urýchlenia implementácie pomocou vykonávania výpočtov na GPU a využívania viacjadrových procesorov.

Kľúčové slová: GPU, sférické harmonické funkcie, renderovanie v reálnom čase, BRDF vlnkové transformácie, vzhľad, lesk, virtuálny leskomer.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Definition of the Problem

Detailed realistic lighting, materials, interreflections and all-frequency shadows are important effects in realistic image synthesis. Conventional rendering methods (running on computers with limited computing power) for integrating over large-scale lighting environments are impractical for real-time rendering like games or they greatly increases the design cycle in applications for appearance industry.

Real-time, realistic global illumination encounters three obstacles:

- it shall model the complex, spatially-varying BRDFs of real materials (BRDF complexity)
- it requires integration over the hemisphere of lighting directions at each point (light integration)
- it shall account for bouncing/occlusion effects, like shadows, due to intervening matter along light paths from sources to receivers (light transport complexity)

Much research has focused on extending BRDF complexity (*e.g.*, glossy and anisotropic reflections), solving the light integration problem by representing incident lighting as a sum of directions or points.

A second line of research samples radiance and preconvolves it with kernels of various sizes. This solves the light integration problem but ignores

light transport complexities like shadows since the convolution assumes the incident radiance is unoccluded and unscattered. Finally, clever techniques exist to simulate more complex light transport.

From the other side, appearance of a product often determines the acceptability of a product to the end-user. The appearance of an object is the result of a complex interaction of the light incident on the object, the optical characteristics of the object, and human perception.

Appearance engineering rely on accurate physically based models of light interaction. Fast and accurate techniques for visualization of complex materials under realistic lighting may greatly decrease the design cycle of materials (coatings) in appearance industry.

1.2 Our contribution

In our work we implemented the method of efficient wavelet rotation for environment mapping in the form of visualization tool preserving all frequencies in BRDF and lighting environments. Our contribution to the original work lies in the support of multi-core CPUs, which speeds up the precomputation and rendering code almost linearly to the number of cores available.

Another contribution is the profit from the presence of fast programmable graphic processor, which we use to perform tileboarded nonstandard Haar wavelet transform. This is a step toward more dynamic changes of the lighting environment and it also overcomes the need of keeping rotation matrices in the memory, thus lowers the overall consumed memory.

In summary, we implemented standalone visualization tool enabling us to render highly specular materials under natural detail lighting environments at interactive rates, employing multi-core CPU and fast GPU to speed up computation, while giving the virtually measured numerical specular gloss value of rendered material as feedback to the user.

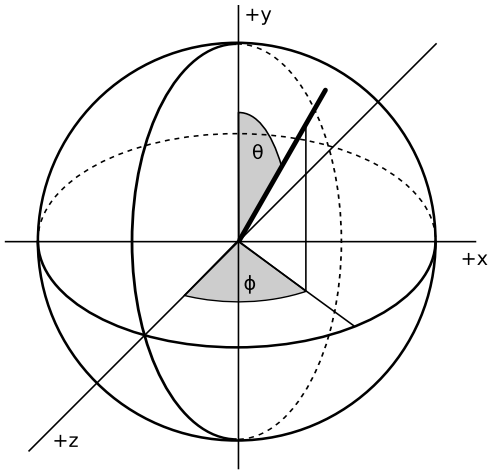


Figure 1.1: Spherical coordinate system

1.3 A Little of Math

1.3.1 Spherical coordinates

In lighting equations usually functions parametrized by direction appear (*e.g.* BRDF). A direction indicated by the unit vector can be represented by a point on the unit sphere. Spherical (polar) coordinates are natural for describing positions on a sphere.

Directions $\vec{\omega}$ can be used interchangeably with spherical coordinates (θ, ϕ), where θ denotes the azimuthal angle in the xz-plane from the x-axis with $0 \leq \theta \leq 2\pi$ and similarly ϕ denotes the polar angle from the y-axis with $0 \leq \phi \leq \pi$ as can be seen on Figure 1.1.

The complete spherical coordinates consist also of the distance (radius) r from a point on a sphere to the origin. As we are not interested in distance but just the direction (we assume the lighting to be at infinity) we ignore this by setting r equal to 1.

$$(x, y, z) = (sin\theta \sin\phi, cos\phi, cos\theta \sin\phi)$$
$$(\theta, \phi) = (\arctan\frac{x}{z}, \arccos\frac{y}{\sqrt{x^2 + y^2 + z^2}})$$

1.3.2 Basis Functions/Projection/Reconstruction

Let $f_1, f_2, \ldots, f_n$ is a set of 1D functions. We say that such functions are *orthogonal* if they satisfy

$$\int f_i(x) f_j(x)\, dx = \begin{cases} c_{ij} = 0 & i \neq j \\ c_{ij} \neq 0 & i = j \end{cases}$$

If $c = 1$ we say the functions are *orthonormal*. Given some function $f(x)$ and set of orthonormal basis functions $B_i(x)$ We can *project* function $f(x)$ into basis functions space getting coefficients

$$c_i = \int f(x) B_i(x)\, dx =: \langle f(x), B_i(x) \rangle$$

Reconstruction of the original function

$$f(x) = \sum c_i B_i(x)$$

In most cases we just want to approximate the original function $f(x)$ with function $\widetilde{f}(x)$ using less coefficients which also usually means introducing some error:

$$\widetilde{f}(x) = \sum_{i=0}^{N} c_i B_i(x)$$

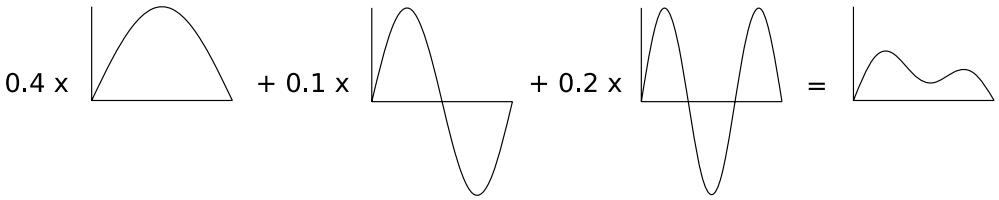


Figure 1.2: Example reconstruction with sinusoidal linear basis

We define *coupling coefficients*:

$$c_{ij} = \int_{S^2} \psi_i(\omega) \psi_j(\omega)\, d\omega$$

As long as we use orthonormal basis functions $c_{ij} = \delta_{ij}$ - Kronecker deltas.

Thus following simplification is possible:

$$
\begin{aligned}
B &= \int_{S^2} \left(\sum_i L_i \Psi_i(\omega) \right) \left(\sum_i T_i \Psi_i(\omega) \right) d\omega \qquad (1.1) \\
&= \sum_i \sum_j L_i T_j \int_{S^2} \psi_i(\omega) \psi_j(\omega)\, d\omega \\
&= \sum_i \sum_j L_i T_j c_{ij} = \sum_i \sum_j L_i T_j \delta_{ij} = \sum_i L_i T_i \\
&= T \cdot L
\end{aligned}
$$

Now we can clearly see that the double product integral of orthonormal functions simplifies to dot product of their approximation coefficients.

1.3.3 Spherical Harmonics (SH)

Spherical harmonics define an orthonormal linear basis over the sphere.

The real-valued SH basis function is traditionally represented by symbol y_l^m:

$$
y_l^m = \begin{cases} \sqrt{2} K_l^m \cos(m\theta) P_l^m(\cos\phi) & m > 0, \\ \sqrt{2} K_l^m \sin(-m\theta) P_l^{-m}(\cos\phi) & m < 0, \\ K_l^0 P_l^0(\cos\phi) & m = 0. \end{cases}
$$

where P_l^m are the associated Legendre polynomials and $K_l^m = \sqrt{\frac{(2l+1)(l-|m|)!}{4\pi(l+|m|)!}}$ are normalization coefficients.

Spherical harmonics are not suitable for nonlinear approximation, because they don't localize well in space domain. Small areas (high frequencies) would still need many harmonic coefficients for accurate approximation. Additionally, they have expensive rotation computation.

1.3.4 Wavelets

Wavelets are basis functions which represent a given function at multiple levels of detail. Due to their local support in both space and frequency domain, they are suited for sparse (*i.e.* they only require a small number of nonzero coefficients) approximations of (high frequency) signals. Locality

in space follows from their compact support, while locality in frequency follows from their smoothness (decay towards high frequencies) and vanishing moments (decay towards low frequencies). This allows compression and efficient computations.

Wavelets are generally considered to be superior at approximating high-frequency signals, such as detailed HDR image used for image based lighting in our case).

With their help function is expressed in terms of coarse shape plus multiple levels of details. Multiresolutional analysis is a mathematical framework for studying wavelets. The starting point of this framework is a sequence of closed nested vector subspaces V_i

$$V_0 \subset V_1 \subset V_2 \subset \cdots$$

with finer spaces (functions from them are of better resolution) having higher index and the coarser spaces having lower index with the most coarsest space V_0. The basis functions of space V_i are referred to as *scaling functions*. Wavelets . The basic idea of wavelets is that they encode difference between level of approximation *i.e.* they form a basis for a space complementing V_i in V_{i+1}. Such space is denoted W_i so that $V_{i+1} = V_i \oplus W_i$. Scaling functions and wavelets both satisfy refinement relations.

Fast $O(n)$ algorithms exist to calculate wavelet coefficients, making the use of wavelets efficient for many computational problems.

1dD scaling function is defined as:

$$\phi(x) = \begin{cases} 1 & \text{for } 0 \leq x < 1, \\ 0 & \text{otherwise.} \end{cases}$$

while 1D Haar wavelet function defined as:

$$\psi(x) = \begin{cases} 1 & \text{for } 0 \leq x < \frac{1}{2}, \\ -1 & \text{for } \frac{1}{2} \leq x < 1, \\ 0 & \text{otherwise.} \end{cases}$$

Normalized scaling and wavelet functions at level l are given by equa-

tions

$$\phi_{l,i}(x) = 2^{\frac{l}{2}}\phi(2^l x - i)$$

$$\psi_{l,i}(x) = 2^{\frac{l}{2}}\psi(2^l x - i)$$

Nonstandard two-dimensional Haar wavelet transform alternates between 1D Haar wavelet transform on all row and all columns. Two-dimensional nonstandard basis functions can be enumerated as follows

$$\begin{aligned}
\phi\phi(x,y) &:= \phi(x)\phi(y) \\
\phi\psi(x,y) &:= \phi(x)\psi(y) \\
\psi\phi(x,y) &:= \psi(x)\phi(y) \\
\psi\psi(x,y) &:= \psi(x)\psi(y)
\end{aligned}$$

and finally define the wavelets for the level l:

$$\begin{aligned}
\phi\phi_{0,0,0}(x,y) &= \phi\phi(x,y) \\
\phi\psi_{l,i,j}(x,y) &= 2^l\phi\psi(2^l x - i, 2^l y - j) \\
\psi\phi_{l,i,j}(x,y) &= 2^l\psi\phi(2^l x - i, 2^l y - j) \\
\psi\psi_{l,i,j}(x,y) &= 2^l\psi\psi(2^l x - i, 2^l y - j)
\end{aligned}$$

Each triplet (l, i, j) defines a wavelet square at level l and offset (i, j). Squares at level l have area $\frac{1}{4^l}$ and are disjoint. For a square image of size n with $N = n \times n$ pixels, there are $\log_2 n$ levels: $0, 1, \ldots, (\log_2 n - 1)$ and 4^l squares at level l

Nonlinear Wavelet Approximation

Ng *et al.* [NRH03] first proposed non-linear wavelet approximation of the lighting in the context of precomputed radiance transfer (PRT) Wavelets due to their compact support approximate lighting with features at all frequencies. Many works show, that just really small fraction of coefficients are necessary to approximate detailed photographed lighting, and even lower number of coefficients for synthetic lighting. Also only linear time for wavelet transform algorithm is necessary. Reasonable low number of

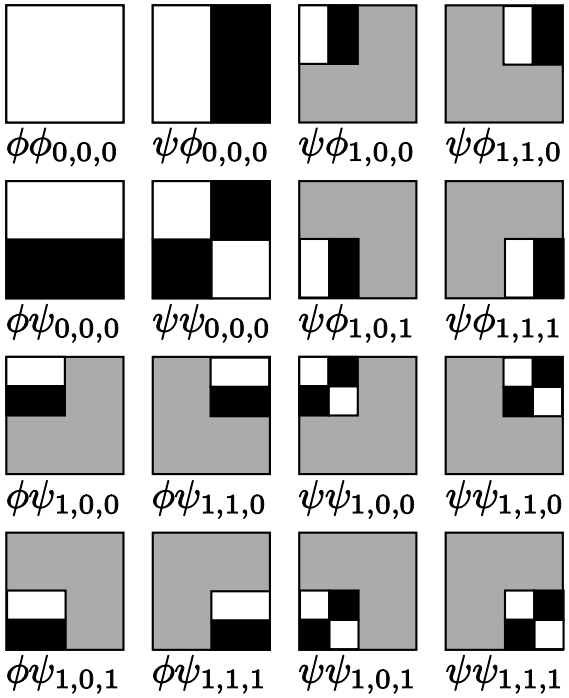


Figure 1.3: Nonstandard 2D Haar wavelet basis functions for V^2 (grey color represents zero, white positive and black negative values)

terms.

Methods of selecting the coefficients proposed by Ng *et al.* in [NRH03]:

Unweighted selection - importance of each coefficient is determined simply by absolute value of its magnitude - this is optimal choice for minimizing L^2 error

Area-weighted selection - importance of each wavelet coefficient is scaled by its area

Spherical Wavelets

Since functions involved in rendering equation, illumination, visibility and BRDF are all (hemi-)spherical functions it makes more sense to represent them in their intrinsic spherical domain. Schröder *et al.* [SS95] are the first who introduced efficient representation of functions defined over sphere

with spherical wavelets. Authors describe a simple technique for biorthogonal wavelets construction with customized properties as instances of a more general *lifting scheme*. The consider two important families of wavelet bases, interpolating and generalized Haar. They demonstrate how the lifting process can be used to improve wavelet basis properties so they can lead to better compression ratio. Subdivision schemes - geodesic subdivision scheme for several levels, beginning with the icosahedron (for reducing the least imbalance in area between constituent spherical triangles.

The bad thing about spherical wavelet or wavelets as such is that we still don't have analytical means to rotate them. In [WNLH06] and idea of computational rotation via rotation matrices was presented. Such rotation matrices are fundamental notion of linear algebra and they may be very large depending on the number of coefficients of source and destination domain. Rotations by matrix-vector multiplication would be deprecated due to its time cost. Fortunately, thanks to localization feature of wavelet domain those matrices are very sparse, which brings them back to the game.

1.3.5 Clustered Principal Component Analysis (CPCA)

Dimensionality reduction techniques have been widely adopted to analyze and compress data in computer graphics. Perhaps the most popular approach is the traditional PCA, which is a linear model and often is computed using *singular value decomposition* (SVD).

In PCA, data samples are transformed from a high-dimensional space into another low-dimensional sub-space spanned by only a few principal components (PCs). Thus, the original samples can be approximated by projecting them onto these PCs and expressing the original samples as a linear combination of PCs. A major drawback of the traditional PCA is the error of the approximation, because the whole data set (which may contain really large spectrum of data) is approximated by few principal components, and some of the original data could be approximated only very roughly. This is what the clustered principal component analysis tries to overcome.

CPCA resembles regular PCA, but we first partition signals (vectors, samples) into fewer each approximating the signal as an affine subspace.

We create clusters using vector quantization (VQ) in order to get clusters of data with similar statistical characteristics and consequently PCA is performed for each cluster separately, exploiting local linearity and thus reducing the error of approximation.

Let x_p is an n-dimensional signal at some sample point p on the surface. x_p stores the exiting radiance as a linear operator on light vector and usually has a form of vector for diffuse materials or matrix for glossy ones. In order to approximate the signal, we partition its samples into clusters and each cluster is approximated by an affine subspace.

$$x_p \approx \widetilde{x}_p = x_0 + w_1 x_1 + w_2 x_2 + \cdots + w_{n'} x_{n'}$$

with w_i being specific to each concrete sample x_p. Vectors $x_0, x_1, \ldots, x_{n'}$ are constant over and bound to this concrete cluster. We call x_0 the *cluster mean* and it together with other n' vectors form representative vectors of the cluster. In order to reduce signal dimensionality, we set $n \ll n'$. PCA can be seen as a trivial application of CPCA using only one cluster.

Various approaches have been studied to reduce error, from the simple methods like clustering the signal with LBG vector quantization, to more sophisticated methods like iterative PCA [SHHS03].

1.4 Compendium on PRT Methods

Under assumption that the object is not emissive, it is common to view light transport as a linear operator [SKS02] [NRH04]. The light that arrives to observer eye is a linear transformation of the distant incident lighting.

This operator $\mathcal{L}$, which we call the light transport operator, encodes the effects of the material properties and light transport interactions between different patches of the object. Thus we can describe the exit radiance $B(\vec{x}, \omega_o)$ at a point $\vec{x}$ (on the object) along the outgoing (viewing) directions ω_o) as the result of applying out integral operator $\mathcal{L}$ on the (distant) incident lighting L

$$B(\vec{x}, \omega_o) = (\mathcal{L}L)(\vec{x}, \omega_o) = \int_S T_{\vec{x},\omega_o}(\omega_i) L(\omega_i)\, d\omega_i$$

where $T_{\vec{x},\omega_o}$ is the integration kernel of $\mathcal{L}$, also sometimes called the *transport function*. For fixed $\vec{x}$ and ω_o, $T_{\vec{x},\omega_o}$ is a 2D function parametrized over the sphere S of incoming directions. It describes, for direction ω_i, the contribution of $L(\omega_i)$ to the total reflected radiance leaving $\vec{x}$ along ω_o. Substituting for the integration kernel we can derive the familiar reflectance equation for direct lighting.

With

$$T_{\vec{x},\omega_o} = f_r(\omega_i, \omega_o) V_{\vec{x}}(\omega_i)(\vec{n}_{\vec{x}})$$

where ω_i is the incoming directions $\vec{n}_{\vec{x}}$ is the surface normal at $\vec{x}$, $V_{\vec{x}}$ is a binary visibility functions, f_r is the bidirectional reflectance distribution functions, we get

$$B(\vec{x}, \omega_o) = \int_\Omega f_r(\omega_i, \omega_o) V_{\vec{x}}(\omega_i)(\vec{n}_{\vec{x}}) L(\omega_i)\, d\omega_i$$

However, the integration kernel $T_{\vec{x},\omega_o}$ is not limited to direct illumination, and can describe many other complex transport effects such as interreflections, refraction, self-occlusion, caustics, subsurface scattering, and other indirect lighting.

Unfortunately, direct evaluation of $\mathcal{L}L$ is feasible. Even assuming the distant lighting, light transport is 6D (two dimensions for each, lighting (assumed to be at infinity) and view direction and two dimensions for the surface position) and fundamentally all previous PRT work is concerned with approximating and compressing some slice of the light transport operator.

Precomputed light transport techniques compute a pixel color at rendertime based on a precomputed approximation of the light transport operator, which evaluates the contribution of all lighting directions for a given spatial location on a mesh and a given view direction.

The key issue common to all PRT techniques is the representation of the light transport operator. There are several criteria used to judge the quality of any proposed representation: rendered visual quality, compactness, storage costs, efficient integration/evaluation, angular and spatial frequency bandwidth (sharpness) of the illumination effects.

View-dependent effects are harder to address because they involve vari-

ation over the full 6D domain, and highlights can vary quickly over space and view direction

The precomputed radiance (PRT) algorithm [SKS02] has recently attracted much attention owing to its ability to allow real-time rendering of complex objects under dynamic lighting environments.

First PRT methods (SH based) either only handled low-frequency lighting environments, or suffer from the unwieldy size of PRT data sets even after compression. For dynamic scenes, the amount of PRT data sets further expands to an impractical degree for real-time applications. The enormous PRT data sets often prohibit high-quality rendering, subsequently restricting practical application of the PRT algorithm.

Typically, the transport operator is represented by linear basis functions such as spherical harmonics or wavelets, and the number of coefficients directly constrains the sharpness of the effects that can be handled.

Compression techniques such as the nonlinear truncation of linear basis or the use of separable approximations improve on both storage and computation, but the number of coefficients required to handle high-frequency effect remains large and the rendering cost is directly proportional.

The PRT algorithm can capture self-shadowing and self-interreflection effects from dynamic lighting environments. As a preprocess, PRT precomputes a solution to the light transport of a scene, and records the simulation results.

To decrease data storage and computational costs, the recorded data are compressed for efficient rendering at run-time. Low-frequency methods projected the per-vertex light transfer functions onto the spherical harmonics basis [SKS02] [LK03].

The coherence among vertices was then exploited using principal component analysis or CPCA [SHHS03] [LK03].

By contrast the all-frequency PRT methods [NRH03] [NRH04] [LSSS04] approximated the densely-sampled PRT data with sophisticated compression techniques, such as non-linear wavelet approximation [NRH03] [NRH04] [TL04] and BRDF factorization [TL04] [LSSS04].

However, previous compression schemes are inadequate for harnessing the power of PRT. Based on the SH basis, the low-frequency PRT algorithms may take tens of thousands of terms to represent all-frequency lighting and

shadowing effects.

As for the all-frequency PRT algorithms, the compressed data are still cumbersome for real-time rendering of objects with glossy BRDFs.

A more complex integral captures how a concave object shadows itself, where the integrand is multiplied by an additional transport factor representing visibility along each direction.

Improvements have been presented to incorporate more complex BRDFs [KSS02] [LK03], but the use of the spherical harmonics basis still limits these approaches to low-frequency incident lighting. Even with compression [SHHS03] the lighting cannot be high-frequency and compressed data are still cumbersome for real-time rendering of objects with glossy BRDFs.

Ng *et al.* [NRH03] use nonlinear approximation (truncation) of wavelets as basis function instead of spherical harmonics. With as few as 100 coefficients, they are able to incorporate high-frequency lighting effects (for diffuse surfaces or static view only). The significant coefficients are selected based on incident lighting which prevents the use of highly glossy surfaces. Arbitrary BRDFs can be incorporated using a method to evaluate triple product integrals [NRH04], but is limited to direct lighting only. Arbitrary BRDFs and wavelets can be combined using separable BRDFs because precomputation is done on a per-BRDF basis (multiple BRDFs also incur additional memory consumption and storage costs). Furthermore, high-frequency BRDFs can only be represented using prohibitively many terms in the separable approximation [LSSS04].

1.4.1 SH Based Methods

Sloan *et al.* were the first who presented in [SKS02] new real-time method for rendering diffuse and glossy object in low-frequency lighting environments that captures soft shadows, interreflections and caustics.

By representing both incident radiance (lighting) and transfer functions using using low-order spherical harmonics linear basis, authors exploit the linearity to reduce the light integral to a simple dot product between corresponding coefficient vectors (diffuse receivers) or a simple linear transform of the lighting coefficient vector through a small transfer matrix (glossy receivers).

If the object is diffuse, a transfer vector at each point on the object is dotted with the lighting's coefficients to produce correctly self-scattered shading. If the object is glossy, a transfer matrix is applied to the lighting coefficients to produce the coefficients of a spherical functions representing self-scattered incident radiance at each point. This functions is convolved with the object's BRDF and then evaluated at the view-dependent reflection direction to produce the final shading.

Dynamic, local lighting is handled by sampling it close to the object every frame; the object can also be rigidly rotated with respect to the lighting and vice versa.

This avoids aliasing and evaluates efficiently on graphics hardware by reducing the shading integral to a dot product of coefficient vector for diffuse receivers.

Authors further introduce functions for radiance transfer from a dynamic lighting environments through a preprocessed object to neighboring point in space. These allow soft shadows and caustics from rigidly moving object to be cast onto arbitrary, dynamic receivers.

Variations of SH basis:

Hemispherical Basis Functions The main bottleneck of spherical harmonics is that they describe spherical functions. On the other hand we want to describe BRDFs which are defined over a hemisphere. So when using SH, there is some data redundancy and discontinuities in the spherical domain at the boundary of the hemisphere. Few attempts were taken to overcome this problem of hemispherical function representation. Sloan *et al.* [SHHS03] proposed ESH (Even reflection SH) - padding of hemispherical function with a mirrored copy of the function itself. This significantly improves the accuracy, but on the other hand such coefficients doesn't represent original function and thus dot product with another SH coefficients yields erroneous result. Another technique for representing hemispherical functions from the same paper is LSOSH (Least Squares Optimal SH) - this also improves accuracy of the upper hemisphere and also suffers from the problem as above. Bespoke solution - *hemispherical basis* - which ensures a more accurate representation of hemispherical functions was introduced in

[GKPB04]. This basis can be combined with SH with minimal effort which allows it to be used in existing implementation instead of SH.

Zonal Spherical Harmonics Precomputed radiance transfer (PRT) using SH captures realistic lighting effects from distant, low-frequency environmental lighting but has been limited to static models or precomputed sequences. This work [SLS05] by Sloan *et al.* is focused on PRT for local effect suchs as bumps, wrinkles or other detailed features, but extend it to handle arbitrary geometric deformations.

Zonal harmonics (ZH) approximate spherical functions as sum of circularly symmetric Legendre polynomial around different axes (which resebmle lobes on the sphere pointing to some directions). More important, it can be trivially rotated (only the direction of lobe is rotated) whereas SH rotation is expensive and unsuited for dense per-vertex or per-pixel evaluation. This property allows, for the first time, PRT to be mapped onto deforming models which re-orient the local coordinate frame.

This nonlinear approximation of the transport by spatially varying both the axes and coefficients of ZH basis functions is fitted using a greedy method combined with local BFGS optimization step [PTVF92].

Although the ZH basis may yield a more compact representation than SH basis, it is still restricted to low-frequency signals and lighting environments, which has been shown [NRH03] to require large numbers of coefficients to represent all-frequency lighting content.

1.4.2 All-frequency Precomputed Radiance Transfer for Glossy Objects

Authors in [LSSS04] present a novel PRT formulation which factors glossy BRDFs into purely view-dependent and light-dependent parts, achieving reasonable accuracy with only $m = 10$ dimensional factors. They then tabulate an $m \times n_L$ transfer matrix at each surface vertex as a preprocess, representing the object's response to this lighting. Because this surface is so high dimensional, reducing m is crucial for making practical both the preprocessing and run-time. To compress the transfer matrices, authors divide the cubemap into 24 lighting segments and apply the Haar wavelet

transform in each segment to provide sensible quantization. Then CPCA is applied to each PRT segment to approximate it as a linear combination of a few representative transfer matrices within a small set of clusters over the surface. This exploits spatial coherence to compress very effectively. Most important, it maintains fast rendering rates with 2-3 orders of magnitude more lighting coefficients than previous methods, which increases accuracy and avoids temporal artifacts in high-frequency lighting environments.

source lighting is represented by a cube map at resolution $n_L = 6 \times 32 \times 32$.

Segmentation doesn't constrain the lighting in any way. It is merely a device to speed compression of the signal by an *a priori* division of it into parts that are likely to be coherent.

BRDF Factorization

BRDF is factored into the sum of products of m functions depending only on the light direction with m depending on the view direction only. Unlike past BRDF factorization methods, authors do account for shadows rather than assuming that source lighting arrives entirely unoccluded. This factoring yields PRT matrices with m rows that are specialized to the particular object's surface reflectance. With a small $m(= 10)$, they obtain accuracy that would require many more coefficients using unspecialized bases (such as SH).

The factoring of the BRDF functions $f(v, s)$ is initiated by forming a matrix Q whose components are $Q_{ij} = f(v_i, s_j)$, with n_v view samples, v_i, and n_s, light samples, s_j. Both types of directions are parametrized using parabolic map.

The singular value decomposition (SVD) is then performed on the matrix Q and set all but the largest m singular values to zero. Then

$$Q_{ij} \approx \sum_{k=1}^{m} G_{ik} \sigma_k F_{kj}$$

absorbing a square root of the diagonal matrix formed by the singular values σ_k into both left and right side factors, one obtain desired two functions

$G(v)$ and $F(s)$ via

$$f(v_i, s_j) \approx \sum_{k=1}^{m} G_k(v_i) F_k(s_j) = G(v_i) \cdot F(s_j)$$

1.4.3 Efficient Wavelet Rotation

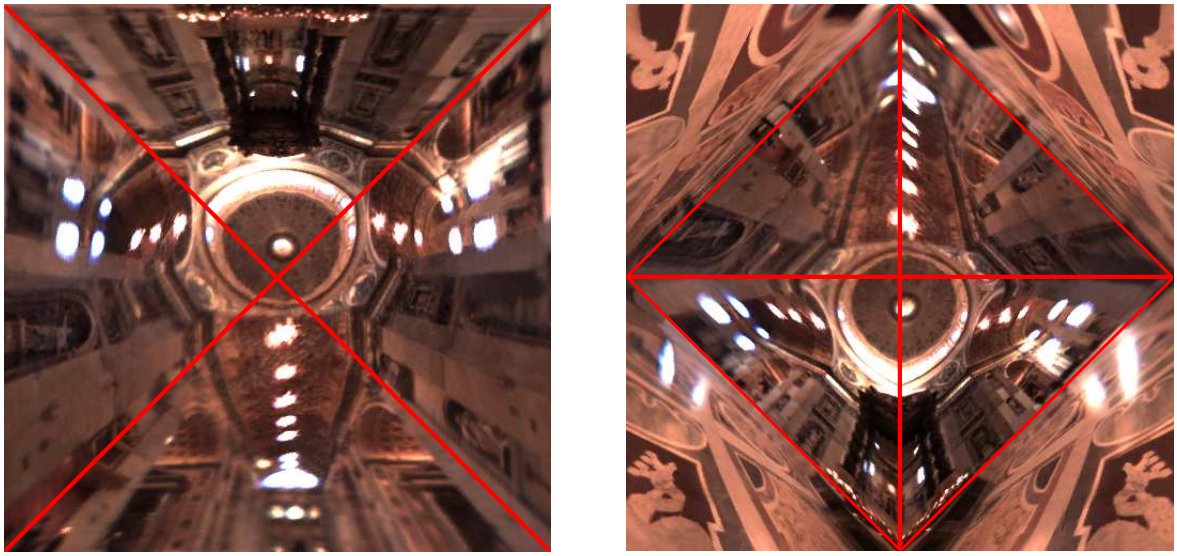
There was an extensive previous work on rotation of functions represented by spherical harmonics. There was and still is a will to develop efficient rotation of wavelets. Analytical solutions is in the time of writing this work still unknown. An efficient computational solution for wavelet rotation was introduced in [WNLH06]. This approach uses precomputed rotational matrices to perform the transformation (in this case rotation) of source wavelet coefficients into destination wavelet coefficients. Since wavelets functions used to compress (approximate) light maps have compact support, these matrices are very sparse, enabling thrifty storage and fast matrix-vector computation. Efficient wavelet rotation now eliminates the need of environment maps prefiltering and is thus faster and more appropriate to work with high frequency lighting and shadows. The viewpoint, model, lighting environment, and BRDFS can be modified at interactive speeds.

Wavelet rotation matrices are bound to used parametrizations of spherical and hemispherical functions. Although there is no restriction on these parametrization. Authors of the paper chose octahedral [PH03] and hemi-octahedral (*a.k.a* pyramidal) parametrization, because of their good L^2 stretch efficiency and both can be unfolded into a square map, which is convenient for later wavelet transform. We add one more advantage for the pyramidal map and that is straightforward efficient implementation on the GPU.

This technique use reflection equation in this form

$$B(\vec{n}, \omega_o) = \int_{\Omega(\vec{n})} \widetilde{L}(\vec{n}, \omega) f_r(\omega, \omega_o)(\omega \cdot y)\, d\omega$$

where B is the reflected light, integral over upper hemisphere at a local frame induced by normal $\vec{n}$. $\vec{n}$ is expressed in global frame, incident direc-



(a) Pyramidal map (b) Octahedral map

Figure 1.4: Spherical parametrizations

tion $\vec{\omega}$ and viewing direction ω_o are both expressed in local frame. Cosine term $(\omega \cdot y)$ is baked into BRDF.

Note, that this techniques doesn't integrate the visibility function discarding the possibility to handle self-shadowing.

In the beginning an on any change to the lighting environment the global lighting is sampled in octahedral parametrization and local lighting slice for every sampled normal is created using the matrix transformations. Similarly BRDF slices are evaluated for every viewing sample. Both local lighting and BRDF slices are in pyramidal parametrization of the same size and Haar wavelet transformed. They are stored as sparse vectors of resulting coefficients. In the rendering routine the rendering equation integral simplifies to dot product. Vertex normal is used to index the local lighting slice. Similarly global viewing direction is transformed into local frame of a vertex and this direction is used to sample BRDF slice. The dot product of those two sparse vectors determines the color of corresponding vertex. This is repeated for every vertex.

1.4.4 Triple Product Relighting (Integrals)

At each vertex we must compute an integral over the incident hemisphere of three factors: the lighting, visibility and BRDF, each one of them represented in form of wavelet coefficients. Previous method do not handle the visibility function.

Double products integrals of functions are simple to evaluate because they reduce to a dot product of coefficients approximating functions in question.

Assuming integral of three functions

$$L(\omega) = \sum_i L_i \Psi_i(\omega) \qquad V(\omega) = \sum_i V_i \Psi_i(\omega) \qquad \tilde{\rho}(\omega) = \sum_i \tilde{\rho}_i \Psi_i(\omega)$$

authors define *tripling coefficients* in analogy to coupling coefficients:

$$c_{ijk} = \int_{S^2} \psi_i(\omega)\psi_j(\omega)\tilde{\rho}_k(\omega)\, d\omega$$

Ng *et al.* presented and proved *Haar tripling coefficient theorem*, which characterizes the tripling coefficients for non-standard Haar basis. This theorem, to be brief, says that a majority of coefficients is zero (it even tells us the exact number). Triple product are considerably more complicated. Authors developed an efficient sublinear algorithm in Haar wavelets and they emphasize that the theorem and algorithm holds just for nonstandard Haar tripling coefficient and can't be used for other bases. Later in [MHL+06] another authors presented *tripling coefficients theorem for spherical wavelets*.

Reflection equation with direct lighting has to be evaluated

$$B(\vec{x}, \omega_o) = \int_{\Omega_{2\pi}} L(\vec{x}, \omega)\rho(\vec{x}, \omega, \omega_o)V(\vec{x}, \omega)(\omega \cdot \vec{n})\, d\omega$$

integral over the visible hemisphere indicated by normal $\vec{n}$. The cosine term $\omega \cdot \vec{n}$ is incorporated into the BRDF.

For rendering, proposed algorithm for evaluating triple coefficients is used.

In [MHL+06] authors proposed similar implementation using spherical wavelets with per-pixel computation running on GPU.

1.4.5 Nonlinear Gaussian Functions

Green *et al.* in [GKMD06] proposed hybrid PRT method for static scenes incorporating new representation of light transport operator $T_{\vec{x},\omega_o}$ based on sum of weighted isotropic Gaussians

$$\widetilde{T}_{\vec{x},\omega_o}(\omega_i) = \sum_{k}^{N} w_k G(\omega_i; \mu_k, \sigma_k)$$

where G is a 2D spherical Gaussian centered around μ_k with standard deviation σ_k and weight w_k.

The spherical Gaussians are parametrized by a RGB color, mean direction and variance. Even for accurate approximation they typically use a small number of Gaussians, between one and three, which leads to small storage cost.

The parameters preserve the qualitative shape of the kernel across interpolation which leads to a good visual reproductions of highlights. In particular, the nonlinear effect of the mean direction μ_k permits better interpolation and prevents cross-fading artifacts and the nonlinear effect of the variance σ_k parameter allows for direct encoding of scale, thereby affording arbitrary bandwidth (all-frequency effects).

These nonlinear parameters make a significant departure from the previous work based on nonlinear approximation [NRH03] that starts with linear basis and sets the small coefficients to zero. Key to sparse representation is that interpolation of the Gaussian parameters closely matches the behavior of integration kernels.

The visual quality of the former techniques depends heavily on the tessellation of the model because high frequency effects require a fine tessellation, otherwise cross-fading effects are visible. This method prevents this through nonlinear interpolation of transport functions instead of using linear blending.

Light transport data is evaluated for all views (92 fixed viewing directions obtained by icosahedron subdivision) and mesh vertices in precomputation. Computation of light transport data is not specific to this technique.

Authors have chosen the way of an optimization approach because it provides the flexibility to directly incorporate various energy coherence

terms (minimizing L^2 error with data, alignment of means, *etc.*) enforcing artifact-free interpolation of Gaussians parameters between neighboring views and mesh vertices.

However fitting the precomputed light transport data to this new representation requires solving a nonlinear regression problem, that is more involved than traditional linear and nonlinear (truncation) approximation.

Authors do not focus on view-independent (diffuse) component of light transport because it can be handled by well known techniques (*e.g.*. SH, wavelets).

Rendering involves integrating the lighting $L(\omega_i)$ against the the approximated and interpolated transport functions $\widetilde{T}_{\vec{x},\vec{v}}(\omega_i)$ for every visible point $\vec{x}$:

$$B(\vec{x},\omega_o) = \int_S \left(\sum_k^N w_k G(\omega_i;\mu_k,\sigma_k) \right) L(\omega_i) d\omega_i$$

This integral can be evaluated with small number of texture lookups to Gaussian-prefiltered mip-mapped environment maps (various locations and scales can be precomputed in such way, where each level corresponds to a Gaussian variance) [MLH02]. Per-pixel interpolation of the transport functions, which is critical to achieve high visual quality, is performed using barycentric interpolation of the Gaussian directions and variance parameters.

The closest PRT technique to this work was proposed by Sloan *et al.* [SLS05], but the main goal of their work is the ability to efficiently rotate the representation in order to handle local geometric deformations. Fundamentally they are limited to the same class of low frequency transport functions as spherical functions, which require large number of coefficients to represent all-frequency lighting as has been shown in [NRH03].

This technique requires prefiltering of the environment maps and thus is not currently suitable to support dynamic lighting. From our point of view the main disadvantage of this method for possible implementation in our work was its typical precomputation time – 60 computer-hours – which might be too difficult to implement and test due to our time/financial limitations of this master thesis.

Additionally, this method is currently restricted to model specular ef-

fects with Gaussian functions, and it is unclear, as the authors do not discuss the fact explicitly, whether other all-frequency effects, such as all-frequency shadows, could be handled, as pointed by Tsai *et al.* in [TS06] where they put their approach in contrast by the possibility of rendering objects with all-frequency shadows in their unified framework, which is indeed a more general approach of using *spherical radial basis functions* for approximating light transfer operator.

1.4.6 Clustered Tensor Approximation and Radial Basis Functions

in [TS06] new data representation and compression technique for PRT was introduces. The light transfer functions and light sources are modeled with spherical radial basis function (SRBFs). A SRBF is a rotation-invariant functions that depends on the geodesic distance between two points on the unit sphere. Rotating functions in SRBF representation is as straightforward as rotating the centers of SRBFs. Moreover, high frequency signals are handled by adjusting the bandwidth parameters of SRBFs. To exploit inter-vertex coherence, the light transfer functions are further classified iteratively into disjoint clusters, and tensor approximation is applied within each cluster. Compared with previous methods, the proposed approach enables real-time rendering with comparable quality under high-frequency lighting environments. The data storage is also more compact than previous all-frequency PRT algorithms.

Authors adopt for SRBFs for three reasons:

- the spatial localization property of SRBFs allows high frequency signals to be handled efficiently
- SRBFs are circularly axis-symmetric and rotation-invariant functions defined on the unit sphere. Radiance functions can be modeled in their intrinsic domain and it is also simple to rotate and convolute functions represented in SRBFs
- similar to radial basis functions (RBFs) SRBFs are naturally applicable to interpolate and extrapolate scattered data.

Spherical radial basis functions are special RBFs defined on the unit sphere. Their intrinsic nature in the spherical domain and other appealing properties, such as rotational invariance and positive definiteness, make them appropriate for modeling and analyzing spherical data without introducing any artificial boundaries or distortions. When combined with multi-resolution approaches, such as spherical wavelets [SS95], SRBFs become a powerful tool for analyzing scattered data on a sphere, including information measured by satellites and observed stations on entire globe.

Tensor approximation methods allow a higher compression ratio than the traditional PCA. CTA approach overcomes the major drawbacks of previous all-frequency PRT algorithms, and achieves real-time rendering performance without sacrificing much image quality. Experimental results reveal that SRBFs are effective in dealing with high-frequency signals, and provide an intrinsic representation for PRT data sets in spherical domain.

To investigate inter-vertex data coherence, the approximated results of diffuse and glossy objects are further compressed using *clustered principal component analysis* (CPCA) [SHHS03] and *clustered tensor approximation* (CTA), respectively. For glossy objects, the CTA algorithm classifies the light transfer function into groups to reduce inter-cluster variance. Since the PRT data sets within each cluster are intrinsically a multi-dimensional array, authors retain their original structure, and analyze the coherence along each dimension with tensor approximation. Additionally, a technique for iteratively updating cluster members is introduced to minimize least-squares errors.

1.4.7 Normal Mapping

This is not a standalone technique but merely an enhancement to other methods which select lighting data according vertex normal. Sloan [Slo06] introduced the possibility of use normal mapping along with PRT. The main advantage of normal mapping is the reduction of amount of 3D data by simulating the fine details by simple normal modification (more general technique than bump mapping). The information within bump maps is stored as greyscale image height map. On contrary, normal map is stored in three (R, G, B) channels with each channel holding one coordinate of

normal vector direction. Such vector is used to modulate normal vector with help of tangential space. Normal mapping in relation to PRT was only successful when simulating diffuse materials, lately also with more sophisticated methods that use local frame and use normal to index light precomputed data [MHL+06].

1.4.8 Per-vertex and per-pixel shading

In [MHL+06] the use of per-pixel shading and visibility textures for efficient GPU implementation was introduced. If rendering is performed on a per-vertex base, even if the flat surface typically requires fine tessellation to capture visibility variation Better solution is to pass the computation to the more powerful pixel shaders for noticeably faster GPU implementation. Authors sample the visibility function over a surface and store it in a visibility texture. After this the former fine tessellated mesh is replaced with a coarser one with additional visibility texture along with per-pixel shading. In order to convert per-vertex shading into per-pixel shading, the key is to store the visibility coefficients in texture.

For each texel p_t in the texture space it is necessary to find its corresponding point p_m in the model space and then use p_m and its local frame to sample the visibility. After applying projection of the visibility function into linear basis , store the coefficients back to p_t.

1.4.9 Prefiltered Environment Maps

Because of inefficiency of computing lighting integral at real-time speeds, this problem was usually managed by prefiltering environment maps offline (as it is rather expensive computation) to later simulate glossy reflections in real-time. Expensiveness of prefiltering excludes the possibility of dynamical change of light. Various techniques were developed to speed up prefiltering, but they are limited to specific BRDFs and thus not general and this is the feature we are looking for. Technique proposed by Green [Gre86] observed that an environment map prefiltered by a BRDF could be used to simulate diffuse and even glossy reflections. There has been a large amount of related research, but none of these methods handle shadowing

or indirect lighting.

1.4.10 Reflectance Prefiltering

Reflectance prefiltering was pioneered by Fournier [1992 - normal distribution functions and multiple surfaces. He uses nonlinear optimization to approximate distributions of microfacet normals at multiple resolutions using a sum of cosine lobes. Recently Tan [TLQ+05] proposed multiresolution reflectance filtering using Gaussian mixture model. They use well known *EM algorithm* to estimate parameters.

1.5 Appearance

The appearance of an object is the result of a complex interaction of the light incident on the object, the optical characteristics of the object, and human perception.

Appearance often determines the acceptability of a product to the end-user. The quality and consistency of the appearance of a product is psychologically related to its expected performance and useful life. It therefore determines its acceptance (or rejection) by potential purchasers.

The three elements of human perception – light, object and observer, can be quantified and further reduced to a set of colorimetric specifications. These colorimetric values can be converted into a variety of color and difference scales which can be used for numerical specifications.

Various methods of measurement are highlighted including the basic CIE optical geometrics for colorimeters and spectrophotometers, gloss meters, goniophotometers, gonio-spectrophotometers and image analysis.

All those methods require careful attention in selecting the specimen, measurement techniques, instrument calibration, standardization and verification and validation. This will provide means for more objective appearance communication thus minimizing disagreements and product returns.

There are active programs to standardize such observations and measurements, in the American Society for Testing and Materials (ASTM) and in a joint BAM and DIN Committee in Germany.

Appearance prediction and appearance engineering both rely on accurate physically based models of light interaction.

Metallic and pearlescent colors are rendered using three aspecular measurements defined in a proposed standard for goniochromatic color. Westlund *et al.* [WM01] try to apply appearance standards (gloss, haze) to light reflection models (Phong, Ward, Cook-Torrance). *Gloss* and *haze* are appearance attributes resulting from the first surface reflection. Gloss is an uncomplicated appearance attribute – an object is either glossy or matte. Gloss is a measure of the magnitude of the specular reflection and haze captures the width of the specular lobe.

Metallic paints and plastics have been widely used but relevant scientific concepts and terminology to describe the appearance of these materials

has not yet been established and are still evolving. By *metallic* we identify paints and plastics containing metal flake pigments rather than object made of metal which is a common understanding for the term.

The attributes of appearance of these materials are of two kinds, those observed at a distance of several meters and those observed at reading distance. The first kind may be called *macro appearance*, the second *micro appearance*.

Hunter in his seminal work on gloss measurement differentiated no less than six types of gloss:

- specular gloss
- sheen
- contrast gloss (or luster)
- absence-of-bloom gloss
- distinctness of image gloss
- surface-uniformity gloss.

As experts tried to find attributes which would describe appearance of object, two branches revealed: computer graphics researches have made an effort to develop the most general robust surface reflection model and to build sophisticated reflection measurement devices. This, as we know, led to introduction of various BRDFs with many driving parameters and for example spectrogoniophotometer as a device used to measure surface quality attributes.

On contrary, appearance industry professionals attempted to determine the minimum number of measurements and attributes to cover the largest possible set of appearance problems. This, in fact, caused the birth of one-dimensional scales of attributes (e.g. gloss, haze,...) and measurement devices such as glossmeter. This resulted in a set of appearance measurement standards. The use of a standard appearance scales instead of complex BRDF parameters, also provides a more intuitive way of selecting the reflection model parameters and a reflection model independent method of specifying appearance.

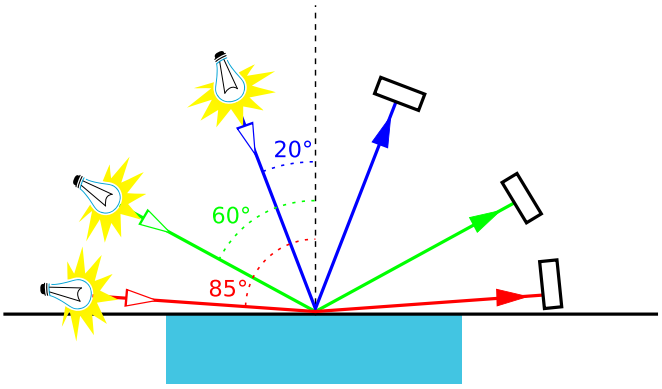


Figure 1.5: Glossmeter measurements at various angles

1.5.1 Virtual light meter

The measure of a gloss is a simplification of the BRDF to single appearance related attribute. A virtual light meter was constructed for the purpose of correspondence between BRDF model parameters and standard appearance measurements. In fact, it a customized 2D integration tool which uses numerical quadrature of particular BRDF over subdivided light source and detector apertures, which simulates the work of a real-life virtual light meter - the light source light hits the surface being measured and the detector catches the reflected light and evaluates the numerical result.

Virtual light meter can be used to evaluate know standards (specular gloss, haze, *etc.*) and can be customized for other measurements. In our work we implemented glossmeter. The measurement of specular gloss consists of comparing the luminous reflectance from tested specimen to that from a gloss standard, under some geometric and spectral conditions well defined by national or international standards (*e.g.* ISO 2813).

Those standards usually prescribe the measurements to be taken at angles 20°, 60°and 85°as illustrated at Figure 1.5,because these degrees of specular gloss measurements offer numerical values which are roughly linearly correlated over a range of values to perceived gloss of high-gloss, medium-gloss, and low-gloss surfaces respectively. The numerical gloss value, assigned to a surface typically ranges from 0 (low gloss) to 100 (high gloss).

1.6 Color & Light Representation

Most CG computation are computed with RGB values, as this is sufficient for most purposes. Spectrum-based rendering improves the tristimulus representation (RGB, XYZ,...) by incorporating spectral power distributions allowing more advanced visual effects (*e.g.* thin film effect). In [DK05] a framework for spectrum-based rendering on GPU was introduced. It was demonstrated on Phong model extended with spectral information, and also the ability to hand multilayered thin film interference effects. In both cases scene was lit by an area light obtained through spectral cube map. Authors proposed method for conversion of regular tristimulus environment map into a spectral one.

As we were searching in area of spectrum-based rendering we found an interesting project `http://www.imageval.com` providing high dynamic range spectral (HDRS) image database, which offers complete spectral approximations of the light radiance field. We would like to use this, or produce similar data on our own in our future work.

[Pee93] proposed general linear method for handling full spectral information by expressing it in a linear basis so it can speed computations and lower the cost storages.

1.6.1 High dynamic range imaging

High dynamic range imaging is an emerging field borrowing ideas from several fields that study light and color. The intention of HDRI is to accurately represent the wide range of intensity levels found in real scenes ranging from direct sunlight to the deepest shadows. The range of values afforded by a conventional image is about two orders of magnitude, stored as a byte for each of the RGB channels per pixel. The visual quality of high dynamic range images is vastly higher than conventional low-dynamic-range images.

It is not possible to directly print or display images with a much higher dynamic range. Indeed there is possibility to display HDR images on new HDR display devices. But these are more involved and expensive and we rather limit ourselves to conventional low dynamic range display devices.

Thus, to simulate the effect of reducing an HDR image to within a displayable range, we reduce a conventional photograph in dynamic range to well below two orders of magnitude.

One problem with HDR has always been in viewing the images. CRTs, LCDs, prints, and other methods of displaying images only have a limited dynamic range. Thus various methods of "converting" HDR images into a viewable format have been developed, generally called "tone mapping".

Early methods of tone mapping were simple. They simply showed a "window" of the entire dynamic range, clipping to set minimum and maximum values. However, more recent methods have attempted to show more of the dynamic range. The more complex methods tap into research on how the human eye and visual cortex perceive a scene, trying to show the whole dynamic range while retaining realistic colour and contrast.

Many works use HDR image as a source of lighting because the image than carries *radiance i.e.* radiometric quantity denoting the flux (power) incident, passing through or leaving a unit surface area dA from a unit set of direction$[Wm^{-2}sr^{-1}]$.

For instance, the combination of shutter, lens, and sensor in digital camera restricts incoming light in this fashion. When a picture is taken, the shutter is opened for a small amount of time (exposure). During that time, light is focused through a lens that limits the number of directions from which is light received. The image sensor is partitioned into small cells, so that each cell records light over a small area. Because camera records radiance, it is therefore possible to relate the voltage extracted from the camera sensor to radiance, provided pixels are neither under- nor overexposed. To support spectral based rendering the radiance may be defined per unit wavelength interval, which is then rreferredas spectral radiance.

1.7 Streaming SIMD processing on CPU & GPU

Scalar processing - one operation produces one result *SIMD processing* - one operation (with multiple inputs) produces multiple results

Streams are collections of records requiring similar computation (vertex position, pixel RGBA quadruples, *etc.*). Records or elements of the stream

shall have a few or better no dependencies between each other in order to exploit data parallelism.

Streaming computations can be characterized as being highly parallel and computationally intensive with little reuse of input data. However, many computations are both parallel and computationally intensive, but exhibit significant per element use.

Kernels are functions which are applied to each element in the stream (transform vertex position, modify pixel color, *etc.*) A streaming processor executes a kernel over all elements of an input stream, placing the results into output stream.

Streams are categorized into four types:

Input streams - that contain read-only data for kernel processing.

Output streams - store the result of the kernel computation.

Gather streams - permit arbitrary indexing to retrieve stream elements. They are read-only.

While kernel provide a mechanism for applying a function to a set of data (streams), reduction provide a data-parallel method for calculating a smaller stream or even to a single value from a set of records. Examples of reduction operations include arithmetic sum, computing maximum, SAXPY. In order to perform the reduction in parallel, we require the reduction operation to be associative (*i.e.* $a \circ (b \circ c) = (a \circ b) \circ c$). This allows the system to evaluate the reduction in whichever order is best suited for the underlying architecture and/or algorithm.

The notion of *arithmetic*, or more general *computational intensity* is a ratio between time spent in kernel computation K on one stream element and the amount of time spent on transferring one stream element (*i.e.* $I = \frac{K}{T_{transfer}}$). The higher the computational intensity I is (significant amount of computation relative to the short time spent transferring data), the better suited it is for stream computation.

SIMD (Single Instruction Multiple Data) is the primary performance feature on most platforms.

1.7.1 CPU

SSE (Streaming SIMD Extensions) is a powerful and versatile implementation of SIMD on CPU.

Authors of [MY02] discuss the possibility of using SSE for 3D geometry processing. SSE2 data types can hold anything that fits to 128 bits (*e.g.* 4 floats, 8 word, 16 bytes). SSE/SSE2 loads/store expect data aligned on 16-byte boundary; otherwise crash! There are unaligned load/store versions, but these are significantly slower.

Another important feature of the SSE is the memory streaming instruction extensions, which allow programmers to prefetch data into a specified level of the cache hierarchy. Most multimedia applications present the streaming data access pattern; that is, data are accessed sequentially and seldom reused. Therefore, prefetching this type of data into the L2 cache is an effective way to improve the memory system performance.

Memory layouts:

Array of Structures (AoS) - mostly used layout for example in images (RGBA), but, unfortunately, defeats SIMD efficiency.

Structure of Arrays (SoA) - provides maximum parallelism

Hybrid structure - also SSE and memory friendly

We find SSE restriction to fixed maximum vector length (4), assumption of AoS memory layout and alignment too limiting and we avoid using it in our work.

1.7.2 GPU

The GPU on commodity video cards has evolved into an extremely flexible and powerful processor. The emergence of programmable graphics hardware has led to increasing interest in offloading numerically intensive computations to GPUs. The combination of high-bandwidth memories and hardware that performs floating point arithmetic at significantly higher rates than conventional CPUs makes graphics processors attractive target for highly parallel numerical workloads.

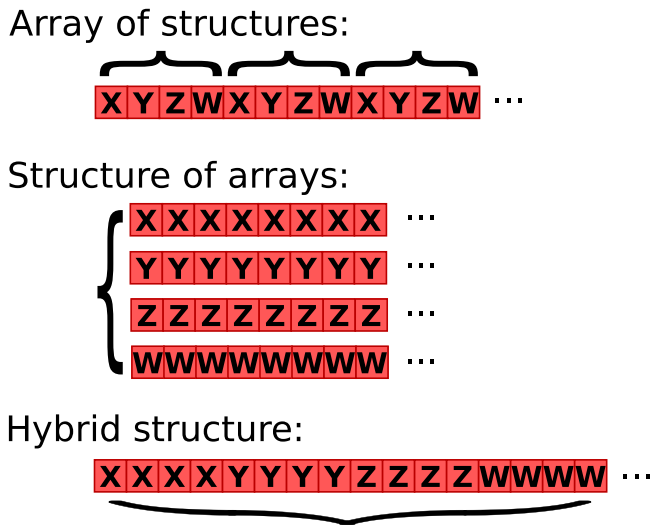


Figure 1.6: Memory layouts (braces denote span of a structure)

Vertex processors are responsible for primitive construction, that is convert input vertices into primitives (of specified type) with area (*e.g.*. lines, triangles, quadrilaterals). Each vertex may have some standard attributes associated with it (like color, normal, texture coordinates, *etc.*) or new user specified (for example temperature).

Vertex processor can be seen as a programmable (SIMD or in nowadays even MIMD) which can process 4-vectors (RGBA) It is capable of scatter operation as it may change the resulting window position of vertex processed, but is not able to read data from any other vertices and thus not support gather operations (recently made available to GPGPU applications using ATI/AMD CTM and NVIDIA CUDA vertex texture fetch).

Fragment processor is responsible for rasterization, that is conversion of geometric primitives supplied by vertex processor to image primitives consisting of pixels or more generally fragments (pixel + associated data: color, depth, stencil, etc.) In the process of rasterization per-vertex attributes are interpolated across pixels within the primitive being rasterized (it is mostly useful for interpolating addresses - texture coordinates)

Fragment Processor can be also seen programmable (SIMD/MIMD) processor which likewise processes 4-vectors, contrary to vertex processor, it is able to gather information (access data for reading in textures on any position) texture fetch, while output address is fixed to a specified fragment. It

is generally more useful than vertex processor because it has more pipelines than in vertex processor and it is at the end of a processing pipeline.

Usually we can look at texture memory as a read-only memory interface optimized for coherent 2D access, or when using extensions enabling rendering to texture, it can be seen as write-only memory interface.

Many applications beyond traditional graphics have been demonstrated to run on GPUs. Particularly, algorithms that can be structured as streaming operations often realize notable performance gains. The combination of regular, predictable data access with the independence of each kernel invocation maps very nicely to graphics architectures.

The *map operation* has simple logic to iterate over the elements of a given stream meanwhile applying kernel function $f(x)$ to element of input stream and storing the result in an output stream. function $f(x)$ and input stream $S_i(e_0, e_1, \ldots, e_n)$ $S_o(f(e_0), f(e_1), \ldots f(e_n))$

GPU implementation is pretty straightforward: S_i is a texture, e are texels, fragment shader F implements $f(x)$. We then draw the rectangle with as many pixels as texels in S_i with fragment shader F active. Rendered (= computed) output is then stored in another texture.

High level shading languages for programming GPU are Cg, HLSL, OpenGL Shading Language. They resemble C-like languages, but are enhanced with vector data types, *etc.*.

The problem with general purpose programming on the GPUs is their difficult usage caused by their original purpose design – for real-time rasterization of geometric primitives in CG and game industry. In relation to this their programming interfaces provide programmers with unusual model and idioms tied exactly to CG. Up to few months ago there was no possibility to simply port CPU code GPU without completely rewriting it to the constrained environment provided by shading languages.

High-level languages for GPGPU usually extend base language *e.g.* C++ to support stream programming by including various simple data-parallel constructs (*e.g.*. float[2,3,4], input/output streams), meanwhile they hide details on graphics API and often the same source code can be compiled to run on CPU or GPU, or those backend may be chosen even at runtime. This transparency and new data types allows programmers to focus on the algorithm and do not force them to solve low-level implementation problems.

To mention such languages:

- Accelerator (Microsoft Research, http://research.microsoft.com/research/downloads/)
- CTM (ATI/AMD http://ati.amd.com/companyinfo/researcher/documents.html)
- CUDA (NVIDIA http://www.NVIDIA.com/object/cuda.html)
- Peakstream (http://www.peakstreaminc.com)
- RapidMind (Commercial follow-on to Sh)
- Sh (Stanford `http://www.libsh.org`)
- Brook for GPUs (`http://graphics.stanford.edu/projects/brookgpu`)

Lefohn *et al.* in [Lef06] presents Glift, an abstraction and generic template library for defining complex, parallel random-access data structures (stack, quadtree, and octree, *etc.*) on GPUs.

Performance tuning of algorithms on graphics hardware is difficult because vendors do not disclose specific architecture details, such as cache parameters or the physical layout of texture data in memory.

An experiment which explored the cache architecture of GPUs was done in [GLGM06].

1.8 OpenMP

The goal of OpenMP is to provide a standard and portable API for writing shared memory parallel programs. It works in conjunction with C/C++ (and also Fortran) and is comprised of a set of compiler directives (to be more specific - #pragma directives - instructional notes to a compiler placed directly in code) that describe the parallelism in the source code, along with a supporting library of subroutines available to applications. Pragmas have special benefit that if the compiler does not recognize particular pragma, the directive is simply ignored and in case of OpenMP directives the code in question would be compiled normally – as a single-threaded operation.

This approach allows the same code base to be used for development on both single-processor and multi-processor platforms. Incremental approach

- starting from a sequential program, the programmer can embellish the same existing program with directives that express parallel execution.

An OpenMP program always begins with a single initial thread (called *master thread*) of control that has an execution context (data environment) associated with it. The master thread an its execution context exist for the duration of the entire program. When the master thread encounters a parallel construct, new threads of execution are created along with an execution context for each thread. Each thread has its own stack within its execution context.

Parallel construct is a `#pragma` directive of form:

```
#pragma omp parallel [clause[ [, ]clause] ...] new-line
structured-block
```

Here is an example of OpenMP `parallel for` work-sharing construct that, as its name suggests, parallelizes the for loop:

```
int i;


#pragma omp parallel for
for ( i = 0; i < ARRAY_SIZE; i++);
    array[i] += i;
```

The pragma, located just before the outer-most for-loop, tells the compiler to parallelize the loop. In case the compiler does support OpenMP, it generates a lot of invisible threading code, *e.g.* it will insert code that determines the best number of threads to use on the execution platform and then break up the loop across that number of threads. We develop our software on a dual-core system, so it would create two threads, each one processing its half of loop sequentially.

At the end of the for loop (which forms an implicit barrier), multiple threads join and the code returns to single threaded until, for exemple, another OpenMP pragma is encountered.

From other commonly used features of OpenMP we will mention the ability of parallel constructs to choose either to share a single copy between all the threads or to provide thread with its own private copy for the duration of the parallel construct and may vary from one parallel construct

to another. It is specified via scoping clauses for individual variables. A variable may have one of three basic attributes: *shared*, *private* or *reduction* (variable is a target for arithmetic reduction).

Languages with single program multiple data (SPMD) constructs, work-sharing constructs, and synchronization constructs, and they provide support for the sharing and privatization of data.

Covers only user-directed parallelization, wherein the user explicitly specifies the actions to be taken by the compiler and runtime system in order to execute the program in parallel.

Notice that this pragma has removed a lot of overhead and housekeeping, such as: The pragma approach of OpenMP simplifies the creation of multi-thread applications by allowing user to concentrate on solving a real problem rather than to fighting the annoying explicit function-based multi-thread libraries (like POSIX pthreads, *etc.*).

But as in the case conventional multi-thread libraries the OpenMP-compliant implementations are not required to check for dependencies, conflicts, deadlocks, race conditions, or other problems that may appear in the realm of multi-thread programming. The user is the one responsible for using OpenMP in his applications to produce a conforming program. Additionally OpenMP does not cover compiler-generated automatic parallelization (also know as vectorization).

However, it is undeniable that all the work OpenMP executes on your behalf is done in the background. This fact makes debugging OpenMP applications somewhat more difficult.

Chapter 2

Design & Implementation

2.1 Short Architecture Review

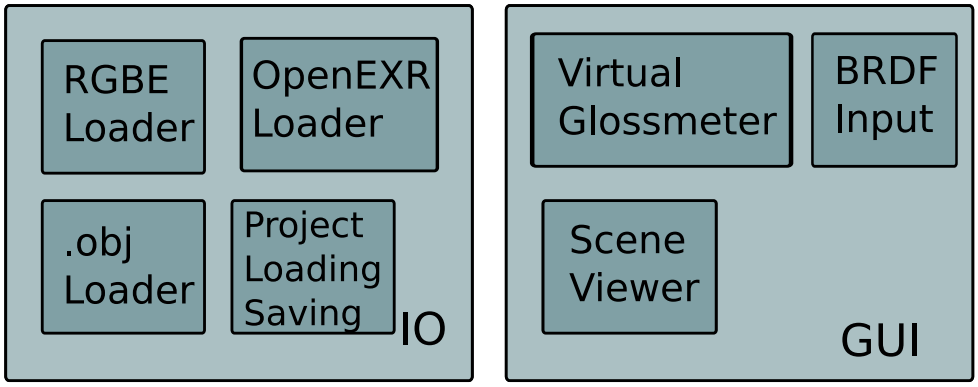


Figure 2.1: Simple block diagram of software components

Here follows a simple description of Figure 2.1:

3D Model Loader is the part responsible for loading 3D models for further processing and viewing.

Project Loading/Saving is part of the IO, that is responsible for loading and saving user data.

GUI Graphical user interface that allows user to load/remove 3D models and/or lightings and it is also comprised of following subparts:

Scene Viewer is a simple 3D scene viewer allowing us to do basic rigid operations to scene - rotating, zooming of the whole scene and rotating the 3D model and lighting separately.

BRDF Input provides use with the one- or two-lobe version of Cook-Torrance model.

Virtual Glossmeter simulates the behavior of a real-world glossmeter.

2.2 Beginning

In the early stage of work, after we finished compendium on the related works, we began implementing low-level data structures. The core consists of templated structures for 2-, 3- and 4-vector which provide the standard behaviour of these mathematical entities (scalar multiplication, dot product, *etc.*). Another math elements are 3- and 4- matrices, which are also implemented as templated structures with interface allowing common matrix operations (matrix-matrix multiplication, matrix-vector multiplication, inverse matrix, and so on). In our code we use float version of these structures.

The most CG related data structure is obviously a 2D image buffer. We are aware, that there are plenty of such implementations available, but they usually offer up to 4 channels (RGBA). We aimed to develop image buffers, which could hold image with arbitrary number (set at compilation time at least) of channels. Such request resulted from one of our possible goal - augment current PRT techniques to support spectrum-based effects. Such channels could hold coefficient representing approximation of the spectrum in particular chosen basis as discussed in Section 1.6 on page 29. They are implemented as templated arrays parametrized over element type and number or channels per pixel. Indeed we implemented various buffers with slightly different interface (1D or 2D indexing of elements, *etc.*)

As a starting exercise we implemented SH with fixed number of 36 coefficients, with dot product computation running on the GPU. We are able to display model with 120k vertices in about 20 FPS (this program can

be found in tests directory under the name `gpusphericalharmonics`. This method offers only simulation of diffuse reflections. Our shader performs dot product in the for cycle across 1D vectors, which is not cache efficient. It would be probably more efficient to implement it as a parallel reduction over 2D blocks (of size $6 \times 6 = 36$).

We use vertex buffer object (VBO) to store vertex data. VBO is an OpenGL extension intended to enhance the performance of OpenGL by providing the benefits of vertex array and display list, while avoiding downsides of their implementations. VBOs allows vertex array data to be stored in high-performance graphics memory on the server side and promotes efficient data transfer. VBOs provide control over the mappings and unmappings of buffer objects and define the usage type of the buffers. This allows graphics driver to optimize internal memory management and choose the best type of memory – such as cached/uncached system memory or graphics memory – in which to store the buffers.

After this little exercise we started implementing method described in the paper *All-Frequency Precomputed Radiance Transfer for Glossy Objects* [LSSS04]. We implemented SVD, CPCA, parabolic parametrization and wavelet transforms, we encountered the paper *Efficient Wavelet Rotation for Environment Map Rendering*, which allows dynamic change of lighting, BRDF and viewing direction. This freedom does matter more for us than the ability to have realistic GI data baked (with significatly slower precomputation).

As the method is pretty straightforward, we will recall it in coarse details. We implemented pyramidal and octahedral parametrization of hemisphere and sphere, respectively. Example images can be seen in Figure 1.4 on page 18. We developed functions to enumerate i^{th} nonstandard 2D Haar basis functions in particular order. This function also fetch the basis function to allocated memory. Rotation matrix for each sampled normal is then created by enumerating basis functions, wich are reparametrized from octahedral to pyramidal map to compute the resulting elements of the transformation matrix.

We store these matrices in sparse format. Sparse vectors are simply implemented as a composition of two vectors – one vector of float values and one vector of integers to hold positions. Sparse matrix is stored in

N	M	NO	Computation time	Size of zlib file (MB)	Load time (sec)
256	64	4096	9s	2.1	0.5
1024	256	1024	35s	13	2.6
1024	256	4096	2m 30s	51	9
4096	1024	1024	9m 30s	74	12
4096	1024	4096	37m	294	76

Table 2.1: Statistics of rotational matrices computation. N is the number of pixels of global lighting map, of the local lighting slice. NO is the number of sampled normal. Computations were performed on double-core CPU

compressed row format as vector of sparse rows (vectors).

Lighting and BRDF data are converted to sparse structures performing the nonlinear compression by *index sorting* of coefficients magnitudes and only few with the highest magnitude and their corresponding positions in original dense vector are stored. We also experimented replacing sorting by *quick select* (also known as k^{th} select) which should be asymptotically better than sorting, but this was not true as there are only few non-zero coefficients and the algorithms tries to sort the nearly zero coefficients.

In the rendering routine, we proceed with each vertex: we determine its visibility to the observer by its normal. If it is visible, we use the normal to index local lighting slice. With help of vertex tangent vectors, we transform global viewing direction into the local one. We use this direction to sample the BRDF slice. Then we perform three dot products of sparse vectors for each RGB channel and store the resulting color into local buffer. Once every vertex is processed we send the buffer with colors to GPU to update the actual colors of VBO holding the vertex data on the graphics card.

In our work we do not use quantization neither in CPU nor GPU computation in contrary to original work.

Some statistics about precomputation of rotation matrices is shown in Figure 2.1.

We use OpenMP for parallelizing loops in several places of our code -

name of model	# of vertices	FPS
Armadillo	34k	10
Bunny	36k	10
Suzanne	32k	11
Suzanne	126k	2
icosphere	15k	24
icosphere	61k	6

Table 2.2: Performance results of our WENV implementation

in computation of rotational matrices, processing (sampling, wavelet transforming and storing slices of) global environmental lighting and BRDFs.

Listing below shows a skeleton of the computation of rotational matrices.

```
float* ndata = 0;

#pragma omp parallel private(ndata) shared(totalnNNZ)
{
ndata = new float[WENV_n*WENV_n];
...
#pragma omp for
for (int normIndex = 0; normIndex < WENV_NO; ++normIndex) {
...
#pragma omp critical
{
        ++totalNNZ;
}
...
}
delete [] ndata;
...
}
```

We specify the `ndata` variable to be private for every spawned thread, ssubsequentlyallocating memory for it in the parallel block so every thread will have its private chunk of memory for `ndata`. Likewise we delete this memory before the end of the parallel block so there are no memory leaks. The

critical block int the sample above is what it's name suggests – it is an critical section where only one thread at a time may pass through. In this case, threads are explicitly synchronized at this point and they correctly increment the number of non-zero coefficients `totalnNNZ` for our statistics. For this purpose `totalnNNZ` has to be declared as *shared* among the threads. Indeed, we can conclude that the speed has doubled (for our double core implementation) comparing to single threaded implementation.

On the other hand, in display routine we use OpenMP construct with dynamic scheduling, because the computing of vertex colors depends on visibility condition – if the vertex belongs to some back-facing triangle, it is discarded (set to black color), otherwise the dot product of BRDF and light sparse vectors is carried out. This exploits uneven distribution of time needed to compute colors of vertices. If we would use simple `#pragma omp for` directive as above, one thread would compute color for first half of mesh vertices, second thread for the rest of vertices. It could happen, that vertices processed by first thread would be invisible to the observer and their computed instantly, but still we would have to wait for the second thread to process it's data. Luckily for us, OpenMP provides directive clause to enable dynamic scheduling – if specified, the iterations are guided assigned to threads in chunks as the threads request them. The thread executes the chunk of iterations, then requests another chunk, until no chunks remain to be assigned. In the following listing the chunk size is set to 100:

```
#pragma omp parallel for schedule(dynamic, 100)
for (int i = 0; i < currentModel_->numOfVertices(); ++i) {
...
}
```

In this case the speed up was significant, but not clearly doubled as in precomputation of rotational matrices, obviously due to scheduling overhead.

The *noniterative* work-sharing `sections` construct contains a set of structured blocks (`section`) that will be divided among, and executed by, the threads in a team. Each structured block is executed only once by one of the threads in the team.

```
#pragma omp parallel sections
{
```

```
#pragma omp section
{
// process R channel data
}
...
// same for G and B channel
...
}
```

We do not use SSE2 CPU capability because of the restrictions discussed in 1.7.1 on page 32. Instead we bet on new *autovectorization* enhancement of g++ compiler. The Vectorizer is enabled by the flag `-ftree-vectorize`. Information on which loops were or were not vectorized and why, can be obtained using the flag `-ftree-vectorizer-verbose`. For our disappointment, only very few loops had been vecorized and indeed they were no that crucial for the overall speedup.

2.3 GPGPU Nonstandard Haar wavelet transform

The need of storing rotation matrices in the memory for all the time was bothering us from the first time. Once the lighting changes it is transformed to local slices induced by normal directions. These slices are then indexed and dotted with BRDF slices in rendering routine. We came with an idea to bypass the transforming the global light to many local slices, dropping the memory consuming transformation matrices, and sample and wavelet transform slices on the GPU. We usually use the setup of 32×32 normal samples (in octahedral parametrization) and the size of local slice also 32×32. Easy calculation can prove, that if we tiled local slices side by side, row by row, we would end up with tileboard of size 1024×1024 wich fits int the GPU memory.

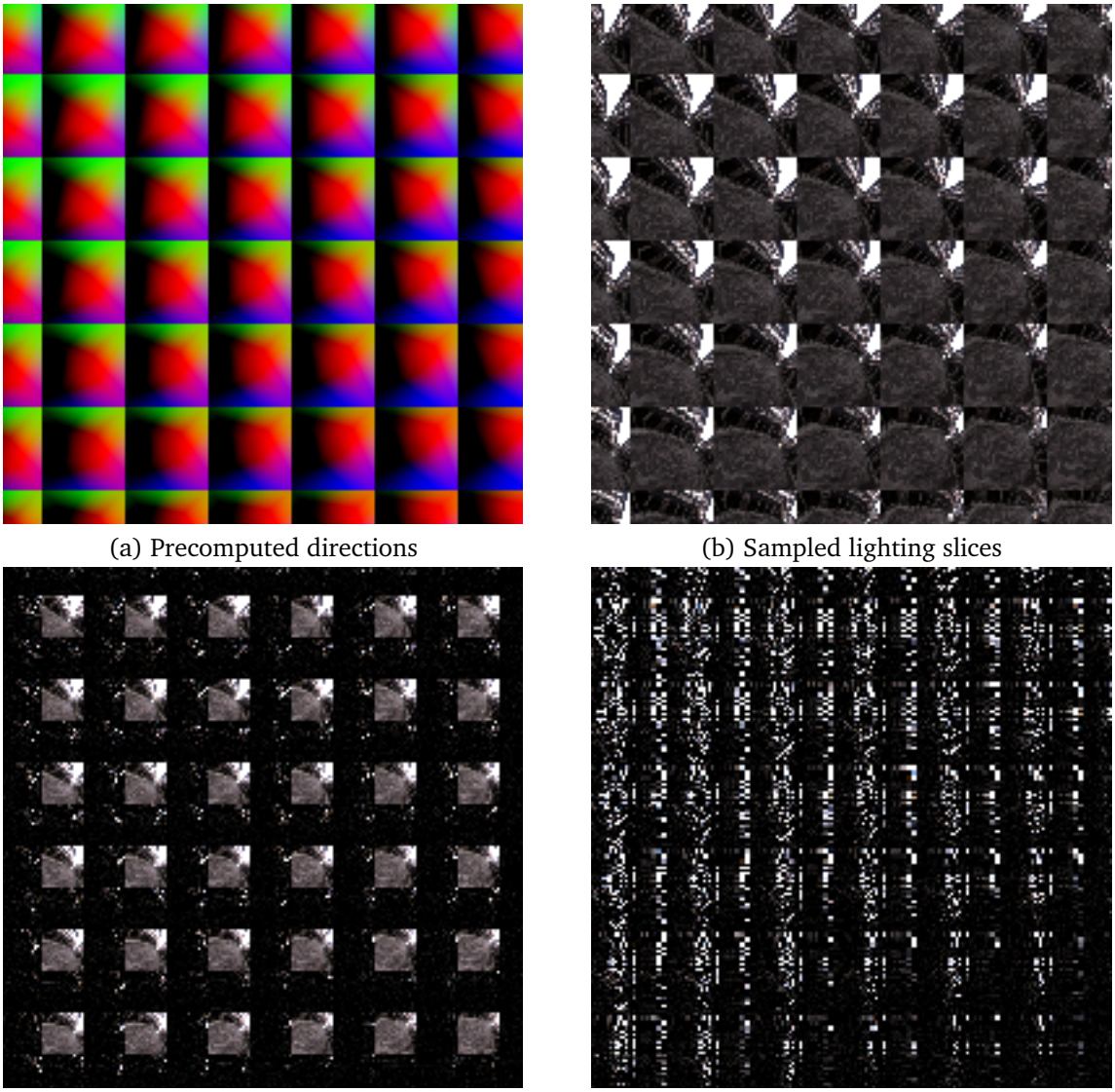
For each tile = slice induced by normal we first sample the lighting in the direction of this normal. Those HDR image data are stored on graphics card until the whole Haar transform is performed and only after that those data are send to system memory, where there are transformed to sparse

vectors as in purely CPU version. Haar transform is performed in the fragment shader by horizontal and vertical steps on all the tiles.

For this we use the well known *ping pong* technique of two image buffers (textures) - we use first texture as a source for the shader. Assuming horizontal step we execute shader for this with parameters indicating level of the transform (*i.e.* what data to transform and what data just to pass unchanged). We render the results in the second texture using framebuffer extension. After this the role of the texture is swapped. We use the second texture as input for the first and perform vertical step, and so on. For better understanding you may have a look at Figure 2.2

We encountered slow-down when using shader with if nested in another if statement to determine what operation (averaging, difference of pass) has to be performed for pixel in question by fragment shader, so we rather precomputed these conditions into RGB texture as can be seen in Figure 2.3 - for every pixel we now compute all branches, but the branch which will be eventually stored as a result is determined by the value of corresponding pixel in corresponding precomputed texture (level of the transform). This technique of getting rid off branching (or marginal states) is called *domain decomposition*. This helped us to speed up dynamic change of lighting by factor of 2-3.

We implemented test for dot product of dense vectors on the GPU, but this would be slower including the transfers between GPU and CPU. Dot product of two sparse vector is not well parallelizable, so there would be no speed up of such computation on GPU. The case dot product computation with dense vector stored in graphics hardware memory



(a) Precomputed directions

(b) Sampled lighting slices

(c) First iteration of Haar transform

(d) Corresponding transformed lighting slices

Figure 2.2: Visualization of GPU Haar transformation tileboards (cropped)

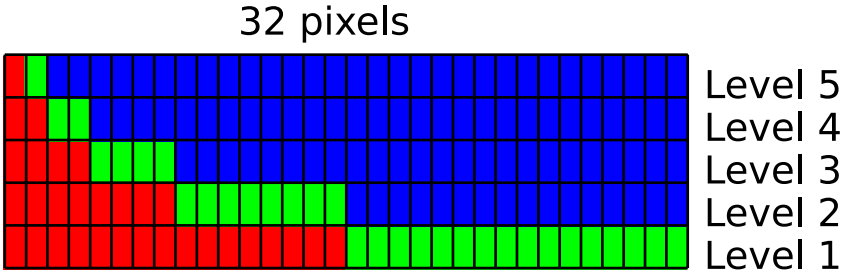


Figure 2.3: Domain decomposition for GPU Haar transform (red denotes positions where low-pass filter results are stored, green high-pass and blue identical values

2.4 BRDF

We implemented multi-lobe form of the physically derived Cook-Torrance model [CT81] that has shown to perform well with many materials. Here follows the BRDF definition

$$f_r(\omega_i, \omega_o) = \frac{K_d}{\pi} + \sum_{i=1}^{N} \frac{K_{si} F_{r_i} D_{m_i} G}{\pi(\vec{n} \cdot \omega_i)(\vec{n} \cdot \omega_o)}$$

where N is the number of specular lobes, Kd and K_{si} are the diffuse and specular parameters, respectively, and m and r are the parameters for the distribution and Fresnel terms, respectively. We use Beckmann distribution function for roughness D of the surface.

Although for purpose of this thesis there is only possibility to choose between one- and two-lobe model from our graphic user interface.

2.5 Virtual Glossmeter

Specular gloss is widely used in paint, paper, plastic and textile industries for the characterization of mirror-like appearance of a surface by specific reflectance measurements. We have choosen wavelet environment mapping for visualization, because it preserves high frequencies in view-dependent specular highlights.

Those highlights are rather glossy than mirror-like, but this is sufficient for a large spectrum of materials. Because of this we are interested in the numerical gloss value of 3D model being displayed as a feedback.

To implement virtual glossmeter we need a 2D integration tool to compute the equation for the total exitant flux $\Theta_{S,D}(\rho)$ passing through the detector aperture

$$\Theta_{S,D}(\rho) = AL_S \sum_{k=1}^{K} \sum_{j=1}^{J} \rho(\vec{s}_j, \vec{d}_k)(\vec{s}_j \cdot \vec{n})(\vec{d}_k \cdot \vec{n})$$

for derivation, please, have a look at original paper [WM01]. Westlund *et al.* used the Cubpack++ library to compute cubature in their work. In our work we have chosen the library *Cuba*.

The Cuba library offers a choice of four independent routines for multidimensional numerical integration: Vegas, Suave, Divonne, and Cuhre. They provide methods for deterministic/Monte Carlo integration with different strategies of variance reduction. Cuba has not such large support for predetermined primitives as Cubpack++ (*e.g.* rectangle, plane, strip, *etc.*) and instead is designed to compute only Riemann integrals on hypercube

$$If := \int_0^1 d^d x f(\vec{x})$$

where it is assumed that $f(\vec{x})$ is given as a function or subroutine that can be sampled at arbitrary points $x_i \in [0,1]^d$.

Though this is not a serious restriction since most integrands can be easily transformed to the unit hypercube:

$$\int_{a_1}^{b_1} \cdots \int_{a_d}^{b_d} = \int_0^1 d^d x f(\vec{x}) \prod_{i=1}^{d} (b_i - a_i)$$

where $x_i = a_i + (b_i - a_i)y_i$.

With help of Cuba library we integrate over the rectangle aperture of the detector so that we set the fixed direction from the point on detector (in global frame) and then we perform nested integration over the rectangle aperture of a light source using the stored direction from the detector and directions to the light source rectangle. Cuba also takes care of adaptive subdivision during the integration. We convert RGB triplet to single luminance value applying coefficients $(0.2126, 0.7152, 0.0722)$.

In order to compute gloss value, we need to compute and compare the

total exitant flux of measured surface and total exitant flux of a standard surface and divide them

$$G = 100 \frac{\sum_{k=1}^{K} \sum_{j=1}^{J} \rho(\vec{s}_j, \vec{d}_k)\ d\Omega_{S_j} d\Omega_{D_k}}{\sum_{k=1}^{K} \sum_{j=1}^{J} F(\vec{n}, \vec{s}_j) \delta(mirror(\vec{s}_j) - \vec{d}_k)\ d\Omega_{S_j}}$$

The numerical gloss value, G, is measured in so called *gloss units* (GLU) and typically ranges from 0 (low gloss) to 100 (high gloss).. The evaluation time of our virtual glossmeter takes from fractions of a second to few second, depending on the measurement angle and BRDF parameters.

By definition, the ultimate standard surface for gloss measurements is a smooth black glass surface having a refractive index of 1.567 for the wavelength 589.26 nm for all angles. For all angles of incidence this surface by definition has an assigned gloss value of 100 GLU. The blackness limits the exitant flux to first standard reflection and smoothness ensures presence of reflections only in specular direction. Corresponding specular reflectance values at 20°, 60°and 85°computed using the Fresnel equations are approximately 0.049, 0.998, 0.619.

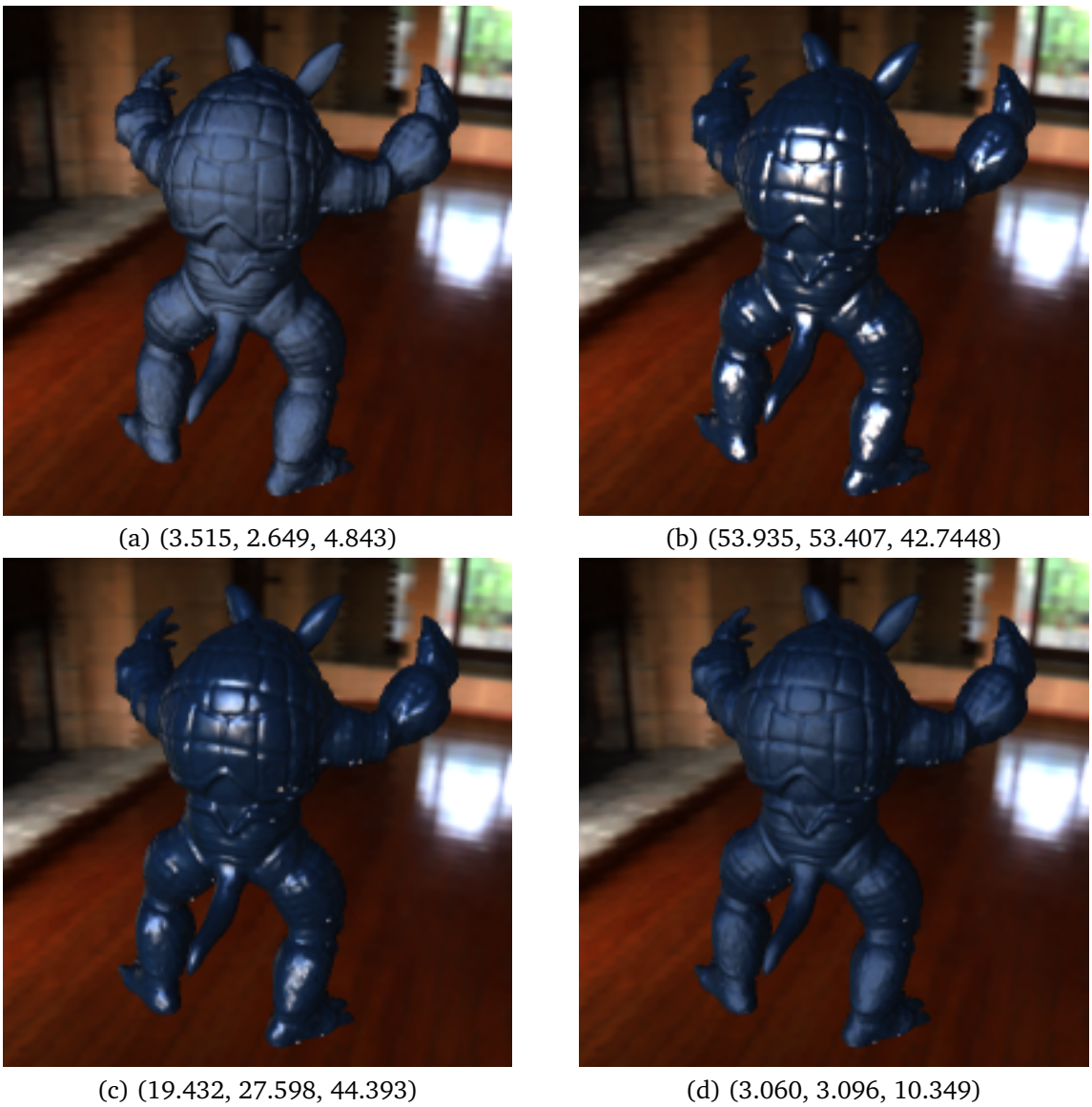
Our implementation of virtual glossmeter is fully customizable. User can change size and location of apertures, specular angle and surface orientation.

Unfortunately, we could not find real reference images of complex materials with measured gloss value and supplied BRDF model with parameters (for synthetic reconstruction). Thus we were not able to compare and validate the results of our virtual glossmeter to results of a real glossmeter. For now we show off few BRDF setups with corresponding measured gloss unit results in Figure 2.4

The GUI allows to zoom and rotate scene utilizing quaternion math to overcome gimbal lock.

3D model is imported within our own Wavefront .obj file loader. We assume rather simplified version of that format – only definition of vertex data (position, normal, texture coordinates) and triangular faces are supported. Our 3D models do not come with tangent space vectors beside vertex normal. So the computation of tangent and bitanget vector is necessary.

Please note, we are referring to *bitangent*. We emphasize this, because



(a) (3.515, 2.649, 4.843)

(b) (53.935, 53.407, 42.7448)

(c) (19.432, 27.598, 44.393)

(d) (3.060, 3.096, 10.349)

Figure 2.4: Example virtual glossmeter measurements at 20°, 60°and 85°angles in gloss units. Scene with Stanford Armadillo model and Ennis lighting environment.

for some reason the word *binormal* is often used with the same meaning. This is an error since binormal is term used in study of curves.

The tangent vector $\vec{t}$ at some vertex is computed by Gram-Schmidt orthonormalizing process using vertex normal and global reference vector (set to $(1, 1, 1)$) and bitangent vector $\vec{b}$ is then given by $\vec{b} = (\vec{t} \times \vec{n})$. The absence of intrinsic tangent spaces and using the global reference vector to compute them yields visual artifacts in the direction of the reference vector and opposite one (*i.e.* $(1, 1, 1)$ and $(-1, -1, -1)$). However, this is not defect of the visualization method itself and can be solved by providing correct tangent spaces within a 3D model.

2.6 HDRI & Post-processing

Another implemented feature is loading of the HDR images. We support RGBE and OpenEXR formats with various parametrization: panoramatic, cube cross and angular probes.

We use these HDR environment maps to light the scene. Consequently we have to deal with the rendered HDR images. Tone mapping is the task to represent this high dynamic range on low dynamic range displays appropriately. Many different algorithms were proposed, unfortunately, most of them are not suited for realtime rendering due to their time costs. To circumvent these problems we choose a simple logarithmic compression of the dynamic range [RWPD05], which may in fact compete with more complex operators when applied to medium-dynamic-range images.

The logarithm is a compressive function for values larger that 1, and therefore range compression may be achieved straightforward by mapping luminances as follows

$$L_d(x, y) = \frac{log_10(1 + L_w(x, y))}{log_10(1 + L_{max})}$$

where L_w and L_{max} are the *world* luminance of a particular pixel and maximum luminance (of the whole image), respectively. L_w is computed from the RGB triplet by the dot product with coefficients $(0.2126, 0.7152, 0.0722)$. L_d is the resulting *display* luminance. To get the compressed image we have to recombine luminance values into a color image via $(R_d, G_d, B_d) =$

$\frac{L_d}{L_w}(R_w, G_w, B_w)$.

The tone mapping routine is done on the GPU. We do not compute L_{max} from the image, because it would probably taken some parallel reduction for finding the maximum, hence slowing the frame rate of the display routine. Instead we choose the way of providing slider in our GUI to change the L_{max} value. Effects can be seen at Figure 2.5

(a) Under-exposed

(b) Good exposure

(c) Over-exposed

Figure 2.5: Effects of global logarithmic tone mapping operator on scene with Stanford Armadillo model and Ennis lighting environment.

2.7 Data organization

Our visualization tool works with projects, which can be saved to and loaded from a file. Each project consists of arbitrary number of HDR lightings and 3D models. Each model may have assigned BRDF along with corresponding parameters. Only one lighting and one model can be displayed at a time. Project data file is stored as zlib compressed binary data stream.This is implemented via `boost` *serialization* library and the data may not be portable between different architectures (different endianness, *etc.*).

2.8 Project Directory Structure

- `brdf` - BRDF related code
- `core` - global project constants, color, *etc.*
- `generators` - generators of random samples (used by SH)
- `gpu` - OpenGL Shading Language wrapper, common functions
- `helpers` - function used as intermediate product (function objects)
- `io` - loading of 3D models and HDR images
- `math` - small vectors (2, 3, 4), small matrices (3x3, 4x4), generic fixed size vector, Haar wavelet transform *etc.*
- `memory` - various templated memory structures
- `methods` - now only wavelet rotation environment mapping
- `misc` - index sorting, quick select, trackball
- `numerical` - sparse vectors, sparse and dense matrices, CPCA
- `projections` - projections to basis functions (only SH)
- `scene` - containers holding lightings and models
- `shaders` - GPU shaders

- `solvers` - GI solvers (only for SH)
- `tests` - various purpose-built tests
- `widgets` - GUI related code

2.9 Working Environment

We use g++ from GNU Compiler Collection (GCC) of version 4.1.1. Although OpenMP support in GCC is scheduled for version 4.2 it is already include in 4.1 versionwhichh comes with Fedora Core 6.

For tracking and managing changes of our source code we use the Subversion (SVN). The goal of the Subversion project is to build a version control system that is a modern compelling replacement for Concurrent Versioning System (CVS) which is probably the most spread system in the open source community.

SVN is better than CVS having many features, to mention few: commits are truly atomic operations; directories, renames, and file metadata are versioned *etc.*.

We adopt the basic SVN repository paradigm - repository consists of the trunk (main development branch), tags (for tagged snapshots) and branches (experimental branches...) directory.

For building Build (construction) tools – Autotools – which is *de facto* most spread building solution in the open source development projects.

2.10 Asset preparation

For preparation of digital assets we use mainly two software product:

Blender - for preparation of 3D models and exporting to .obj format.

pfstools - for manipulation (resizing) of HDR images. All programs in the package exchange data using unix pipes and a simple generic HDR image format (pfs).

```
pfsinrgbe input_file_name | pfssize -r scale_ratio | \
pfsoutrgbe output_file_name
```

Chapter 3

Results, Conclusion & Future Work

We implemented environment mapping visualization tool preserving all frequencies in BRDF and lighting environments. Our contribution to the original work lies in the support of multi-core CPUs, which speeds up the precomputation and rendering code almost linearly to the number of cores available.

Another contribution is the profit from the presence of fast programmable graphic processor, which we use to perform tileboarded nonstandard Haar wavelet transform. This is a step toward more dynamic changes of the lighting environment and it also overcomes the need of keeping rotation matrices in the memory, thus lowers the overall consumed memory.

In summary, we implemented standalone visualization tool enabling us to render highly specular materials under natural detail lighting environments at interactive rates, employing multi-core CPU and fast GPU to speed up computation, while giving the virtually measured numerical specular gloss value of rendered material as feedback to the user.

As we were developing under Linux only, we were not able to use profile tools for our shader code – Nvidia NVPerfKit is tied with much older Linux kernel as we are using and gDebugger is for Windows platform only. We hope NVPerfKit for Linux will be better supported in the future so we could get the maximum from our GPU code.

In the future we would like to extend environment mapping technique

to triple product integral scene relighting and thus incorporating seft-shadowing of objects. Also we would like to switch from per-vertex to per-pixel computation, performed on the GPU. We are looking to implement spherical wavelet transform with help of new geometric shaders, which now come within newest graphic cards.

Appendix A

Libraries & Tools

In this appendix we list libraries that we actively use in our software product Tools used to prepare assets. Stuff we found useful to complete my thesis:

A.1 Tools

Subversion

`http://subversion.tigris.org` **License:** Subversion License

The goal of the Subversion project is to build a version control system that is a modern compelling replacement for Concurrent Versioning System (CVS) which is probably the most spread version control system in the open source community.

Blender

`http://www.blender.org` **License:** GNU GPL

Blender is the free open source 3D content creation suite, available for all major operating systems under the GNU General Public License.

pfstools

`http://www.mpi-inf.mpg.de/resources/pfstools` **License:** GNU LGPL

pfstools package is a set of command line programs for reading, writing, manipulating and viewing high-dynamic range (HDR) images and video frames.

A.2 Libraries

gtkmm

`http://www.gtkmm.org` **License:** LGPL

gtkmm is an C++ wrapper for the Gtk+ library. We have chosen it because this is the GUI library we used in recent projects and are most familiar with.

gtkglextmm

`http://gtkglext.sourceforge.net/` **License:** LGPL

gtkglexmm is addition to gtkmm. It allows gtkmm to use OpenGL through additional GDK widgets.

Boost

`http://www.boost.org` **License:** Boost Software License

The Boost C++ libraries are a collection of peer-reviewed, open source libraries that extend the functionality of C++. Boost libraries are intended to be widely useful, and usable across a broad spectrum of applications (providing the programmer with smart pointers to more involved high-level C++ template metaprogramming framework). Part of it will hopefully become part of a C++ in the future versions of C++ standard.

OpenMP

`http://www.openmp.org` **License:**

The OpenMP (Open Multi-Processing) is an industry-standard API for multi-platform shared memory multiprocessing programming in C/C++ and Fortran on many architectures.

OpenGL & OpenGL Shading Language

`http://www.opengl.org` **License:**

OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. Currently there is specification for version 2.1.

OpenEXR

`http://www.openexr.com` **License:** modified BSD

OpenEXR is a high dynamic-range (HDR) image file format developed by Industrial Light & Magic for use in computer imaging applications.

Cuba

`http://www.feynarts.de/cuba/` **License:** LGPL

Cuba is a library for multidimensional numerical integration offering four independent methods of integration - deterministic/Monte Carlo with different strategies of variance reduction.

References

[CT81] Robert L. Cook and Kenneth E. Torrance. A reflectance model for computer graphics. In *SIGGRAPH '81: Proceedings of the 8th annual conference on Computer graphics and interactive techniques*, pages 307–316, New York, NY, USA, 1981. ACM Press.

[DK05] Roman Durikovič and Ryou Kimura. Spectrum-based rendering using programmable graphics hardware. In *SCCG '05: Proceedings of the 21st spring conference on Computer graphics*, pages 233–236, New York, NY, USA, 2005. ACM Press.

[GKMD06] Paul Green, Jan Kautz, Wojciech Matusik, and Frédo Durand. View-dependent precomputed light transport using nonlinear gaussian function approximations. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 7–14, New York, NY, USA, 2006. ACM Press.

[GKPB04] Pascal Gautron, Jaroslav Křivánek, Sumanta N. Pattanaik, and Kadi Bouatouch. A novel hemispherical basis for accurate and efficient rendering. In *Rendering Techniques 2004, Eurographics Symposium on Rendering*, pages 321–330, June 2004.

[GLGM06] Naga K. Govindaraju, Scott Larsen, Jim Gray, and Dinesh Manocha. Memory—a memory model for scientific algorithms on graphics processors. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 89, New York, NY, USA, 2006. ACM Press.

[Gre86] Ned Greene. Environment mapping and other applications of

world projections. *IEEE Comput. Graph. Appl.*, 6(11):21–29, 1986.

[KSS02] Jan Kautz, Peter-Pike Sloan, and John Snyder. Fast, arbitrary brdf shading for low-frequency lighting using spherical harmonics. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 291–296, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[Lef06] Aaron Lefohn. *Glift: Generic Data Structures for Graphics Hardware*. PhD thesis, Computer Science, University of California, Davis, September 2006.

[LK03] Jaakko Lehtinen and Jan Kautz. Matrix radiance transfer. In *SI3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 59–64, New York, NY, USA, 2003. ACM Press.

[LSSS04] Xinguo Liu, Peter P. Sloan, Heung Y. Shum, and John Snyder. All-frequency precomputed radiance transfer for glossy objects. In Alexander Keller and Henrik W. Jensen, editors, *Eurographics Symposium on Rendering*, pages 337–344, Norrköping, Sweden, 2004. Eurographics Association.

[MHL+06] Wan-Chun Ma, Chun-Tse Hsiao, Ken-Yi Lee, Yung-Yu Chuang, and Bing-Yu Chen. Real-time triple product relighting using spherical local-frame parameterization. *Vis. Comput.*, 22(9):682–692, 2006.

[MLH02] David K. McAllister, Anselmo Lastra, and Wolfgang Heidrich. Efficient rendering of spatial bi-directional reflectance distribution functions. In *HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 79–88, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[MY02] Wan-Chun Ma and Chia-Lin Yang. Using intel streaming simd extensions for 3d geometry processing, 2002.

[NRH03] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. All-frequency shadows using non-linear wavelet lighting approximation. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 376–381, New York, NY, USA, 2003. ACM Press.

[NRH04] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. Triple product wavelet integrals for all-frequency relighting. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 477–487, New York, NY, USA, 2004. ACM Press.

[Pee93] Mark S. Peercy. Linear color representations for full speed spectral rendering. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 191–198, New York, NY, USA, 1993. ACM Press.

[PH03] Emil Praun and Hugues Hoppe. Spherical parametrization and remeshing. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 340–349, New York, NY, USA, 2003. ACM Press.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[RWPD05] Erik Reinhard, Greg Ward, Sumanta Pattanaik, and Paul Debevec. Morgan Kaufmann Publishers, December 2005.

[SHHS03] Peter-Pike Sloan, Jesse Hall, John Hart, and John Snyder. Clustered principal components for precomputed radiance transfer. *ACM Trans. Graph.*, 22(3):382–391, 2003.

[SKS02] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 527–536, New York, NY, USA, 2002. ACM Press.

[Slo06] Peter-Pike Sloan. Normal mapping for precomputed radiance transfer. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 23–26, New York, NY, USA, 2006. ACM Press.

[SLS05] Peter-Pike Sloan, Ben Luna, and John Snyder. Local, deformable precomputed radiance transfer. *ACM Trans. Graph.*, 24(3):1216–1224, 2005.

[SS95] Peter Schroeder and Wim Sweldens. Spherical wavelets: efficiently representing functions on the sphere. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 161–172, New York, NY, USA, 1995. ACM Press.

[TL04] Rui Wang 0003, John Tran, and David P. Luebke. All-frequency relighting of non-diffuse objects using separable brdf approximation. In *Rendering Techniques*, pages 345–354, 2004.

[TLQ+05] Ping Tan, Stephen Lin, Long Quan, Baining Guo, and Heung-Yeung Shum. Multiresolution reflectance filtering. In *Rendering Techniques*, pages 111–116, 2005.

[TS06] Yu-Ting Tsai and Zen-Chung Shih. All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation. *ACM Trans. Graph.*, 25(3):967–976, 2006.

[WM01] Harold B. Westlund and Gary W. Meyer. Applying appearance standards to light reflection models. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 501–51., New York, NY, USA, 2001. ACM Press.

[WNLH06] Rui Wang, Ren Ng, David Luebke, and Greg Humphreys. Efficient wavelet rotation for environment map rendering. In *Eurographics Symposium on Rendering*, pages 173–182. Eurographics Association, 2006.