2006

Wavelets In Real-time Rendering

Weifeng Sun
University of Central Florida

Wavelets in Real-time Rendering

by

Weifeng Sun
B.S. University of Petroleum, China, 1997
M.S. Institute of Software, Academia Sinica, 2000
M.S. University of Central Florida, 2006

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2006

Major Professor:
Amar Mukherjee

ABSTRACT

Interactively simulating visual appearance of natural objects under natural illumination is a fundamental problem in computer graphics. 3D computer games, geometry modeling, training and simulation, electronic commerce, visualization, lighting design, digital libraries, geographical information systems, economic and medical image processing are typical candidate applications. Recent advances in graphics hardware have enabled real-time rasterization of complex scenes under artificial lighting environment. Meanwhile, pre-computation based soft shadow algorithms are proven effective under low-frequency lighting environment. Under the most practical yet popular all-frequency natural lighting environment, however, real-time rendering of dynamic scenes still remains a challenging problem.

In this dissertation, we propose a systematic approach to render dynamic glossy objects under the general all-frequency lighting environment. In our framework, lighting integration is reduced to two rather basic mathematical operations, efficiently computing multi-function product and product integral. The main contribution of our work is a novel mathematical representation and analysis of multi-function product and product integral in the wavelet domain. We show that, multi-function product integral in the primal is equivalent to summation of the product of basis coefficients and *integral coefficients*. In the dissertation, we

give a novel *Generalized Haar Integral Coefficient Theorem.* We also present a set of efficient algorithms to compute multi-function product and product integral.

In the dissertation, we demonstrate practical applications of these algorithms in the interactive rendering of dynamic glossy objects under distant time-variant all-frequency environment lighting with arbitrary view conditions. At each vertex, the shading integral is formulated as the product integral of multiple operand functions. By approximating operand functions in the wavelet domain, we demonstrate rendering dynamic glossy scenes interactively, which is orders of magnitude faster than previous work. As an important enhancement to the popular Pre-computation Based Radiance Transfer (PRT) approach, we present a novel *Just-in-time Radiance Transfer* (JRT) technique, and demonstrate its application in real-time realistic rendering of dynamic all-frequency shadows under general lighting environment.

Our work is a significant step towards real-time rendering of arbitrary scenes under general lighting environment. It is also of great importance to general numerical analysis and signal processing.

To my beautiful wife Miao, for your love

ACKNOWLEDGMENTS

I would like to thank my advisor, Prof. Amar Mukherjee, for insightful discussions and for his guidance, encouragement and support during all these years. It has been an honor and a privilege to work with Amar through these years.

I would also like to thank other committee members, Prof. J. Michael Moshell, Prof. Hassan Foroosh and Prof. Niels da Vitoria Lobo, for the advice, support and encouragement they have provided over the years. Especially I would like to thank Prof. Sumanta N. Pattanaik for introducing graphics/SIGGRAPH to me and for several discussions at different phases of my research. Special thanks also goes to Prof. Mubarak Shah and Prof. Charles Hughes for occasional discussions and encouragements. Thank Ruifeng Xu, Yugang Min, Nan Zhang, Tao Tao, Ravi Vijaya Satya and many others for being such wonderful friends.

In particular, I am deeply grateful to my parents, relatives, my wife and our princess Ariel for their endless support and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

My understanding to computer graphics research is *to interactively simulate natural objects under natural illumination, with natural surface appearance and by natural dynamics*. Over the years, numerical algorithms and systems have been developed to simulate natural effects based on physics laws. Using some off-line simulation systems, we are able to produce some photo-realistic computer-generated imageries (CGI), such as those special effects used in the film industry. Thanks to the recent advances in programmable graphics processing units (GPU) and new image synthesis techniques, even in the field of interactive rendering, such as in training and simulation, 3D geometry modelling, lighting design, and most popularly, 3D computer games, we witnessed a significant progress in enhancing the realism of lighting and shading effects.

Despite these improvements, there are still issues in physical-based simulation systems. These issues are largely due to the practical limitations of the computer systems used in the simulation. Ideally, we would like to have a 'perfect' Turing machine with 'unlimited' long tape and, most importantly, with 'unlimited' processing power. Practically, however,

we are only able to load a very limited number of data into the memory subsystem of the simulation systems, which is relatively small compared to the extraordinarily huge size of the real data. Meanwhile, the complexity of the inter-interactions in the simulation system is so extraordinarily high that it is far beyond the processing power of the current computer hardware. Due to the limitation of the memory subsystem, the granularity of the sampling scheme used in the simulation has to be decreased to keep the problem tractable. This, inevitably, leads to the 'infamous' aliasing effects. An alternative approach is to employ some data compression techniques to exploit the redundancy in the input data, and as a result to fit more data in the available memory. This approach, nevertheless, puts more burden on the already-limited processing units. Considering that normally simulation is performed on the uncompressed data, de-compression must be conducted on-the-fly to transform the compressed data into the uncompressed format.

In this dissertation, we present a unified framework, *compressed domain real-time rendering*, to address the limitation of the memory subsystem and the processing units simultaneously in the field of real-time rendering. In our framework, input data are approximated and efficiently compressed in the wavelet domain, and the complexity of the simulation system is effectively reduced by performing simulation directly in the compressed wavelet domain. We demonstrate our approach in real-time rendering of dynamic glossy objects with realistic all-frequency shadows under distant all-frequency environment lighting. Note that our approach may have broad applications in general numerical analysis and signal processing.

1.1 Physical-Based Image Synthesis

Here we assume that there is no participating media in the scene, and that light travels along straight lines. In this case, the equilibrium distribution of light energy in the scene can be described using the rendering equation [Kaj86], which represents the exitant radiance $L_o(x \rightarrow \Theta)$ at surface point x in the outgoing direction Θ as the function of the self-emission $L_e(x \rightarrow \Theta)$ and the reflected radiance from the neighboring objects:

$$L_o(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_{\Omega_x} f_r(x, \Psi \leftrightarrow \Theta) L(x \leftarrow \Psi) \cos(N_x, \Psi) d\omega_\Psi \tag{1.1}$$

where f_r is the $BRDF$, N_x is the normal at x and Θ is the incoming direction.

There are five issues inherent in the rendering equation: integration complexity, transport complexity, geometric complexity, material complexity and lighting complexity. These issues are crucial for the efficient simulation of photo-realistic imageries.

Integration Complexity As shown in the rendering equation, exitant radiance at a surface point involves the integration of all incoming radiance, which inherently is a recursive operation. Generally speaking, the rendering equation is a *Fredhom integral equation of the second kind*. Traditional global illumination approaches focus on solving the *recursive* nature of the light transport, which are only good for off-line rendering due to the extraordinarily long computation. As a very important leeway, the recursive nature of the light integration can be effectively discarded by assuming that the scene is illuminated by the environment light. This assumption, amazingly, does not degrade the perceptual quality of the rendered

images. Although indirect illuminations were eliminated in the computation, the generated imageries still demonstrates a strong sense of global illumination effects and, most importantly, at interactive framerate. In this dissertation, we follow this approach and focus on the efficient computation of the light integration in the dynamic scenes, illuminated by distant environment lighting.

Transport Complexity Physical-based simulation must take care of the complex photon interactions between different surface patches in the scene. These light interactions account for many global illumination lighting effects, such as shadows, self-reflections, inter-reflection, caustics, et al. To simulate these lighting effects, visibilities between different surface patches must be explicitly computed. Unfortunately, visibility computation is notoriously slow and not good for real-time rendering. As a result, many real-time rendering approaches, such as pre-computed radiance transfer, compute the visibilities completely off-line. There are also some other approaches opt to local illumination models, such as Phong model, where light transport is aggressively simplified.

Geometric Complexity One natural approach to describe the surface geometry is to densely discretize the object surface, and encode the topology (which only represents the connectivity) with supplementary surface position information. An appropriate sampling resolution must be determined for the goal of efficient measurement and representation. Generally speaking, we many describe the surface geometry progressively, from the coarsest macro-scale level to the intermediate meso-scale level, and finally at the finest micro-scale level. For each level, there are many options available to encode the topology and position

information, such as point clouds, triangle/quad meshes, parametric surfaces and volumetric representations.

Note that for some natural objects, such as vegetables, fruits, grasses, trees, flowers, rusts, barks, sands, snows and smokes, etc, this level-of-detail representation will have problems. One possible approach is to describe these surfaces quantitatively using some distribution functions. Another challenging issue is to simulate the natural dynamics of these objects, such as fluid dynamics and mesh manipulation.

Material Complexity The surface appearance of natural objects closely relates to transport complexity and geometry complexity (surface geometry). For objects exhibiting strong subsurface scattering or translucency effects, material complexity is coupled with the transport complexity. For objects with complexity surface topology, such as rusts, the material complexity is closely coupled with the geometry complexity. Quantize the surface property using intrusive or non-intrusive measurement equipments remains one of the most challenging problems in computer graphics. Historically this line of research focuses on BRDF measurement and recently enhanced with spatial/temporal extension.

Note that there are four kinds of complexities in the material complexity: spatial complexity, depth complexity, angular complexity and temporal complexity. Historically, bidirectional reflectance distribution function (BRDF) is used to represent the view-dependent angular variations of surface material, while texture is used to describe the spatial variation. Due to its simple form, texture is great in representing high-frequency details. On the other hand, surface geometry is used to represent low-frequency surface variations. A very promising

approach to describe depth complexity, such as subsurface scattering effects, is the bidirectional surface scattering distribution function (BSSRDF). How to efficiently represent all-frequency surface variation, by combining the geometry, BRDF and texture remains one of the hottest research area in computer graphics. Currently there are also some nice experiments [GTR, WTL] in capturing and representing temporal complexity, although their practical applicability is highly obscure, considering the prohibitively high storage complexity.

There have been a vast amount of research efforts in studying spatial complexity of the surface materials, a.k.a., texture synthesis, which was originally proposed as an efficient quick-fix for real-time rendering. Very recently, research focuses are shifted to the angular complexity, especially due to the great success in simulating low-frequency soft shadows. Most of these research efforts decompose the captured data into sets of forms with different physical meanings, powered by some factorization techniques. This line of research normally uses computer vision techniques, and can be categorized as the inverse problem in computer graphics.

Lighting Complexity Real-world light sources exhibit a wide range of properties, such as volumetric, time-dependent, angular-dependent (directional), spatial-dependent, et al. Due to its simple implementation, point light sources are traditionally popular, especially in some real-time or interactive applications. Currently environment light sources become popular in some interactive image synthesis systems. However, efficiently capturing real-world light sources, and the related problem of compactly representing the captured dataset remains

one of the most challenged problems in computer graphics, and may create a completely new industry in the future.

All the aforementioned complexity issues arise in simulating physically valid light transport. In fact, since the main purpose of computer-generated imaginaries is for display which is to be observed by human beings, there are some opportunities for optimization, for example, by exploiting the spacial/temporal lighting-sensitiveness of human eyes. Perceptually-based optimization techniques is one of the research area in computer graphics but whined a bit recently due to the slow progress in the low-level vision research. Another optimization possibility is based on the observation that, since computer-generated imaginaries are used by human beings as an effective communication tool, we can enhance the communication channel by manipulating synthesized imaginaries at high-level, for example, at the semantic level. In its simplest form, this line of research extends to image/video enhancement and non-traditional rendering.

1.2 Data Compaction and Function Approximation

As shown in the previous section, given a certain signal/function, we are interested in representing this function approximately and efficiently (with less space). Function approximation is closely related to data compression, or data compaction, which is a well researched area.

In this section, we briefly overview some important data compaction techniques, such as entropy encoding, quantization/clustering, hashing and basis function approximation.

Entropy Encoding

Given a set of independent N events with probabilities p_i $(1 \leq i \leq N)$, such that $\sum_{i=1}^{N} p_i = 1$, the *first-order entropy* H is the sum of the self-information over all events,

$$H = -\sum p_i \log p_i \tag{1.2}$$

which also measures how much uncertain we are aware of the outcome. Generally, entropy is the lower bound on compression, and it is very difficult to compute entropy. Data compression, however, seeks a message representation that uses as few bits as possible to retain the information content. There are many models have been proposed to improve entropy computation, and many compression algorithms have been proposed to compress messages close to the entropy. Two of the most popular entropy encoding algorithms are Huffman encoding and Arithmatic encoding. Given a message, Huffman coding creates binary (Huffman) tree such that path lengths correspond to symbol probabilities. Then path labels are used to encode the message. Whereas, Arithmetic coding combines probabilities of subsequent symbols into a single fixed-point number of high precision. Then the binary representation of that number is used to encode the message.

Huffman codes are minimum redundancy codes for a given probability distribution of the message. Huffman coding guarantees a coding rate l_H within one bit of the entropy H, $H \leq l_H \leq H + 1$. Theoretically, average code length $l_H < H + p_{max} + 0.086$, where p_{max} is

the probability of the most frequently occurring symbol. As a result, if p_{max} is quite big, in other words, if the alphabet is small and the probability of occurrence of different symbols is skewed, Huffman coding will be quite inefficient. However, if the alphabet is large and probabilities are not skewed, Huffman coding rate is pretty close to entropy.

Arithmetic coding is especially suitable for small alphabet (such as binary sources) with highly skewed probabilities. It is very popular in image and video compression applications.

Quantization and Clustering

Quantization is the process of representing a large (possibly infinite) set of values with a much smaller set. It is one of the simplest and most general idea in lossy compression. Vector quantization maps k-dimensional vectors in the vector space $\Re^k$ into a finite set of vectors $Y = \{y_i, 1 \leq i \leq N\}$, where each vector y_i is called a codeword, and the set of all codewords is called a codebook. A Voronoi region is associated with each codeword y_i,

$$V_i = \{x \in \Re^k : \| x - y_i \| \leq \| x - y_j \|, \qquad \text{for all } i \neq j\} \tag{1.3}$$

which defines a nearest neighbor region. The set of all Voronoi regions partition the entire space $\Re^k$.

One of the most popular vector quantization algorithm, Linde-Buzo-Gray (LBG), or the generalized Lioyd algorithm (GLA), is very similar to the k-means clustering algorithm. It is no surprise that other clustering algorithms, such as mean-shift algorithm, can also be used in vector quantization.

In vector quantization, each input vector is mapped to the closest codeword. A common practice to accelerate the mapping operation is to organize the codebook with some tree-structures. Tree-structured vector quantization (TSVQ) is very effective in decreasing the mapping complexity. By substituting the regular clustering algorithm with TSVQ, the popular clustered principle component analysis algorithm (CPCA) can be implemented more efficiently to decrease the pre-processing time in recently proposed pre-computation based real-time rendering approaches.

Hashing

Recently, a very interesting research area in computer graphics is how to organize the memory system on the GPU efficiently. This year, Lefebvre and Hoppe [LH06] proposed a *perfect hashing* technique to pack sparse data into a compact table. The main advantage of this technique is that it permits efficient random access, which is very prominent to graphics hardware. Based on pre-computed information, they designed a perfect multi-dimensional hash function without a single hash collision. This suggests a new lossless approach to approximating certain functions compactly.

Basis Functions

Another popular approach to function approximation is transform coding. By transform correlated data into a representation where they are uncorrelated, the transformed values are usually smaller on average than the original values. For lossy compression, the transformed coefficients can now be quantized/compressed using their statistical properties (or using the

entropy encoders), reducing the redundancy, and therefore producing a much compressed representation of the original data. In general, the key of the transform coding is to find an set of good basis functions[1].

FOURIER (1807)

In 1807, Joseph Fourier asserted that any 2π-periodic square-integral function[2] $f(x)$ has a *Fourier series* representation:

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{inx} \tag{1.5}$$

where the *Fourier coefficients* c_n are calculated by

$$c_n = \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-inx} dx \tag{1.6}$$

Fourier series representation (eq. 1.5) utilities a set of *basis functions*, e^{inx}. It is on this set of basis functions that function $f(x)$ is decomposed into a sum of infinite components. The set of basis functions in Fourier series representation have two important features:

Orthogonal For any two basis function b_1 and b_2, we have:

$$\langle b_1, b_2 \rangle = \begin{cases} 1, & \text{for all} \quad b_1 = b_2 \\ 0, & \text{otherwise} \end{cases} \tag{1.7}$$

[1] A basis is a collection of functions. Given a function space, any function in the space can be uniquely represented as a linear combination of the basis functions. Here we also define the *inner product* $\langle g, h \rangle$ of two functions g and h as $\int g(t) \cdot h(t) dt$.

[2] A *square integrable function* $f : \mathcal{R} \to \mathcal{R}$ satisfies following condition:

$$\|f\| = \langle f, f \rangle^{1/2} = [\int_{-\infty}^{\infty} f^2(t) dt]^{1/2} < \infty \tag{1.4}$$

Dilation All basis functions in the Fourier series representation are generated by *dilation* of a single function

$$b_0 = e^{ix} = \cos x + i \sin x \tag{1.8}$$

which is a *sinusoidal wave*, the *only* function required to generate all 2π-periodic square-summable functions.

As a result, every 2π-periodic square-integrable function can be decomposed as a *superposition* of integral dilations of the sinusoidal basis function. Fourier series representation has been a very popular tool as *frequency analysis*. Note that there are an infinite number of basis sets conforming to *orthogonal* and *dilation* properties.

Lifted by Legendre polynomials, Fourier bases have been extended over the sphere, which defines a novel orthonormal basis set, spherical harmonics. Spherical harmonics have been successfully applied to represent 4D BRDF functions, as well as in recently very active low-frequency shadow analysis and generation. Note that one of the most prominent problem of Fourier representation, and therefore, spherical harmonics, is that the basis sets are not compactly supported. The directly implication of this limitation is that they are not good at efficiently representing high-frequency signals.

HAAR (1909)

In 1909, Alfréd Haar discovered the simplest set of basis functions to decompose a continuous square-integrable function. He found following orthonormal basis (also called a Hilbert basis)

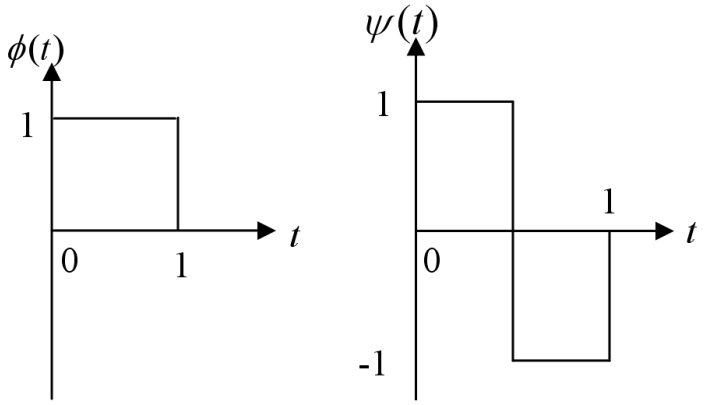


Figure 1.1: 1D Haar mother scaling basis function (left) and mother wavelet (right).

for $L^2[0,1]$:

$$b_0(x) = 1, \qquad 0 \le x < 1 \tag{1.9}$$

$$b_n(x) = 2^{j/2} b_1(2^j x - k), \qquad n = 2^j + k, j \ge 0, 0 \le k < 2^j \tag{1.10}$$

where

$$b_1(x) = \begin{cases} 1 & : \quad 0 \le x < \frac{1}{2} \\ -1 & : \quad \frac{1}{2} \le x < 1 \\ 0 & : \quad \text{else} \end{cases} \tag{1.11}$$

As a result, a continuous function $f(x)$ can be approximated by step functions (Haar basis functions) with coefficients as the mean values of $f(x)$ on the appropriate dyadic intervals. However, because Haar basis functions are not continuous in themselves, there are some problems with this Haar approximations. For example, if $f(x)$ is a C^1 continuous function

(a C^1 function has a continuous derivative), we may have to resort to higher-order polynomial basis functions.

The main theme of the dissertation focuses on a very interesting property of Haar basis functions, the integral coefficients, and their application in efficiently approximating (the integration of) the product of multiple functions.

DAUBECHIES (1988)

Perhaps the most exciting achievement in basis approaches, or in wavelets theory, is due to Ingrid Daubechies' work in finding a set of Haar-like wavelets with compact support, continuous derivatives, and most importantly, superior *compression*. These properties results their immediate applications in engineering signal processing, such as data compression. A detailed explanation on Daubechies wavelets is found in [Dau92].

1.3 Multi-Resolution Analysis (MRA)

> *The wavelet transfrom is a tool that cuts up data, functions or operators into different frequency components, and then studies each component with a resolution matched to its scale.*
>
> — *Dr. Ingrid Daubechies*

'Wavelets' is a very popular yet nice tool for approximating functions in the field of signal processing. It has been successfully applied in many areas such as time-frequency analysis,

functional representation, et al. Mathematically, wavelets is just a set of basis functions with some appealing properties. This chapter gives an overview of wavelets and multiresolution analysis.

Analogous to Fourier transform, a wavelet transform decomposes a function into a series of summation of basis coefficients of the appropriate basis functions.

Multi-resolution analysis (MRA) is a very nice to approximate a function progressively, from the coarsest scale to more finer scales. A MRA is a nested sequence

$$\cdots \subseteq V_{-1} \subseteq V_0 \subseteq V_1 \subseteq V_2 \subseteq \cdots \tag{1.12}$$

of subspaces of $\mathcal{L}^2[\mathcal{R}]$ with a scaling function $\phi(t)$ defined on V_0 such that

- $V = \cup_{n \in \mathcal{Z}} V_n$ is dense[3] in $\mathcal{L}^2[\mathcal{R}]$.
- The intersection of these subspaces is a singleton set containing the all-zero function or zero vector.
- $f(t) \in V_n$ iff $f(2^{-n} \cdot t) \in V_0$.
- $\{\phi(t-k)\}_{k \in \mathcal{Z}}$ is an orthonormal basis for V_0.

For any $j \in \mathcal{Z}$, we define W_j as the orthogonal complement of V_j in V_{j+1}. Consequently, we have

$$V_{j+1} = V_j \oplus W_j \tag{1.13}$$

[3]A subset B of A is *dense* in A if any given element in A can be approximated as closely as we like by an element in B. For example: the set of rational numbers Q is dense in real number set R.

In other words, we have

$$V_j = V_{j-1} \oplus W_{j-1} \tag{1.14}$$

and

$$V_{j+1} = V_{j-1} \oplus W_{j-1} \oplus W_j \tag{1.15}$$

It is trivial to show that for any $k \leq j$

$$V_{j+1} = V_k \oplus W_k \oplus W_{k+1} \oplus \cdots \oplus W_j \tag{1.16}$$

As a result, for any given function in V_{j+1}, it can be exactly represented by the scaling basis functions in V_{j+1}. It can also be decomposed as a linear combination of the scaling basis functions at a coarser resolution in V_k, followed by a sequence of functions defined in the detail spaces at resolution level $k, k+1, \cdots, j$ to represent the leftover details. Here, approximating the function at a lower resolution involves the scaling basis functions, while progressively representing the detail involves the wavelet basis functions.

1.3.1 Scaling Basis Function

The mother scaling function $\phi(t)$ defined on V_0 is the core of wavelet analysis. It has following properties:

- $\int_{-\infty}^{\infty} \phi(t)dt = 1$.
- It has unit energy. $|\phi(t)|^2 = \int_{-\infty}^{\infty} |\phi(t)|^2 dt = 1$.

- $\phi(t)$ is orthogonal to its integer translation $\phi(t-n)$.

From definition, set $\{\phi(t-k)\}$ constitutes an orthonormal basis for V_0, and set $\{\phi(2t-k)\}$ constitutes an orthonormal basis for V_1. Since $\phi(t) \in V_0 \subseteq V_1$, $\phi(t)$ can be represented as a linear combination of these basis functions in V_1, we have the following *dilation equation*:

$$\phi(t) = \sum_k c_k \phi(2t-k) \tag{1.17}$$

where $\sum_k |c_k|^2 < \infty$.

Taking integration of the dilation equation, we have:

$$\begin{aligned}
\int \phi(t)dt &= \int \sum_k c_k \phi(2t-k)dt && (1.18)\\
&= \sum_k c_k \int \phi(2t-k)dt && (1.19)\\
&= \sum_k c_k \int \phi(x)d(\frac{x+k}{2}) && (1.20)\\
&= \frac{1}{2}\sum_k c_k \int \phi(x)dx && (1.21)
\end{aligned}$$

As a result, we have:

$$\sum_k c_k = 2 \tag{1.22}$$

We can also take the square integration of the dilation equation:

$$\begin{aligned}
\int |\phi(t)|^2 dt &= \int \sum_k c_k \phi(2t-k) \sum_m c_m \phi(2t-m)dt && (1.23)\\
&= \sum_k \sum_m c_k c_m \int \phi(2t-k)\phi(2t-m)dt && (1.24)\\
&= \frac{1}{2}\sum_k \sum_m c_k c_m \int \phi(x-k)\phi(x-m)dx && (1.25)
\end{aligned}$$

Only if $k = m$, the integral on the right hand side is 1; otherwise, it is zero. So, we have:

$$\sum_k c_k^2 = 2 \tag{1.26}$$

1.3.2 Wavelet Basis Function

For the detail space W_0 in MRA, we define a mother wavelet function $\psi(t)$ with following properties:

- $\int_{-\infty}^{\infty} \psi(t)dt = 0$.
- It has unit energy. $|\psi(t)|^2 = \int_{-\infty}^{\infty} |\psi(t)|^2 dt = 1$.
- $\psi(t)$ is orthogonal to its integer translation $\psi(t-n)$.
- $\psi(t)$ is orthogonal to the scaling functions defined on V_0. $\langle \psi(t), \phi(t-k) \rangle = 0$.

From definition, $\psi(t) \in W_0 = V_1 - V_0 = \subseteq V_1$. Therefore, $\psi(t)$ can be represented as a linear combination of these basis functions in W_1, we have:

$$\psi(t) = \sum_k d_k \phi(2t-k) \tag{1.27}$$

which is very similar to the dilation equation.

1.3.3 Constructing Orthonormal Basis Functions

Practically, we are interested in the representations with normalized coefficients:

$$c_k = \sqrt{2}h_k \tag{1.28}$$

$$d_k = \sqrt{2}g_k \tag{1.29}$$

As a result, we have:

$$\phi(t) = \sum_k \sqrt{2}h_k\phi(2t-k) \tag{1.30}$$

$$\psi(t) = \sum_k \sqrt{2}g_k\phi(2t-k) \tag{1.31}$$

Here we consider the inner product of $\phi(t)$ and its translation $\phi(t-m)$:

$$\begin{aligned}
\langle\phi(t),\phi(t-m)\rangle &= 2\int \sum_k h_k\phi(2t-k))\sum_l h_l\phi(2t-2m-l)dt && (1.32)\\
&= 2\sum_k\sum_l h_k h_l \int \phi(2t-k))\phi(2t-2m-l)dt && (1.33)\\
&= \sum_k\sum_l h_k h_l \int \phi(x-k))\phi(x-2m-l)dx && (1.34)\\
&= \sum_k\sum_l h_k h_l \delta_{k,2m+l} && (1.35)\\
&= \sum_k h_k h_{k-2m} && (1.36)
\end{aligned}$$

Therefore, we have:

$$\sum_k h_k h_{k-2m} = \delta_m \tag{1.37}$$

$$\sum_k h_k = \sqrt{2} \tag{1.38}$$

$$\sum_k h_k^2 = 1 \tag{1.39}$$

From these equations we are able to derive some scaling basis functions. For example, suppose $K = 2$, we have :

$$\sum_{k=0}^{1} h_k = \sqrt{2} \tag{1.40}$$

$$\sum_{k=0}^{1} h_k^2 = 1 \tag{1.41}$$

The unique solution of these equations $h_0 = h_1 = \sqrt{2}$ gives the Haar scaling basis.

However, when $K > 3$, there are multiple solutions to these equations. In this case, we need to enforce further conditions for unique solution. In chapter 1, we show that Haar basis functions are step functions, and they are not good to approximate smooth functions. Theoretically, the quality of approximation using a basis set for a polynomial signal is related to the number of the *vanishing moments* of the basis functions. The m^{th} moment of a function f is defined as $\int t^k f(t)dt$.

By enforcing the first N moments of $\psi(t)$ vanish, we have

$$\int t^m \psi(t)dt = 0, \qquad \text{where} \quad m = 0, \cdots, N-1 \tag{1.42}$$

Then, we have

$$\sum k^m g_k = \sum (-1)^k k^m h_k = 0, \qquad \text{where} \quad m = 0, \cdots, N \tag{1.43}$$

Vanishing moments form a necessary condition for $\psi(t)$ to be C^N (N times continuously differentiable. In terms of the functional approximation, the magnitude of wavelet coefficients rapidly decreases as N grows.

In the case of $K = 4$, the solutions to these equations include the Daubechies 4-tap solution.

1.4 Real-Time Rendering

Conventional global illumination techniques [DBB03], such as ray tracing, photon mapping and radiosity, can take hours to generate photorealistic imageries, and that limits their application in interactive industrial design. Recent advances in computer hardware, such as SSE, SIMD, especially the programmable graphics processing units (GPUs), provide a solid foundation for real-time rendering. In fact, real-time rendering is not a new concept. Some long-existing examples are image-based rendering, plenoptic function and light field. In this section, we focus on pre-computation based real-time image synthesis techniques.

A common practice to accelerate global illumination is to retain only the direct lighting, for instance, environment mapping [BN76]. Recently, Sloan et al. [SKS02] proposed Pre-computed Radiance Transfer (PRT) to extend environment mapping technique for real-time rendering of soft shadows in a static scene. PRT supports diffuse and low-glossy materials, and has been extended to support all-frequency lighting [NRH03], high-glossy materials [LSS04, NRH04, WTL04], and dynamic objects with diffuse and low-glossy materials [JF03, KLA04, KL05, SLS05, ZHL05] . Here, we categorize some recent work on PRT as follows:

Static Scene, Low-Frequency Materials PRT methods [KSS02, SKS02, LK03, NRH03, SHH03, SLS03] approximate global illumination effects with coarsely filtered radiance transfer, and exploit orthonormality in basis functions to reduce the shading integral to a low-dimensional dot product between the lighting and transport coefficients. These approaches are effective in real-time relighting static scenes, but are limited to diffuse and low-glossy

materials. High-glossy materials would incur a prohibitively massive radiance transfer matrix to account for view-dependent lighting effects, and there is no effective compression technique to directly reduce the size of this massive dataset.

Static Scene, High-Frequency Materials Liu et al. [LSS04] and Wang et al. [WTL04] extend PRT to render glossy materials by separating BRDF and consequently keeping the size of the radiance transfer tractable. Ng et al. [NRH04] densely sample the lighting, BRDF and visibility for a static scene. They introduce triple product wavelet integral to render static objects under time-variant all-frequency lighting and view conditions in a few seconds. They also show that the product of two functions can be approximated using the tripling integral coefficients. This year Tsai and Shih [TS] used spherical radial basis functions to attack the problem.

Dynamic Scene, Low-Frequency Materials Sloan et al. [SKS02] demonstrate a neighborhood-transfer technique to render soft shadows from a dynamic neighboring object. James and Fatahalian [JF03] render deformable objects by pre-computing light transport for a representative set of poses. Sloan et al. [SLS05] propose a quick-rotatable radiance transfer using zonal harmonics. All these approaches are effective in rendering local soft shadows. To account for global cast shadows from dynamic neighboring objects, global visibilities with respect to these occluders are indispensable. Kautz et al. [KLA04] use an approximated hemispherical rasterization technique to render soft shadows with explicit occlusion information. Kontkanen and Laine [KL05] pre-compute ambient occlusion field to render soft cast shadows in real-time. Zhou et al. [ZHL05] pre-compute shadow field to render dynamic

objects illuminated by multiple local light sources. Their work on evaluating shading from a single light source falls into our framework on multi-function product integral. They address the problem by recursively computing two-function products using the tripling integral coefficients [NRH04]. The approach is improved in [RWS].

Dynamic Scene, High-Frequency Materials Note that all previous approaches are limited to diffuse and low-glossy materials and cannot account for high-frequency view-dependent lighting effects, such as intricate all-frequency cast shadows by multiple occluders and specularities in real-time, which is the main topic of the dissertation.

1.5 Contributions

This dissertation presents a significant theoretical and practical results of multiple function product and product integral, as well as real-time rendering of glossy objects in dynamic scenes. The dissertation is likely to have significant impacts both theoretically (computation of multi-way products) and practically (real-time rendering).

1.5.1 Theoretical Contributions

This dissertation makes a significant contribution to applied mathematics in generalizing previous methods for integrating the product of multiple functions represented in a wavelet basis.

In the dissertation, we present a new mathematical representation and analysis of multi-function product and product integral in the wavelet domain. We show that multi-function product integral in the primal corresponds to summation of the product of basis coefficients and *integral coefficients*. We propose a novel *Generalized Haar Integral Coefficient Theorem* to evaluate arbitrary Haar integral coefficients. This theorem has potential applications beyond the current dissertation, since it provides a set of efficient algorithms for the rather basic operation of multiplying M functions.

This dissertation also presents an interesting question on the existence of a family of wavelet filters with Haar-like integral efficiency. Proving the uniqueness of the Haar filter in terms of the efficient integral coefficient conditions, or finding its siblings, is a very important theoretical research topic which definitely deserves further consideration.

1.5.2 Algorithmic and Practical Contributions

In the dissertation, we consider real-time rendering of dynamic glossy objects under distant time-variant all-frequency environment lighting and arbitrary view conditions, with realistic all-frequency shadows. By formulating the shading integral at each vertex as the product integral of multiple functions, we reduce the problem to two rather basic mathematical operations — efficiently computing multi-function product and product integral. By approximating operand functions using wavelet encoding, both problems can be efficiently solved in the compressed domain. Based on the theoretical results, we present a set of efficient algorithms to approximate multiple function product and product integral.

In the dissertation, we also present a novel Just-in-time Radiance Transfer (JRT) technique for dynamic scenes. To our knowledge, this is the first actual demonstration of rendering dynamic glossy objects in real-time. We believe this JRT approach has the potential to significantly increase the realism for real-time rendering.

Our work is a significant step towards efficiently integrating the product of multiple signals, which is of great importance to general numerical analysis and signal processing.

1.6 Overview of the Dissertation

In chapter 1 we present a general introduction on compressed-domain real-time rendering and associated problems. In chapter 2, we present the systematic representation and analysis of quadruple function product integral in the wavelet domain. The results in this section is generalized in the next chapter, with an extensive analysis of multiple function product and product integral in the wavelet domain, as well as the practical application of Just-in-time Radiance Transfer (JRT) technique in real-time rendering. We conclude the dissertation with some discussion of the advantage of our work and some future research directions in chapter 4.

CHAPTER 2

QUADRUPLE WAVELET PRODUCT INTEGRAL

View-dependent all-frequency shadows and specular highlights are important lighting effects for interactive rendering systems. As shown in Figure 2.1, cast shadows (such as those on the floor) are mainly due to the occlusion by neighboring objects. Specular highlights, on the other hand, is the reflection of the light source due to high-frequency reflections of the surface material. In this chapter, we present a novel quadruple wavelet product integral technique to efficiently render such complex lighting effects in the dynamic scenes.

The approach is a significant advance to recently proposed pre-computed radiance transfer (PRT) technique. Sloan et al. [SKS02] approximate global illumination effects with coarsely filtered radiance transfer, and use double function product integral to reduce the shading integral to a low-dimensional dot product between the lighting and transport coefficients. This technique, however, is limited to low-frequency materials. Ng et al. [NRH04] proposed triple product integral to support high-frequency materials. Both approaches are quite effective in generating attached shadows, which is mainly due to the variation of the surface curvature. Furthermore, by fusing neighboring objects as a part of the shadow receiver,

they can even support cast shadows. Nevertheless, as an important visual cue, cast shadows suggest that shadow receiver and occluder are distinct objects, which further indicates that shadow occluder can move with changing cast shadows. Both double product integral and triple product integral techniques, however, are inherently limited to static scenes and cannot support for cast shadows from dynamic neighboring objects. Although several extensions [JF03] [KLA04] [KL05] [ZHL05] [SLS05] were proposed to leverage the approaches to dynamic scenes, fundamentally these extensions are double/triple product integral techniques. They cannot support dynamic glossy objects with all-frequency cast shadows.

In this chapter, we present a novel quadruple product integral technique to directly support all-frequency cast shadows by dynamic neighboring objects. The outstanding advantage of the new mathematical representation is the computational efficiency and flexibility. In this chapter, we explicitly incorporate global occlusions from neighboring objects into the shading integral. At each vertex, the outgoing radiance is formulated as the product integral of four functions, involving the lighting, BRDF, self-occlusion and global occlusion. We show that quadruple product integral in the primal corresponds to the summation of the product of quad-coefficient and basis coefficients. We propose a novel *Haar quad-coefficient theorem*, and present an efficient tree-structured sub-linear algorithm to integrate the product of four functions in the wavelet domain. The degeneration of the algorithm is a natural yet efficient triple product integral algorithm.

Our technique is based on the pre-computed information of individual scene entities, which is a generalization of previous work [NSD94, DAG95, SKS02, NRH03, NRH04, ZHL05]. We

use cubemaps to represent the lighting integral operand functions on each vertex, which are subject to tabulation, wavelet transformation and augmented quad-tree encoding in the pre-computation stage. In the rendering stage, we directly integrate the product of these four functions in the wavelet domain using quadruple product integral technique. Our approach is not limited to pre-animated models, and is applicable of interactively rendering dynamic scenes with complex realistic view-dependent lighting effect, such as those in the lighting design.

In the reminder of this chapter, we present our work as follows. In section 2.1 we present the Haar quad-coefficient theorem. The tree-structured quadruple product integral algorithm is presented in section 2.2. Details of the implementation and some rendering results are described in section 2.3. We conclude this chapter in section 2.4.

Figure 2.1: A 180,000 vertex dynamic scene rendered using quadruple wavelet product integral algorithms. Note that the approach inherently support dynamically moving table, with time-variant all-frequency lighting and view condition. Images are rendered in less than ten seconds per frame. More examples are shown in Figure 2.8.

2.1 Mathematical Formalization

In this chapter, we do not consider inter-reflections between any two surfaces in the scene. Exitant radiance B at a surface point x along view direction θ due to distant environment lighting L is the product integral over all incident directions sampled at a surrounding cubemap S:

$$B(x,\theta) = \int_S L(\varphi) O_s(x,\varphi) O_g(x,\varphi) \rho(x, \varphi \leftrightarrow \theta)(N \cdot \varphi) d\varphi \tag{2.1}$$

where φ is the incident direction, N is the normal at x, ρ is the BRDF, O_s is the *self-occlusion* at x, O_g is the *global-occlusion* at x from neighboring objects in the scene. In the equation, all operand functions are defined in a universal global coordinate system. Incorporating the cosine term $(N_x \cdot \varphi)$ into the BRDF ρ, shading on a fixed vertex x and view direction θ can be simplified as the product integral of four functions:

$$B = \int F_1(\varphi) F_2(\varphi) F_3(\varphi) F_4(\varphi) d\varphi \tag{2.2}$$

Projecting each operand function $F_i(\varphi)$ $(1 \leq i \leq 4)$ in equation (3.3) onto an orthonormal basis set $\mathcal{B}$ yields :

$$F_i(\varphi) = \sum_{j=1}^{M} f_{ij} b_j(\varphi) \tag{2.3}$$

where f_{ij} is the j^{th} basis coefficient, $b_j(\varphi)$ is the j^{th} basis function, and M is the size of $\mathcal{B}$. We define *quad-coefficient* C_{stuv} as :

$$C_{stuv} = \int b_s(\varphi) b_t(\varphi) b_u(\varphi) b_v(\varphi) d\varphi \tag{2.4}$$

Figure 2.2: Mother Haar basis functions. Functions are positive where white, negative where black and zero where gray. For simplicity, amplitude is ignored in the diagrams. More examples are shown in Figure 2.3.

It is worth noting that the quad-coefficient is a generalization of the coupling/tripling coefficient in [NRH04]. Substituting equation 3.4 into equation 3.3, we have:

$$\begin{aligned} B &= \int \sum_s f_{1s} b_s \sum_t f_{2t} b_t \sum_u f_{3u} b_u \sum_v f_{4v} b_v d\varphi \\ &= \sum_s \sum_t \sum_u \sum_v f_{1s} f_{2t} f_{3u} f_{4v} \cdot \int b_s b_t b_u b_v d\varphi \\ &= \sum_s \sum_t \sum_u \sum_v C_{stuv} \cdot f_{1s} f_{2t} f_{3u} f_{4v} \end{aligned} \tag{2.5}$$

Note that quadruple function product integral (equation 2.5) reduces to the quad-coefficient C_{stuv}, which is the focus of the next section. It is worth noting that the result in this subsection applies to any basis set $\mathcal{B}$.

2.1.1 2D Haar Bases

Nonstandard Haar wavelet transform [SDS96] transforms a $2^n \times 2^n$ image into a 2D signal with $2^n \times 2^n$ coefficients. Each coefficient corresponds to a basis function defined in region $\langle j, k, l \rangle$, where j is the *scale* ($0 \leq j < n$), k and l are spatial translations ($0 \leq k, l < 2^j$).

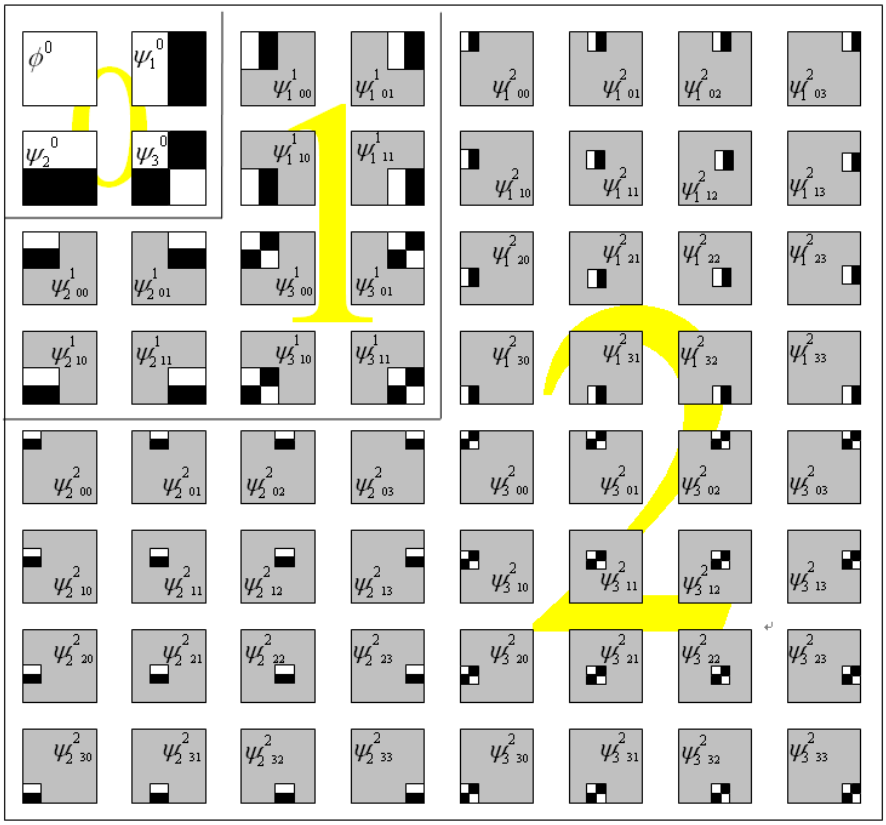


Figure 2.3: Examples of the restricted 2D Haar basis functions (resolution: 8×8). There are three scales of basis functions in the diagrams.

The *support* of a basis function is the non-zero region $\langle j, k, l \rangle$ in which the basis function is defined. All basis functions are categorized into two sets: scaling bases and wavelet bases. In each region $\langle j, k, l \rangle$, four normalized 2D Haar basis functions are defined:

ϕ_{kl}^{j} *normalized Haar scaling basis function*:

$$\phi_{kl}^{j}(x, y) = 2^{j}\phi^{0}(2^{j}x - k, 2^{j}y - l)$$

ϕ^{0} is the mother scaling function.

ψ_{kl}^{j} *normalized Haar wavelet basis function.* There are three types of wavelets defined in one region:

$$\psi_1{}_{kl}^{j} = 2^j \psi_1{}^0(2^j x - k, 2^j y - l)$$

$$\psi_2{}_{kl}^{j} = 2^j \psi_2{}^0(2^j x - k, 2^j y - l)$$

$$\psi_3{}_{kl}^{j} = 2^j \psi_3{}^0(2^j x - k, 2^j y - l)$$

$\psi_1{}^0$, $\psi_2{}^0$ and $\psi_3{}^0$ are three mother wavelets, denoting the horizontal, vertical and diagonal differences.

Mother scaling basis function and mother wavelets are illustrated in Figure 2.2. More examples of Haar basis functions are illustrated in Figure 2.3.

A basis function b_s is said to be the *parent* of another basis function b_t if the support of b_s completely covers the support of b_t, and the scale of b_s is less than the scale of b_t. For simplicity, we define the mother scaling function as the parent of any wavelets. We also define basis functions with scale $j > 0$ as *child basis functions.* It is worth noting that only the mother scaling basis and wavelets are used in the non-standard wavelet transform. Child scaling basis functions are not used in the transform. In the appendix, child scaling basis functions are employed to prove the Haar quad-coefficient theorem.

2.1.2 Haar Quad-Coefficient Theorem

It is observed that many of the Haar quad-coefficients are zero, as illustrated in the first row in Figure 2.4. This observation can be formalized as the Haar Quad-coefficient Theorem.

HAAR QUAD-COEFFICIENT THEOREM *Haar quad-coefficient C_{stuv} has a non-zero value, if and only if for four operand basis functions, one of the following cases hold:*

1. *All four are the mother scaling basis.* $C_{stuv} = 1$.
2. *There are two pairs of identical wavelets, but the first pair may be different from the second pair. All four have identical support at scale* j. $C_{stuv} = 2^{2j}$.
3. *Three are wavelets of different types, but with identical support at scale* j, *and the fourth is a parent basis at scale* $\widetilde{j}$. $C_{stuv} = \pm 2^{j+\widetilde{j}}$.
4. *Two are identical wavelets at scale* j, *the third is a parent at scale* $\widetilde{j}_1$, *and the fourth is another parent at scale* $\widetilde{j}_2$, *where* $\widetilde{j}_1$ *may be different from* $\widetilde{j}_2$. $C_{stuv} = \pm 2^{\widetilde{j}_1+\widetilde{j}_2}$. $\diamondsuit$

In case 2 of the theorem, the sign of the quad-coefficient is that of the sub-region of the parent basis function that the child bases fall into. In case 3, it is the product of the signs of two sub-regions of the parent basis functions that the child bases fall into, respectively. Some examples of the Haar quad-coefficients are illustrated in Figure 2.4.

PROOF Without explicit notations, all basis functions refer to the normalized 2D Haar basis functions. We make following observations on the product of two Haar basis functions:

1. The square of a basis at scale j is the scaling basis of identical support, scaled by 2^j.

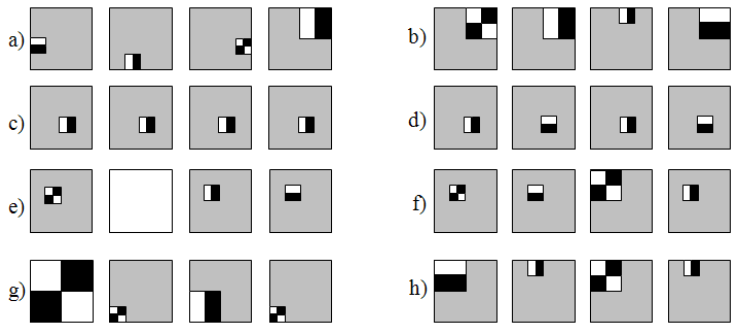


Figure 2.4: Examples of the quad-coefficients of Haar basis functions. The quad-coefficients in the first row are zero. The second/third/fourth row corresponds to case 2/3/4 of the theorem, respectively.

2. The product of a scaling basis at scale j with another basis of identical support is the second basis, scaled by 2^j.

3. The product of two different wavelets of identical support at scale j is the third wavelet of identical support, scaled by 2^j.

4. The product of a child basis with a parent basis at scale $\widetilde{j}$ is the child basis, scaled by $\pm 2^{\widetilde{j}}$. The sign of the product is the sign of the sub-region of the parent basis function that the child basis function falls into.

5. The integral of a wavelet basis vanishes.

6. The integral of a scaling basis at scale j is 2^{-j}.

Since we are only interested in non-zero quad-coefficients, it is clear that in this case the support of four operand basis functions must overlap, and these basis functions can only be

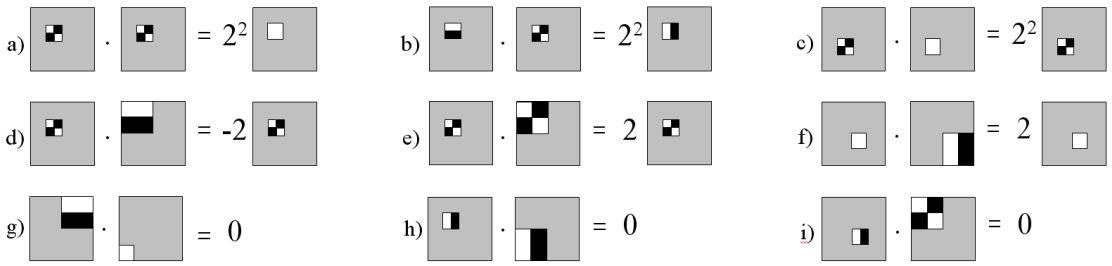


Figure 2.5: Examples of the product of two Haar basis functions. Basis function in the upper row have identical support. In the middle row, the second basis function is the parent of the first one. In the lower row, the product is zero. The magnitude and type of the product is also shown in the figure. White squares in a), c), f) and g) are scaling basis functions.

wavelets or mother scaling function (due to the nonstandard wavelet transform). Nevertheless, the intermediate product of two basis functions may be a *child* scaling basis function.

In the proof, symbol j refers to the scale of the basis at the finest scale, while $\widetilde{j}$, $\widetilde{j}_1$ and $\widetilde{j}_2$ are the scale of parent bases.

I. All four basis functions are wavelets

Case 1: All wavelets at the same scale Since we are only interested in the non-zero quad-coefficient, tt is trivial to show that in this case four wavelets must have identical support. According to the *Pigeonhole Principle*, among them there must be two identical wavelets.

If the other two wavelets are also identical, then three application of observation 1 show that the product of four basis functions is a scaling basis of identical support, scaled by 2^{3j}. Furthermore, observation 6 shows that the quad-coefficient is 2^{2j}.

Otherwise, the other two wavelets must be of different types. Observation 4 shows that their product is a scaled wavelet. Consequently, observation 2 shows that the product of four basis functions is a scaled wavelet, and observation 5 shows that the quad-coefficient is zero.

Therefore, if all wavelets are at the same scale, then there must be two pairs of identical wavelets. This establishes case 2 of the theorem.

Case 2: Three wavelets at the same scale If the fourth wavelet is at a finer scale, then it is the child of the first three wavelets. Three applications of observation 5 show that the product of four wavelets is the scaled wavelet. Observation 5 concludes that the quad-coefficient is zero.

If the fourth wavelet is at a coarser scale, it is trivial to show that first three wavelets must have identical support. There are two possibilities: a) If at least two of the first three wavelets are identical, then observations 1 & 2 show that the product of the first three basis functions is a scaled wavelet. Observation 5 further shows that the product of four bases is a scaled wavelet, and consequently observation 5 concludes that the quad-coefficient is zero. b) Otherwise, first three wavelets are of different types. Observations 4 & 1 show that their product is a scaling basis function of identical support, scaled by 2^{2j}. Observations 5 and 6 conclude that the quad-coefficient is $\pm 2^{j+\tilde{j}}$. This establishes part of case 3 of the theorem.

Case 3: Two pairs of wavelets at the same scale To differentiate from case 1, without loss of generality we assume the scale of the first two bases is smaller than that of the latter two. It is trivial to show that first two wavelets must have identical support, otherwise the quad-coefficient is zero. Observation 1 shows that their product is a scaling basis (scaled by 2^j). Two applications of observation 5 show that the product of four wavelets is also a scaling basis, scaled by $\pm 2^{j+\tilde{j}_1+\tilde{j}_2}$. Observation 6 concludes that the quad-coefficient is $\pm 2^{\tilde{j}_1+\tilde{j}_2}$. This establishes part of case 4 of the theorem.

Case 4: Only two wavelets at the same scale As in case 2, we can show that the other two bases must be at coarser scales, and the two bases at the same scale must be identical. Similar analysis as in case 3 concludes that the quad-coefficient is $\pm 2^{\tilde{j}_1+\tilde{j}_2}$. This establishes part of case 4 of the theorem.

Case 5: All four wavelets at different scales Three applications of observation 5 show that four basis product is a scaled wavelet. Observation 5 gives a zero quad-coefficient.

II. Some bases are the mother scaling basis

Case 1: All four bases are the mother scaling basis The integral coefficient is 1. This establishes case 1 of the theorem.

Case 2: Three bases are the mother scaling basis Because the mother scaling function is defined as the *parent* of any wavelet, three applications of observation 5 show that the product of four bases is a scaled wavelet at the finest scale. Consequently, observation 5 shows that the quad-coefficient is zero.

Case 3: Two bases are the mother scaling basis If the two wavelets are identical, then observations 5 & 1 show that the quad-coefficient is 1; otherwise it is zero. This establishes part of case 4 of the theorem.

Case 4: Only one basis is the mother scaling basis In this case we have three wavelets. Similar analysis as in I shows that if the quad-coefficient is nonzero, then among the three wavelets either two are identical and the third is a parent basis, with quad-coefficient $\pm 2^{\widetilde{j}}$ (part of case 4 of the theorem), or three wavelets have identical support but of different types, with quad-coefficient 2^j (part of case 3 of the theorem).

This concludes the proof of the theorem. □

A special case of the aforementioned theorem is that, one of the four operand basis functions reduces to the mothering scaling basis function. In other words, quad-coefficient is reduced to the third-order integral coefficient. In this case, we have following result:

THIRD-ORDER HAAR INTEGRAL THEOREM *The third-order Haar integral coefficient C_{stu} has a non-zero value, if and only if for three operand basis functions, one of the following cases hold:*

1. *All three are the mother scaling basis. $C_{stuv} = 1$.*
2. *All three are wavelets of different types, but with identical support at scale j. $C_{stuv} = 2^j$.*
3. *Two are identical wavelets at scale j, the third is a parent at scale $\widetilde{j}$. $C_{stuv} = \pm 2^{\widetilde{j}}$.* ♢

This derived result is exactly the Haar tripling coefficient theorem presented in [NRH04].

2.2 Algorithms

In this section, we first present an augmented quad-tree data structure. The tree-structured sub-linear quadruple product wavelet integral algorithm is presented in subsection 2.2.3. In this section we also present a tree-structured triple product wavelet integral algorithm which can be used to render static scenes.

2.2.1 Augmented Quadtree

It has been observed [Sha93] that if a wavelet coefficient at a lower frequency subband is insignificant with respect to a given threshold, then all the coefficients in the same spatial position at higher frequency subbands are very likely to be insignificant with respect to that threshold. This observation leads to a widely used data structure to encode wavelet coefficients, such as the quadtree structure [BBS94, GSC93] in computer graphics and the more complex zero-tree structure [Sha93] in image compression. In our implementation, we adopt this popular data structure with some modifications. Each node of the augmented quad-tree structure is defined as:

struct augnode

α, β, γ : **wavelet coefficients**;

p_{sum} : **signed parent summation**;

$ch[4]$: child pointers;

end

Here, α, β, γ are three wavelet coefficients of different types with identical support. $ch[4]$ point to four immediate child nodes. Field p_{sum} is the key component in our augmented quadtree structure and is detailed in the next subsection. The wavelet tree is defined as:

struct augtree

dc : **mother scaling coefficient**;

$node$: pointer to **struct** augnode;

end

2.2.2 Computation of P_{sum}

The use of field p_{sum} is inspired by Ng et al's work [NRH04] on triple product integral. For any node t in the wavelet tree, field $psum$ stores the summation of coefficients in t's parents, scaled by a constant, which is the magnitude of the product of two basis functions, one in t and another in the parent node. As shown in the following subsections, it is used to quickly accumulate the contribution of parent nodes as we traverse the wavelet trees. In the implementation, field p_{sum} is calculated in the order of tree traversal. The key strategy is to update p_{sum} field from its immediate parent. Particularly, p_{sum} field of the mother wavelets is just the mother scaling coefficient dc. For any node t with support $\langle j, k, l \rangle$ where $j > 0$, assume it lies in the quadrant (q_k, q_l) of its immediate parent node p_t, where

$$q_k = k \bmod 2 \qquad\qquad q_l = l \bmod 2$$

Field $t.p_{sum}$ can be updated as:

$$t._{p_{sum}} = p_{t\cdot p_{sum}} + 2^{j-1} \times [p_{t\cdot\alpha} \times sign(0, q_k, q_l)$$
$$+ p_{t\cdot\beta} \times sign(1, q_k, q_l) + p_{t\cdot\gamma} \times sign(2, q_k, q_l)];$$

where array $sign$ is the sign of four quadrants of the three mother wavelets. It is defined in Figure 2.2 as:

$$sign[3][2][2] = \{1, -1, 1, -1; 1, 1, -1, -1; 1, -1, -1, 1\};$$

2.2.3 Tree-Structured Algorithm for Quadruple Product Integral

The basic intuition for accelerating the quadruple function product integral is to eliminate futile computations involving zero wavelet coefficients and zero quad-coefficients. For this purpose, only significant coefficients are stored in the augmented quad-tree. The entry point of the algorithm is *QuadrupleIntegral*, which takes as input four wavelet trees, each representing the lighting, BRDF, self-occlusion and global-occlusion. From case 1 of the theorem, product integral B is initialized as the direct product of dc components.

algorithm $QuadrupleIntegral$(augtree T_1, T_2, T_3, T_4)

$B = T_{1\cdot dc} \times T_{2\cdot dc} \times T_{3\cdot dc} \times T_{4\cdot dc}$

$Traverse4Trees(T_{1\cdot node}, T_{2\cdot node}, T_{3\cdot node}, T_{4\cdot node})$

end

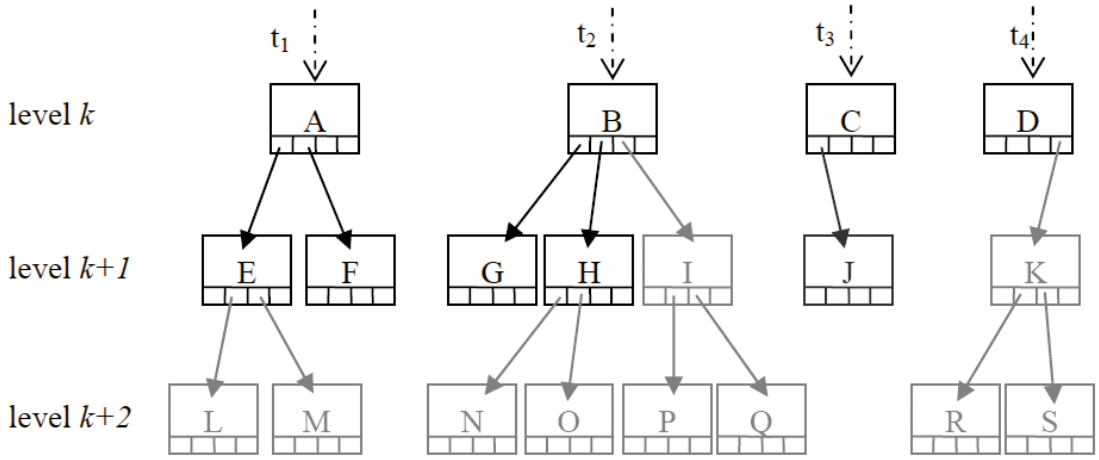


Figure 2.6: Example of four wavelet subtrees. Totally, there are 13 *peer sets*: {ABCD}, {EGJ}, {FH}, {I}, {K}, {L}, {M}, {N}, {O}, {P}, {Q}, {R} and {S}. The tree-structured quadruple product integral algorithm synchronously traverses four wavelet trees. It deliberately eliminates futile computations involving zero quad-coefficients, such as ten gray nodes which have no peers.

In routine $Traverse4Trees$, four wavelet trees are traversed synchronously, while futile computations involving zero quad-coefficients are automatically eliminated. As shown in Figure 2.6, we define the set of nodes in different wavelet trees with identical support as a *peer set*. Nodes in the same peer set are referred to as *peers*. From the quad-coefficient theorem, we observed that nodes without any peers must have zero quad-coefficients. Therefore, they can be safely eliminated (together with their children nodes) to accelerate computation. In Figure 2.6, ten gray nodes without any peers are deliberately eliminated by the traversal algorithm.

We identified that peer sets with four nodes relate to cases 2, 3 and 4 of the theorem. While those with only three nodes relate to cases 3 and 4, and those with only two nodes relate to case 4. Based on these observations, we designed following algorithm:

algorithm $Traverse4Trees$(augnode t_1, t_2, t_3, t_4)

if none of t_1, t_2, t_3, t_4 is null

$$B \mathrel{+}= case_2(t_1, t_2, t_3, t_4) + case_3(t_1, t_2, t_3, t_4)$$
$$+case_3(t_2, t_1, t_3, t_4) + case_3(t_3, t_1, t_2, t_4) + case_3(t_4, t_1, t_2, t_3)$$
$$+case_4(t_1, t_2, t_3, t_4) + case_4(t_1, t_3, t_2, t_4) + case_4(t_1, t_4, t_2, t_3)$$
$$+case_4(t_2, t_3, t_1, t_4) + case_4(t_2, t_4, t_1, t_3) + case_4(t_3, t_4, t_1, t_2)$$

elseif only one of t_1, t_2, t_3, t_4 is null, say, t_1 is null

$$B \mathrel{+}= case_3(t_1, t_2, t_3, t_4) + case_4(t_1, t_2, t_3, t_4)$$
$$+case_4(t_1, t_3, t_2, t_4) + case_4(t_1, t_4, t_2, t_3);$$

elseif only two of t_1, t_2, t_3, t_4 is null, say, t_1 and t_2 are null

$$B \mathrel{+}= case_4(t_1, t_2, t_3, t_4);$$

else

return;

for $i = 0$ to 3

$$Traverse4Trees(t_1.ch[i], t_2.ch[i], t_3.ch[i], t_4.ch[i]);$$

end

As shown in the case 2 of the theorem, support routine $case_2$ sums up 21 permutations of the wavelet coefficients.

routine $Case2$(augnode t_1, t_2, t_3, t_4)

return $4^{t_1 \to scale} \times$

$$(t_{1\cdot\alpha} \times t_{2\cdot\alpha} \times t_{3\cdot\alpha} \times t_{4\cdot\alpha} + t_{1\cdot\alpha} \times t_{2\cdot\alpha} \times t_{3\cdot\beta} \times t_{4\cdot\beta} + t_{1\cdot\alpha} \times t_{2\cdot\alpha} \times t_{3\cdot\gamma} \times t_{4\cdot\gamma}$$

$$+t_{1\cdot\beta} \times t_{2\cdot\beta} \times t_{3\cdot\alpha} \times t_{4\cdot\alpha} + t_{1\cdot\beta} \times t_{2\cdot\beta} \times t_{3\cdot\beta} \times t_{4\cdot\beta} + t_{1\cdot\beta} \times t_{2\cdot\beta} \times t_{3\cdot\gamma} \times t_{4\cdot\gamma}$$

$$+t_{1\cdot\gamma} \times t_{2\cdot\gamma} \times t_{3\cdot\alpha} \times t_{4\cdot\alpha} + t_{1\cdot\gamma} \times t_{2\cdot\gamma} \times t_{3\cdot\beta} \times t_{4\cdot\beta} + t_{1\cdot\gamma} \times t_{2\cdot\gamma} \times t_{3\cdot\gamma} \times t_{4\cdot\gamma}$$

$$+t_{1\cdot\alpha} \times t_{3\cdot\alpha} \times t_{2\cdot\beta} \times t_{4\cdot\beta} + t_{1\cdot\alpha} \times t_{3\cdot\alpha} \times t_{2\cdot\gamma} \times t_{4\cdot\gamma} + t_{1\cdot\beta} \times t_{3\cdot\beta} \times t_{2\cdot\alpha} \times t_{4\cdot\alpha}$$

$$+t_{1\cdot\beta} \times t_{3\cdot\beta} \times t_{2\cdot\gamma} \times t_{4\cdot\gamma} + t_{1\cdot\gamma} \times t_{3\cdot\gamma} \times t_{2\cdot\alpha} \times t_{4\cdot\alpha} + t_{1\cdot\gamma} \times t_{3\cdot\gamma} \times t_{2\cdot\beta} \times t_{4\cdot\beta}$$

$$+t_{1\cdot\alpha} \times t_{4\cdot\alpha} \times t_{2\cdot\beta} \times t_{3\cdot\beta} + t_{1\cdot\alpha} \times t_{4\cdot\alpha} \times t_{2\cdot\gamma} \times t_{3\cdot\gamma} + t_{1\cdot\beta} \times t_{4\cdot\beta} \times t_{2\cdot\alpha} \times t_{3\cdot\alpha}$$

$$+t_{1\cdot\beta} \times t_{4\cdot\beta} \times t_{2\cdot\gamma} \times t_{3\cdot\gamma} + t_{1\cdot\gamma} \times t_{4\cdot\gamma} \times t_{2\cdot\alpha} \times t_{3\cdot\alpha} + t_{1\cdot\gamma} \times t_{4\cdot\gamma} \times t_{2\cdot\beta} \times t_{3\cdot\beta})\ ;$$

Routine $case_3$ corresponds to the case 3 of the theorem. It sums up six permutations.

routine $case_3$(augnode t_1, t_2, t_3, t_4)

return $2^{t_1 \to scale} \times (t_1 \to p_{sum}) \times$

$$(t_{2\cdot\alpha} \times t_{3\cdot\beta} \times t_{4\cdot\gamma}\ +\ t_{2\cdot\alpha} \times t_{3\cdot\gamma} \times t_{4\cdot\beta}\ +\ t_{2\cdot\beta} \times t_{3\cdot\alpha} \times t_{4\cdot\gamma}$$

$$+\ t_{2\cdot\beta} \times t_{3\cdot\gamma} \times t_{4\cdot\alpha}\ +\ t_{2\cdot\gamma} \times t_{3\cdot\alpha} \times t_{4\cdot\beta}\ +\ t_{2\cdot\gamma} \times t_{3\cdot\beta} \times t_{4\cdot\alpha});$$

Here, t_1 may be *null*. In this case, $t_1 \to p_{sum}$ can be instantiated from the p_{sum} field of t_1's closest *physical* parent node, which can be passed as a parameter during the traversal. Routine $case_4$ is trivially converted from the last case of the theorem.

routine $case_4$(augnode t_1, t_2, t_3, t_4)

return $(t_1 \to p_{sum}) \times (t_2 \to p_{sum}) \times (t_{3\cdot\alpha} \times t_{4\cdot\alpha} + t_{3\cdot\beta} \times t_{4\cdot\beta} + t_{3\cdot\gamma} \times t_{4\cdot\gamma})$;

Considering the tree-traversal nature of the quadruple product integral algorithm, the time complexity of the algorithm is $O(m)$, where m is the number of quad-nodes encoded in the wavelet trees.

2.2.4 Tree-Structured Algorithm for Triple Product Integral

The degeneration of the previous algorithm is an algorithm for triple product wavelet integral. The modification to the aforementioned tree-structured quadruple product integral algorithm is quite trivial, just simply eliminating one wavelet tree, say, T_4.

algorithm $TripleIntegral$(augtree T_1, T_2, T_3)

$B = T_{1 \cdot dc} \times T_{2 \cdot dc} \times T_{3 \cdot dc}$

$Traverse3Trees(T_{1 \cdot node}, T_{2 \cdot node}, T_{3 \cdot node})$

end

algorithm $Traverse3Trees$(augnode t_1, t_2, t_3)

if none of t_1, t_2, t_3 is null

$B \mathrel{+}= case_{3t}(t_1, t_2, t_3) + case_{4t}(t_1, t_2, t_3)$

$+case_{4t}(t_2, t_1, t_3) + case_{4t}(t_3, t_1, t_2);$

elseif only one of t_1, t_2, t_3 is null, say, t_1 is null

$B \mathrel{+}= case_{4t}(t_1, t_2, t_3);$

else

return;

for $i = 0$ to 3

$Traverse3Trees(t_1.ch[i], t_2.ch[i], t_3.ch[i])$;

end

Here, routines $case_{3t}$ and $case_{4t}$ are derived from the last two cases of the third-order Haar integral coefficient theorem.

routine $case_{3t}$(augnode t_1, t_2, t_3)

return $2^{t_1 \rightarrow scale} \times$

$(t_{1 \cdot \alpha} \times t_{2 \cdot \beta} \times t_{3 \cdot \gamma} \ + \ t_{1 \cdot \alpha} \times t_{2 \cdot \gamma} \times t_{3 \cdot \beta} \ + \ t_{1 \cdot \beta} \times t_{2 \cdot \alpha} \times t_{3 \cdot \gamma}$

$+ \ t_{2 \cdot \beta} \times t_{2 \cdot \gamma} \times t_{3 \cdot \alpha} \ + \ t_{1 \cdot \gamma} \times t_{2 \cdot \alpha} \times t_{3 \cdot \beta} \ + \ t_{1 \cdot \gamma} \times t_{2 \cdot \beta} \times t_{3 \cdot \alpha})$;

routine $case_{4t}$(augnode t_1, t_2, t_3)

return $(t_1 \rightarrow p_{sum}) \times (t_{2 \cdot \alpha} \times t_{3 \cdot \alpha} + t_{2 \cdot \beta} \times t_{3 \cdot \beta} + t_{2 \cdot \gamma} \times t_{3 \cdot \gamma})$;

The time complexity of the algorithm is $O(m)$, where m is the number of quad-nodes encoded in the wavelet trees.

2.2.5 Tree-Structured Algorithm for Three-Function Product

Following algorithm evaluates the product of three functions. The input to the algorithm *ThreeFunctionProduct* are the root node of three quad-trees. The product function is encoded the quad-tree.

Global augtree T_0, T_1, T_2, T_3;

algorithm ThreeFunctionProduct

$T_{0 \cdot dc} = T_{1 \cdot dc} \times T_{2 \cdot dc} \times T_{3 \cdot dc}$

Traverse3plus1Trees($T_{0 \cdot node}, T_{1 \cdot node}, T_{2 \cdot node}, T_{3 \cdot node}$);

end

algorithm Traverse3plus1Trees (augnode t_0, t_1, t_2, t_3)

if none of t_1, t_2, t_3 is null

$kase_2(t_0, t_1, t_2, t_3)$; $kase_3(t_0, t_1, t_2, t_3)$; $kase_3(t_0, t_2, t_1, t_3)$;

$kase_3(t_0, t_3, t_1, t_2)$; $kase_{3P}(t_0, t_1, t_2, t_3)$; $kase_4(t_0, t_2, t_3, t_1)$;

$kase_4(t_0, t_1, t_3, t_2)$; $kase_4(t_0, t_1, t_2, t_3)$; $kase_{4P}(t_0, t_1, t_2, t_3)$;

$kase_{4P}(t_0, t_2, t_1, t_3)$; $kase_{4P}(t_0, t_3, t_1, t_2)$;

elseif only one of t_1, t_2, t_3, say, t_1 is null

$kase_3(t_0, t_1, t_2, t_3)$; $kase_4(t_0, t_1, t_2, t_3)$; $kase_4(t_0, t_1, t_3, t_2)$;

$kase_{4P}(t_0, t_1, t_2, t_3)$;

elseif only two of t_1, t_2, t_3, say, $t_1$and t_2 are null

$kase_4(t_0, t_1, t_2, t_3)$;

else

return;

for $i = 0$ to 3

Traverse3plus1Trees($t_0.ch[i], t_1.ch[i], t_2.ch[i], t_3.ch[i]$);

end

Support routines $kase_2, kase_3, kase_{3P}, kase_{4P}$ are derived from the theorem. $kase_2$ corresponds to seven permutations.

routine $kase_2$ (augnode t_0, t_1, t_2, t_3)

$$t_{0\cdot\alpha} \mathrel{+}= 4^{t_{1\cdot scale}} \times (t_{1\cdot\alpha} \times t_{2\cdot\alpha} \times t_{3\cdot\alpha} + t_{1\cdot\alpha} \times t_{2\cdot\beta} \times t_{3\cdot\beta} + t_{1\cdot\alpha} \times t_{2\cdot\gamma} \times t_{3\cdot\gamma}$$
$$+ t_{1\cdot\beta} \times t_{2\cdot\alpha} \times t_{3\cdot\beta} + t_{1\cdot\gamma} \times t_{2\cdot\alpha} \times t_{3\cdot\gamma} + t_{1\cdot\beta} \times t_{2\cdot\beta} \times t_{3\cdot\alpha} + t_{1\cdot\gamma} \times t_{2\cdot\gamma} \times t_{3\cdot\alpha});$$

$$t_{0\cdot\beta} \mathrel{+}= 4^{t_{1\cdot scale}} \times (t_{1\cdot\beta} \times t_{2\cdot\alpha} \times t_{3\cdot\alpha} + t_{1\cdot\beta} \times t_{2\cdot\beta} \times t_{3\cdot\beta} + t_{1\cdot\beta} \times t_{2\cdot\gamma} \times t_{3\cdot\gamma}$$
$$+ t_{1\cdot\alpha} \times t_{2\cdot\beta} \times t_{3\cdot\alpha} + t_{1\cdot\gamma} \times t_{2\cdot\beta} \times t_{3\cdot\gamma} + t_{1\cdot\alpha} \times t_{2\cdot\alpha} \times t_{3\cdot\beta} + t_{1\cdot\gamma} \times t_{2\cdot\gamma} \times t_{3\cdot\beta});$$

$$t_{0\cdot\gamma} \mathrel{+}= 4^{t_{1\cdot scale}} \times (t_{1\cdot\gamma} \times t_{2\cdot\alpha} \times t_{3\cdot\alpha} + t_{1\cdot\gamma} \times t_{2\cdot\beta} \times t_{3\cdot\beta} + t_{1\cdot\gamma} \times t_{2\cdot\gamma} \times t_{3\cdot\gamma}$$
$$+ t_{1\cdot\alpha} \times t_{2\cdot\gamma} \times t_{3\cdot\alpha} + t_{1\cdot\beta} \times t_{2\cdot\gamma} \times t_{3\cdot\beta} + t_{1\cdot\alpha} \times t_{2\cdot\alpha} \times t_{3\cdot\gamma} + t_{1\cdot\beta} \times t_{2\cdot\beta} \times t_{3\cdot\gamma});$$

routine $kase_3$ (augnode t_0, t_1, t_2, t_3)

$$t_{0\cdot\alpha} \mathrel{+}= 2^{t_{1\cdot scale}} \times t_{1\cdot p_{sum}} \times (t_{2\cdot\beta} \times t_{3\cdot\gamma} + t_{2\cdot\gamma} \times t_{3\cdot\beta});$$

$$t_{0\cdot\beta} \mathrel{+}= 2^{t_{1\cdot scale}} \times t_{1\cdot p_{sum}} \times (t_{2\cdot\alpha} \times t_{3\cdot\gamma} + t_{2\cdot\gamma} \times t_{3}.\alpha);$$

$$t_{0\cdot\gamma} \mathrel{+}= 2^{t_{1\cdot scale}} \times t_{1\cdot p_{sum}} \times (t_{2\cdot\alpha} \times t_{3\cdot\beta} + t_{2\cdot\beta} \times t_{3}.\alpha);$$

routine $kase_{3P}$ (augnode t_0, t_1, t_2, t_3)

$$val = 2^{t_{1\cdot scale}} \times (t_{1\cdot\alpha} \times t_{2\cdot\beta} \times t_{3\cdot\gamma} + t_{1\cdot\alpha} \times t_{2\cdot\gamma} \times t_{3\cdot\beta} + t_{1\cdot\beta} \times t_{2\cdot\alpha} \times t_{3\cdot\gamma}$$
$$+ t_{1\cdot\beta} \times t_{2\cdot\gamma} \times t_{3\cdot\alpha} + t_{1\cdot\gamma} \times t_{2\cdot\alpha} \times t_{3\cdot\beta} + t_{1\cdot\gamma} \times t_{2\cdot\beta} \times t_{3}.\alpha);$$

UpdateParents(t_0, val);

routine $kase_4$ (augnode t_0, t_1, t_2, t_3)

$$t_{0\cdot\alpha} \mathrel{+}= t_{1\cdot p_{sum}} \times t_{2\cdot p_{sum}} \times t_{3}.\alpha;$$

$$t_{0\cdot\beta} \mathrel{+}= t_{1\cdot p_{sum}} \times t_{2\cdot p_{sum}} \times t_{3\cdot\beta};$$

$$t_{0\cdot\gamma} \mathrel{+}= t_{1\cdot p_{sum}} \times t_{2\cdot p_{sum}} \times t_{3\cdot\gamma};$$

routine $kase_{4P}$ (augnode t_0, t_1, t_2, t_3)

$val = \ t_{1 \cdot p_{sum}} \times (t_{2 \cdot \alpha} \times t_{3 \cdot \alpha} + t_{2 \cdot \beta} \times t_{3 \cdot \beta} + t_{2 \cdot \gamma} \times t_{3 \cdot \gamma})$;

UpdateParents(t_0, val);

routine UpdateParents (augnode t_0, val)

$t_0 \rightarrow dc \mathrel{+}= \ val$;

for scale $j = 0, \cdots, t_0.scale - 1$

// t_0 lies in the quadrant (k, l) of its parent t_p at scale j

$t_{p \cdot \alpha} \mathrel{+}= \ sign(0, k, l) \times 2^j \times val$;

$t_{p \cdot \beta} \mathrel{+}= \ sign(1, k, l) \times 2^j \times val$;

$t_{p \cdot \gamma} \mathrel{+}= \ sign(2, k, l) \times 2^j \times val$;

The time complexity of this algorithm is inherently $O(M \log N)$, where M is the number of nonzero coefficients used in the nonlinear approximation, and N is the resolution of the 2D signal.

2.2.6 Tree-Structured Algorithm for Two-Function Product

Following algorithm evaluates the product of three functions. The input to the algorithm *ThreeFunctionProduct* are the root node of three quad-trees. The product function is encoded the quad-tree.

Global augtree T_0, T_1, T_2;

algorithm TwoFunctionProduct

$T_{0 \cdot dc} = T_{1 \cdot dc} \times T_{2 \cdot dc}$

Traverse2plus1Trees($T_{0 \cdot node}, T_{1 \cdot node}, T_{2 \cdot node}$);

end

algorithm Traverse2plus1Trees (augnode t_0, t_1, t_2)

if none of t_1, t_2 is null

$kkase_2(t_0, t_1, t_2)$; $kkase_2(t_0, t_2, t_1)$; $kkase_3(t_0, t_1, t_2)$; $kkase_{3P}(t_0, t_1, t_2)$;

elseif only one of t_1, t_2, say, t_1 is null

$kkase_3(t_0, t_1, t_2)$;

else

return;

for $i = 0$ to 3

Traverse2plus1Trees($t_0.ch[i], t_1.ch[i], t_2.ch[i]$);

end

Support routines $kkase_2, kkase_3, kkase_{3P}$ are derived from the third-order Haar integral theorem. Support routine *UpdateParents* is given in the previous subsection.

routine $kkase_2$ (augnode t_0, t_1, t_2)

$t_{0 \cdot \alpha}$ += $2^{t_{0 \cdot scale}} \times (t_{1 \cdot \beta} \times t_{2 \cdot \gamma} + t_{2 \cdot \gamma} \times t_{1 \cdot \beta})$;

$t_{0 \cdot \beta}$ += $2^{t_{0 \cdot scale}} \times (t_{1 \cdot \alpha} \times t_{2 \cdot \gamma} + t_{2 \cdot \gamma} \times t_1.\alpha)$;

$t_{0 \cdot \gamma}$ += $2^{t_{0 \cdot scale}} \times (t_{1 \cdot \alpha} \times t_{2 \cdot \beta} + t_{2 \cdot \beta} \times t_1.\alpha)$;

routine $kkase_3$ (augnode t_0, t_1, t_2)

$t_{0\cdot\alpha}$ += $t_{1\cdot p_{sum}} \times t_2.\alpha$;

$t_{0\cdot\beta}$ += $t_{1\cdot p_{sum}} \times t_{2\cdot\beta}$;

$t_{0\cdot\gamma}$ += $t_{1\cdot p_{sum}} \times t_{2\cdot\gamma}$;

routine $kkase_{3P}$ (augnode t_0, t_1, t_2)

$val = t_{1\cdot\alpha} \times t_{2\cdot\alpha} + t_{1\cdot\beta} \times t_{2\cdot\beta} + t_{1\cdot\gamma} \times t_{2\cdot\gamma}$;

UpdateParents(t_0, val);

The time complexity of this algorithm is inherently $O(M \log N)$, where M is the number of nonzero coefficients used in the nonlinear approximation, and N is the resolution of the 2D signal.

2.2.7 Numerical Comparisons

In this subsection, we compare several approaches to computing quadruple function product integral (equation 3.3). We implemented four algorithms. The first algorithm is the tree-structured quadruple wavelet product integral algorithm shown in subsection 2.2.3; the second algorithm performs pixel domain product integral; the third algorithm performs pixel domain product integral with wavelet decomposition; the fourth algorithm implements a re-

cursive two-function Haar tripling integral algorithm. Note that quadruple function product integral (equation 3.3) can also be evaluated recursively as:

$$B = \int [[F_1(\varphi) \cdot F_2(\varphi)] \cdot F_3(\varphi)] F_4(\varphi) d\varphi$$

Therefore, quadruple function product integral reduces to two applications of two-function product, followed by a double function product integral. Ng et al. [NRH04] show that two-function product, i.e., $G = F_1 \times F_2$, can be approximated using the tripling integral coefficients. In other words, the k^{th} $(1 \leq k \leq M)$ basis coefficient g_k of G can be computed as:

$$g_k = \langle G, b_k \rangle = \langle F_1 \times F_2, b_k \rangle = \sum_i \sum_j f_{1i} \; f_{2j} \; C_{ijk}$$

where C_{ijk} is the tripling integral coefficient. In the experiment, the fourth algorithm uses an optimized two-function product algorithm which is derived from the tripling coefficient theorem [NRH04].

Figure 2.7 illustrates the comparison of the running time of four approaches for one representative set of cubemaps . Each cubemap is pre-computed at resolution $6 \times 256 \times 256$. As shown in the figure, for reasonable error, quadruple Haar product integral algorithm is orders of magnitude faster than the pixel domain product integral and the recursive two-function Haar product tripling integral. Note that due to the quad-tree encoding, the quadruple Haar product integral algorithm practically demonstrates a sub-linear time complexity.

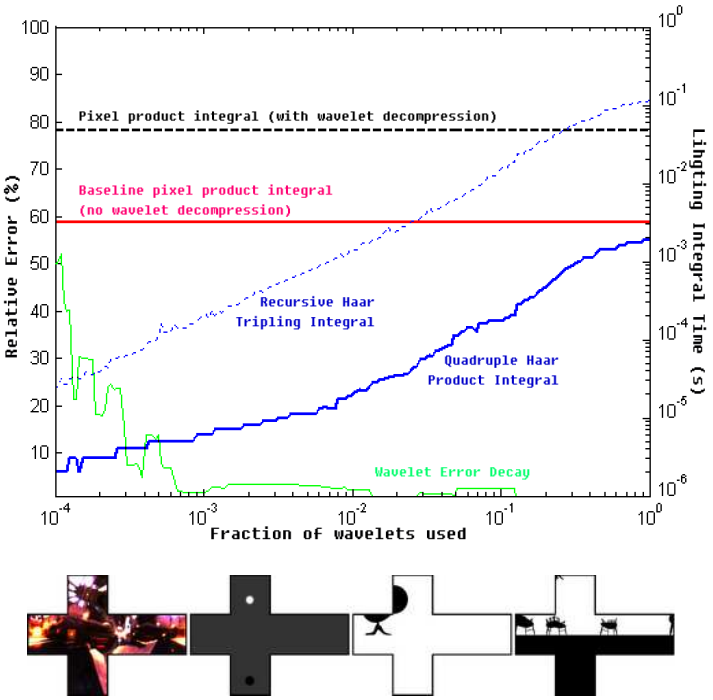


Figure 2.7: Numerical comparison of the running time of four approaches to the quadruple function product integral for one representative set of cubemaps (resolution: $6 \times 256 \times 256$). Note that with reasonable error, the quadruple Haar product integral is orders of magnitude faster than the recursive two-function Haar tripling integral approach and the pixel domain product integral.

2.3 Implementation

In this section, we present the pre-computation and rendering details. All experiments are conducted on a desktop computer with a single Intel Pentium4 3.2GHz CPU. The maximum memory used by the system is nearly 1.6GB.

2.3.1 Pre-Computation

Visibility: For each scene entity, we sample visibility field at its surface point (termed *local visibility*) and in the important surrounding regions (termed *global visibility*). We use two sampling schemes: *planar sampling* and *spherical sampling*. In the planar sampling scheme, we densely sample visibilities in forms of concentric circles on each entity's *virtual* ground plane, which is the lower plane of its bounding box. The center of concentric circles is the projection of the object center on the ground plane. In the spherical sampling scheme, we sample the surrounding region around each scene entity in forms of concentric spheres, which is similar to the object occlusion field [ZHL05], but the center of concentric spheres coincides with the center of concentric circles. In the implementation, for planar sampling scheme, we use up to 100 concentric circles, each being sampled at up to 200 points. The radius of these concentric circles ranges from $0.05r$ to $10r$, where r is the radius of projection of the instance's center on the virtual ground plane. For spherical sampling scheme, we use 20 concentric spheres, ranging from $0.2R$ to $6R$, where R is the radius of the bounding sphere. Each concentric sphere is sampled at $6 \times 9 \times 9$. Since we densely sample visibilities on the ground plane, we are able to account for intricate cast shadows on that plane. The difference of the radius of the neighboring concentric circles/spheres in planar and spherical sampling schemes increases linearly with increasing distance to the sampling center.

As shown in Figure 2.1, the table scene is pre-computed as two distinct objects, the table and the chairs. Note that the ground is fused with the chairs. As a result, although we can freely

Table 2.1: Pre-computation statistics for quadruple wavelet product integral

Entity	Vertices		Sampling Points		Raw	Num. of
	Raw	Fine	Planar	Spherical	Data	Quadnodes
Table	4K	25K	20K	10K	1.3G	9M
Chairs	15K	152K	1K	10K	3.9G	20M

manipulate the table, we cannot move any chair in the scene. Support dynamically moving any single object in the scene is beyond the scope of this chapter. The pre-computation statistic for the scene is shown in the following table. The scene in Figure 3.5 is pre-computed as two objects, the chair and the ground. As shown in Figure 3.5, the chair can be rigidly translated in the scene.

At each sampling point, visibility is pre-computed as a cubemap by rasterizing the coarse model using graphics hardware, with standard super-sampling (up to 8 samples per pixel) enabled. Each cubemap is rasterized at resolution $6 \times 64 \times 64$. The pre-computation takes less than 40 minutes.

BRDF: Pre-computing BRDF is quite similar to previous work [NRH04]. In the implementation, Phong BRDFs (shininess: up to 200) are sampled with resolution in $\theta_r \times \varphi$ of up to $(6 \times 64 \times 64) \times (6 \times 64 \times 64)$, where θ_r is the reflection vector of view direction θ about normal N.

Compression and Representation: We project pre-computed visibility and BRDF onto Haar bases using nonstandard wavelet transform, then perform nonlinear approximation

to discard insignificant coefficients less than a certain threshold. Practically, for each face of the cubemap, keeping $60 - 120$ significant coefficients is sufficient to render images close to the reference images. The resulting data are encoded in the augmented quad-tree structure.

2.3.2 Rendering

In the experiments, we sample the high dynamic range illumination [DM97] at resolution $6 \times 64 \times 64$. Note that it is not necessary to sample the illumination at a higher resolution than that of the BRDF and visibility. In the system, lighting is dynamically sampled at runtime, subject to wavelet transform, non-linear approximation and quadtree encoding. Compared to visibility and BRDF, more wavelet coefficients are required to approximate all-frequency lighting (up to 300 coefficients per face of the cubemap).

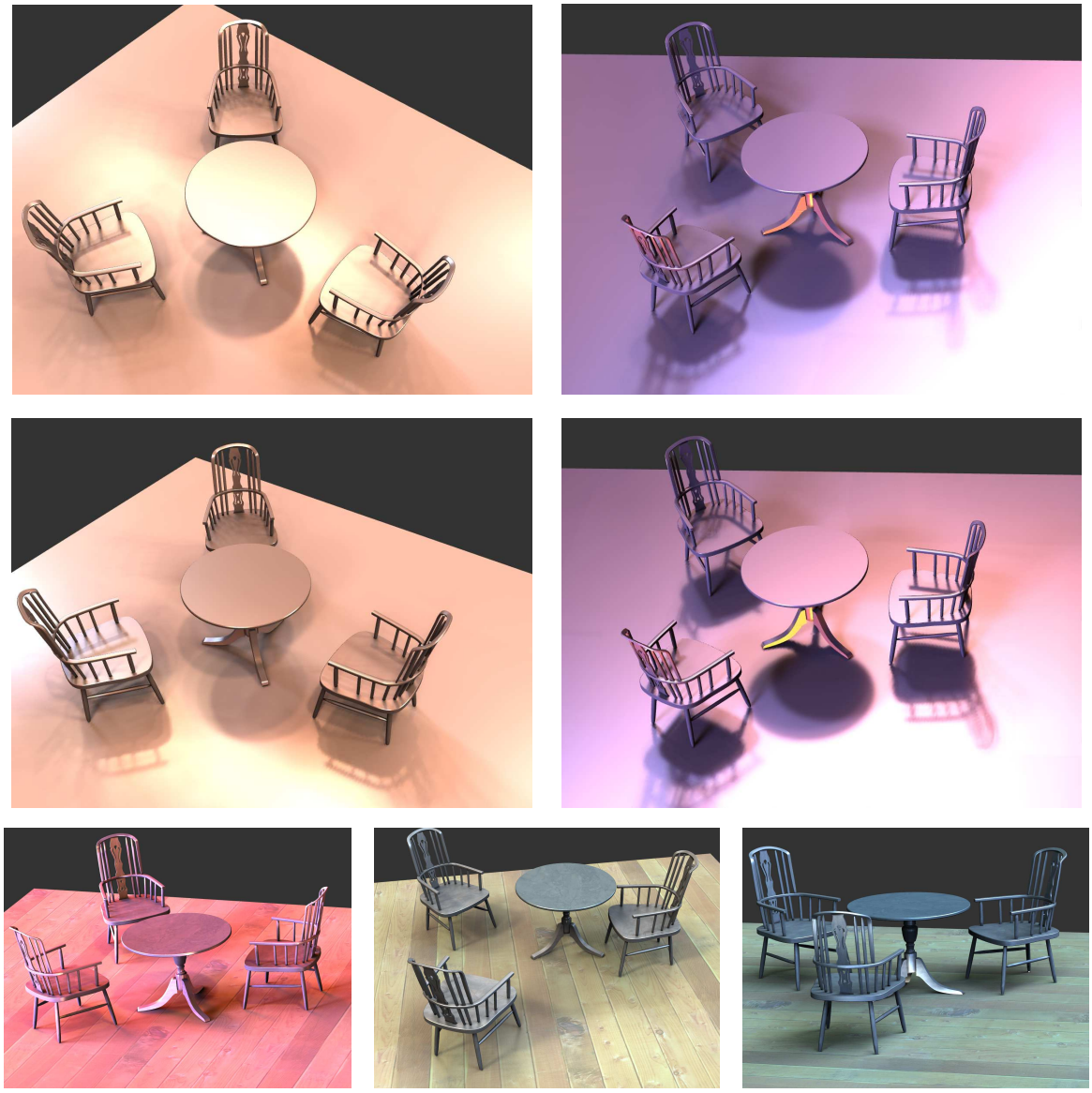


Figure 2.8: Examples of the composed scenes (resolution: 1200×900). Images are rendered in less than ten seconds per frame. In the upper left column, view conditions are slightly changed. The scene is illuminated as inside St. Peter's Basilica. Note that cast shadows and specular highlight change dramatically with changing viewpoint. In the upper right column images are illuminated as inside Grace Cathedral, with lighting being slightly rotated. For these images texture is omitted to enhance shadow details. The lower row shows textured images.

We implement the tree-structured rendering algorithm (section 2.2.3). For each vertex, the algorithm iterates six times to account for each face of the cubemap. Environment lighting can be rotated and sampled at runtime. Note that self-occlusion is readily available. BRDF is interpolated from the tabulated data. Global occlusions are interpolated from tabulated visibility field. If the vertex is on the virtual ground plane of the occluder, it is bi-linearly interpolated from four neighboring points in the occluder's planar sampling field; otherwise it is tri-linearly interpolated from eight neighboring points in the occluder's spherical sampling field. Each color channel is processed independently. To accelerate computation, we only render visible vertices in the scene. As shown in Figure 2.1 and Figure 2.8, the rendering system supports dynamically moving object 'table' at runtime. These images are rendered in less than ten seconds per frame.

One nice feature of the tree-structured rendering algorithm is that rendering speed can be controlled interactively by changing the traversal depth. Figure 3.5 compares the effect of varying traversal depth on rendered images. Smaller depth results in faster rendering, but band-limits high-frequency shadows and view-dependent specularities. As traversal depth increases, more wavelet coefficients are involved, and consequently we get more shadow details at the expense of longer computation time. As shown in the figure, the overall rendering time (including encoding lighting, interpolating BRDF and occlusion, shading integral and rasterization) demonstrates a linear complexity in terms of the traversal depth of the rendering algorithm.

2.4 Conclusions

We have extended double/triple product integral to *quadruple product integral* with a *novel Haar quad-coefficient theorem*. We also formulated a sublinear tree-structured quadruple product integral algorithm. The degeneration of the algorithm is a natural yet efficient triple product integral algorithm. We demonstrated interactive rendering of complex dynamic glossy objects, under time-variant lighting and dynamic viewpoint simultaneously.

This chapter is a step forward in evaluating the product integral of multiple signals in an efficient fashion, which is of great importance to general numerical analysis and signal processing. Currently the rendering algorithm works on a general CPU. It has potential to be optimized to take advantage of the powerful GPU features to achieve better timing performance. Currently our system can only handle rigidly moving single object (such as translation) in the scene. Future work includes supporting objects with non-rigid movement.

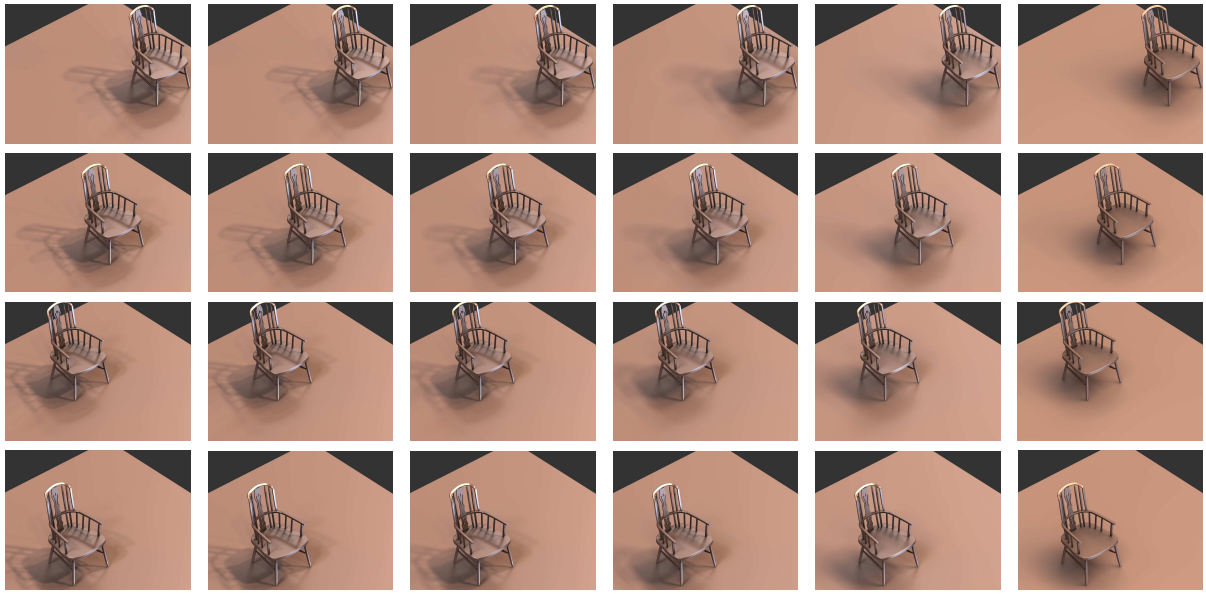


Figure 2.9: Comparison of the rendered images with varying traversal depth of the rendering algorithm. As shown in the figure, quadruple product integral inherently supports dynamically moving chair in the scene. The overall rendering time (including sampling/encoding the lighting, interpolating BRDF and occlusion, shading integral and rasterization) is also shown (in seconds), which demonstrates a linear complexity in terms of the traversal depth of the rendering algorithm. Images in a) is indistinguishable from the reference. The maximum number of wavelets encoded in the augmented quadtrees for each face of the cubemap are: lighting (St. Peter's Basilica) 200, BRDF 70, visibility 70.

CHAPTER 3

GENERALIZED WAVELET PRODUCT INTEGRAL

This chapter appears in ACM SIGGRAPH 2006 conference proceedings, titled as "Generalized Wavelet Product Integral for Rendering Dynamic Glossy Objects" [SM06a]. Under the request of the papers committee, a short technical sketch [SM06b] was also presented in the conference.

3.1 Introduction

Real-time rendering dynamic objects with realistic materials and view-dependent all-frequency intricate shadows is of significant importance for a variety of applications in the field of computer graphics. Computer games, 3D geometry modelling and lighting design are typical examples of such applications. Conventional global illumination techniques [DBB03], such as ray tracing, photon mapping and radiosity, can take hours to generate photorealistic imageries, that limits their application in interactive industrial design. A common practice to accelerate global illumination is to retain only the direct lighting, for instance, environment

Figure 3.1: Examples of the composed scenes using our interactive lighting design system. All images are rendered in real-time using JRT technique (resolution: 1200×900).

mapping [BN76]. Recently, Sloan et al. [SKS02] proposed Pre-computed Radiance Transfer (PRT) to extend environment mapping technique for real-time rendering of soft shadows in a static scene. PRT supports diffuse and low-glossy materials, and has been extended to support all-frequency lighting [NRH03], high-glossy materials [LSS04, NRH04, WTL04], and dynamic objects with diffuse and low-glossy materials [JF03, KLA04, KL05, SLS05, ZHL05]

. However, interactively rendering dynamic high-glossy objects with realistic all-frequency shadows still remains a challenging problem.

Our goal is to design an interactive lighting design system, providing *real-time feedback* with realistic all-frequency shadows for dynamic glossy objects. Previous work [NSD94, DAG95, KAJ05, PVL05] relight static scenes with time-variant lighting. Our system supports interactive manipulation of objects (such as cloning, translating and scaling) with all-frequency shadows. The system also allows the designer to adjust environment lighting and change view conditions. The system captures all-frequency view-dependent lighting effects, such as intricate cast shadows from neighboring objects and specular highlights, as shown in Figure 3.1. Our approach is based on the pre-computed information of individual scene entities, and is not limited to the pre-animated models. The approach is fast and flexible. We believe that it has broad applications in industrial lighting design where both high quality rendering and interactivity are of great importance. It is also applicable to computer games where real-time response is a must.

3.1.1 Contributions

The main contributions of the chapter are:

Representation/Algorithm for Multi-function Product Integral: We explicitly incorporate dynamic occlusions into the shading integral to account for cast shadows from

neighboring objects. At each vertex, shading is formulated as the product integral of multiple functions, involving the lighting, BRDF, local visibility and global occlusions. We show that multi-function product integral in the primal corresponds to the summation of the product of integral coefficients and basis coefficients. To our knowledge, there is no significant previous work on efficiently integrating the product of n functions ($n > 3$). In the paper, we propose a novel *generalized Haar integral coefficient theorem* to evaluate arbitrary Haar integral coefficients. With a novel sub-linear algorithm for integrating the product of multiple functions, we show that dynamic glossy objects can be rendered under time-variant all-frequency lighting and changing view conditions in a few seconds on a commodity CPU, which is orders of magnitude faster than previous techniques.

Just-in-time Radiance Transfer (JRT): To further accelerate shadow computation, we present a novel Just-in-time Radiance Transfer technique. We demonstrate JRT technique in the interactive lighting design system to provide realistic all-frequency shadows in *real-time*. As a new generalization to PRT, JRT reduces the shading integral at each vertex in the dynamic glossy scenes to a fast double function product integral.

In JRT, the light transport for each dynamic scene configuration is pre-computed at runtime interactively. For this purpose, we present an efficient algorithm to compute the light transport vectors on-the-fly, which are represented as the product of multiple functions. For instance, if only lighting varies, for each vertex we compute the radiance transfer vector as the product of BRDF, local visibility and global occlusions. Hence the shading integral reduces to a fast dot product of the resulting transfer vector and the time-variant lighting,

which can be performed in real-time. Similar techniques can be used to support dynamic view conditions. Using JRT, we demonstrate interactively manipulating (such as cloning, translating and scaling) glossy objects with real-time all-frequency shadows.

The rest of the chapter is organized as follows. In section 3.2 we present mathematical formalizations of the generalized multi-function product integral and Just-in-time Radiance Transfer technique. In section 3.3, we propose a novel *generalized Haar integral coefficient theorem*. In section 3.4 we present two efficient algorithms to evaluate the multi-function product integral and the product of multiple functions. Details of the implementation and some rendering results are presented in section 3.5. We conclude the chapter with some discussion of the advantage of our work and future research directions in section 3.6.

3.2 Problem Formalization

3.2.1 Multi-Function Product Integral

Given n distinct objects in a dynamic scene, the exitant radiance B at a surface point x along view direction θ due to distant environment lighting L is the product integral over all incident directions sampled at a surrounding cubemap S:

$$\begin{aligned} B(x,\theta) &= \int_S L(\varphi) O_1(x,\varphi) \prod_{i=2}^{n} O_i(x,\varphi) \rho(x, \varphi \leftrightarrow \theta)(N \cdot \varphi) d\varphi \\ &= \int_S L(\varphi) \widetilde{O}_1(x,\varphi) \prod_{i=2}^{n} O_i(x,\varphi) \rho(x, \varphi \leftrightarrow \theta) d\varphi \end{aligned} \tag{3.1}$$

where φ is the incident direction, N is the normal at x, ρ is the BRDF, O_1 is the *local visibility* at x due to self-occlusion. $O_i(2 \leq i \leq n)$ is the *dynamic occlusion* at x occluded by the i^{th} neighboring object in the scene. Conventionally, cosine term $(N \cdot \varphi)$ is combined with the BRDF term. In our implementation, it is combined with the self visibility O_1 as $\widetilde{O}_1$ to eliminate the dependance of the BRDF on the normal. Nevertheless, the framework can also handle representing $(N \cdot \varphi)$ as a single term in the multi-function product integral. In the equation, all participating functions are defined in a global coordinate system. Here, we assume that shading at a surface point is directly from the environment lighting, and we do not consider the inter-reflections between any two surfaces in the scene.

For a fixed vertex x and view direction θ, equation 3.1 can be simplified as:

$$B = \int L(\varphi)\widetilde{O}_1(\varphi)\prod_{i=2}^{n} O_i(\varphi)\rho(\varphi)d\varphi \tag{3.2}$$

It is exactly the product integral of $(n+2)$ functions:

$$B = \int \prod_{i=1}^{n+2} F_i(\varphi)d\varphi \tag{3.3}$$

Projecting each operand function $F_i(\varphi)$ onto an orthonormal basis set $\mathcal{B}$ yields:

$$F_i(\varphi) = \sum_{j=1}^{M}[f_{i,j}\ b_j(\varphi)] \tag{3.4}$$

where $f_{i,j}$ is the j^{th} basis coefficient, $b_j(\varphi)$ is the j^{th} basis function, and M is the size of $\mathcal{B}$. We define the n^{th}-order *basis product* $P^n_{b_1,b_2,\cdots,b_n}$ as the product of n arbitrary basis functions $b_1, b_2, \cdots, b_n$:

$$P^n_{b_1,b_2,\cdots,b_n} = \prod_{k=1}^{n} b_k(\varphi) \tag{3.5}$$

Its integral is the n^{th}-order *integral coefficient* $C^n_{b_1,b_2,\cdots,b_n}$:

$$C^n_{b_1,b_2,\cdots,b_n} = \int P^n_{b_1,b_2,\cdots,b_n} d\varphi = \int \prod_{k=1}^{n} b_k(\varphi) d\varphi \tag{3.6}$$

Thus, equation 3.3 can be represented as:

$$\begin{aligned} B &= \int \prod_{i=1}^{n+2} \sum_{j_i=1}^{M} [f_{i,j_i}\ b_{j_i}(\varphi)] d\varphi \\ &= \sum_{j_1=1}^{M} \sum_{j_2=1}^{M} \cdots \sum_{j_{n+2}=1}^{M} [\prod_{i=1}^{n+2} f_{i,j_i}\ \cdot \int\ \prod_{i=1}^{n+2} b_{j_i}(\varphi) d\varphi] \\ &= \sum_{j_1=1}^{M} \sum_{j_2=1}^{M} \cdots \sum_{j_{n+2}=1}^{M} [\prod_{i=1}^{n+2} f_{i,j_i} \cdot C^{n+2}_{b_{j_1},b_{j_2},\cdots,b_{j_{n+2}}}] \end{aligned} \tag{3.7}$$

Therefore, multi-function product integral in the primal (equation 3.3) corresponds to *the summation of the product of basis coefficients and integral coefficients* (equation 3.7). To our knowledge, there is no significant previous work on efficiently integrating the product of n-functions ($n > 3$). The most closely related work is on the 3^{rd}-order Haar product integral, namely, Haar triple product integral by Ng et al. [NRH04].

The integral coefficient in this chapter is a generalization of the tripling coefficient in [NRH04], namely, the 3^{rd}-order integral coefficient. In quantum physics literature, it is referred to as the Wigner coefficient [BL81]. The most popular $3-j$ coefficients (or Clebsch-Gorden coefficients, the 3^{rd}-order integral coefficients) and $6-j$ coefficients (or Racah coefficients, the 4^{th}-order integral coefficients) are well-studied with analytic formulae. The results have also been extended to $3n-j$ coefficients, which have been analytically examined and practically tabulated. However, the difficulty is in the efficient computation of the summations in equation 3.7.

An alternative approach to evaluating equation 3.3 goes as follows:

$$B = \int [[[[F_1(\varphi) \cdot F_2(\varphi)] \cdot F_3(\varphi)] \cdots F_{n+1}(\varphi)] \cdot F_{n+2}(\varphi)] d\varphi \tag{3.8}$$

Consequently, integrating the product of multiple functions reduces to computing a series of two-function products (as shown inside the square brackets), followed by a final double function product integral. In Physics, this approach relates $6-j$ coefficients with $3-j$ coefficients, and $9-j$ coefficients with $6-j$ coefficients [BL81]. It is also exploited by Zhou et al. [ZHL05] in computing the radiance from a single light source. Analyses in subsection 3.2.3 show that this recursive approach is computationally expensive and practically inefficient.

3.2.2 Just-In-Time Radiance Transfer

An effective approach to accelerating the evaluation of equation 3.3 is stated as follows:

$$B = \int [\prod_{i=1}^{n+1} F_i(\varphi)] \cdot F_{n+2}(\varphi) d\varphi = \langle T, F_{n+2}(\varphi) \rangle \tag{3.9}$$

where the radiance transfer vector T is the product of $n+1$ functions:

$$T = \prod_{i=1}^{n+1} F_i(\varphi) \tag{3.10}$$

If F_1, F_2, $\cdots$, F_{n+1} are fixed, in other words, only F_{n+2} varies, radiance transfer vector T needs to be computed only once. Therefore, shading integral reduces to a simple double function product integral of T and $F_{n+2}(\varphi)$, which can be approximated by an efficient low-dimensional dot product between the basis coefficients of both functions. Here we assume

that only one function in the shading integral varies. This assumption is very reasonable for lighting design systems, where normally the designer adjusts only one variable at a time, and real-time feedback is highly appreciated. For example, the designer may experiment with different lighting effects by fixing view conditions and objects. The designer may also render the scene from different view conditions by fixing the lighting and the objects. Another popular operation is to fix the lighting and view conditions, and relocate a single object in the scene. As long as there is only one (note that it can be any one) varying parameter, this approach can be used to generate all-frequency shadows in real-time.

In equation 3.9, the product of $n+2$ functions is factored into the product of two sets, one with $n+1$ functions, and the other with only one function. More generally, this factorization has the following form:

$$B = \int [\prod_{i=1}^{p} F_i(\varphi)] \cdot [\prod_{i=p+1}^{n+2} F_i(\varphi)] \, d\varphi = \langle T_1, T_2 \rangle \tag{3.11}$$

where $T_1 = \prod_{i=1}^{p} F_i(\varphi)$ and $T_2 = \prod_{i=p+1}^{n+2} F_i(\varphi)$ $(1 \leq p \leq n+2)$. As a result, the product of $n+2$ functions reduces to the double function product integral of two radiance transfer vectors.

Another factorization is to reduce high-order product integral (equation 3.7) to low-order product integral, for instance, the 3^{rd}-order product integral:

$$B = \int [\prod_{i=1}^{n+2} F_i(\varphi)] d\varphi = \int T \cdot F_{n+1}(\varphi) \cdot F_{n+2}(\varphi) d\varphi \tag{3.12}$$

where T is the product of n functions, $T = \prod_{i=1}^{n} F_i(\varphi)$. This approach may benefit certain operations such as adjusting the lighting and view conditions simultaneously while fixing

the scene (this resembles the triple product integral for relighting a static scene [NRH04]) . However, in our work on rendering dynamic glossy objects it is of little practical use because of the inefficiency of the triple product integral and the requirement of real-time feedback. In the chapter, we restrict the final shading in JRT as a double function product integral (equation 3.9).

By projecting T onto the orthonormal basis set $\mathcal{B}$, we have the k^{th} basis coefficient t_k of T, which is the inner product of T and the k^{th} basis function b_k:

$$\begin{aligned}
t_k &= \langle T, b_k \rangle \\
&= \int \prod_{i=1}^{n+1} [\sum_{j_i=1}^{M} (f_{i,j_i}\ b_{j_i})] \cdot b_k d\varphi \\
&= \int [\sum_{j_1=1}^{M} (f_{1,j_1}\ b_{j_1}) \cdots \sum_{j_{n+1}=1}^{M} (f_{n+1,j_{n+1}}\ b_{j_{n+1}})] \cdot b_k d\varphi \\
&= \sum_{j_1=1}^{M} \sum_{j_2=1}^{M} \cdots \sum_{j_{n+1}=1}^{M} [\ \overbrace{f_{1,j_1} \cdots f_{n+1,j_{n+1}}}^{n+1 \text{ coeffs}} \int \overbrace{b_k \cdot b_{j_1} \cdots b_{j_{n+1}}}^{n+2 \text{ bases}}\ d\varphi\ ] \\
&= \sum_{j_1=1}^{M} \sum_{j_2=1}^{M} \cdots \sum_{j_{n+1}=1}^{M} [\prod_{i=1}^{n+1} f_{i,j_i} \cdot C^{n+2}_{b_k, b_{j_1}, b_{j_2}, \cdots, b_{j_{n+1}}}]
\end{aligned} \tag{3.13}$$

Note that both the product integral of multiple functions (equation 3.7) and the k^{th} coefficient of the multi-function product (equation 3.13) reduce to the $(n+2)^{th}$-order integral coefficient C^{n+2}, which is the focus of the next section. It is worth noting that the results in this subsection apply to any basis set $\mathcal{B}$.

3.2.3 Comparison of Computational Complexity

In this subsection, we compare the time complexity of different approaches to computing the product integral of multiple functions in the dynamic scenes. Without loss of generality, we assume that the total number of operand functions is n ($n \geq 3$), involving the lighting, one BRDF, one local visibility, and $n-3$ global occlusions. As shown in section 3.5.1, the lighting is dynamically sampled at runtime, BRDF is dynamically interpolated from the tabulated data, and the global occlusions are also interpolated from the pre-computed visibility field on-the-fly. We assume that the total number of basis functions is M (each face of the cubemap is sampled with resolution at up to 256×256, hence $M = 65,536$). Each function is approximated with m significant coefficients in the basis domain (normally $m \ll M$).

General Basis: For an arbitrary basis set $\mathcal{B}$, there is no special formula of the integral coefficient $C^n_{b_{j_1},b_{j_2},\cdots,b_{j_n}}$. A straightforward brute-force approach will result in an algorithm with time complexity $O(M^n)$. Here we assume the n^{th}-order integral coefficients can be pre-computed and tabulated.

Recursive Two-function Products Using General Basis: Using equation 3.8, integrating the product of multiple functions reduces to the recursive two-function products, followed by a final double function product integral. From equation 3.8, we assume that the intermediate two-function product $G_i = G_{i-1} \times F_i$ ($1 < i \leq n+1$, $G_1 = F_1$). The k^{th} ($1 \leq k \leq M$) basis coefficient $g_{i,k}$ of G_i can be evaluated using the 3^{rd}-order integral

coefficient:

$$g_{i,k} = \langle G_i, b_k \rangle = \langle G_{i-1} \times F_i, b_k \rangle = \sum_{j_{i-1}} \sum_{j_i} g_{i-1,j_{i-1}} \; f_{i,j_i} \; C^3_{j_{i-1},j_i,k} \tag{3.14}$$

For an arbitrary basis, each two-function product takes $O(M^3)$. Here we assume the 3^{rd}-order integral coefficients can be pre-computed and tabulated. The total time complexity of the whole algorithm is $O(M^3 n)$.

Pixel Domain: In this case, we tabulate each operand function using the cubemap. Here, the integral coefficient $C^n_{b_{j_1},b_{j_2},\cdots,b_{j_n}}$ is simply the generalized Kronecker delta function: $\delta_{b_{j_1},b_{j_2},\cdots,b_{j_n}}$, which is 1 only when $b_{j_1} = b_{j_2} = \cdots = b_{j_n}$, otherwise 0. The time complexity of the resulting linear algorithm is $O(Mn)$. Despite its numerical efficiency, however, the pixel domain representation is of little practical use because the required dataset is prohibitively large (see section 3.5.1) and it is impossible to apply non-linear approximation in the pixel domain considering the dynamic nature of operand functions.

Recursive Two-function Products Using Spherical Harmonics: Here we consider evaluating the two-function products in equation 3.8 using spherical harmonics. Each two function product takes $O(M^{5/2})$ [NRH04], and the time complexity of the whole algorithm for evaluating equation 3.8 is $O(M^{5/2}n)$. It is worth noting that by keeping only a very small number m of low-frequency harmonic coefficients, the time complexity of the resulting algorithm reduces to $O(m^{5/2}n)$, where $m \ll M$. This linear approximation, however, band-limits high-frequency operand functions, as a result the approach filters out high-frequency components of the generated shadows and cannot account for high-glossy materials.

Table 3.1: Comparison of the time complexities of various approaches

Approaches	Time Complexity
General Basis	$O(M^n)$
Recursive General Basis	$O(M^3 n)$
Pixel Domain	$O(Mn)$
Recursive Spherical Harmonics	$O(m^{5/2} n)$
Recursive Haar	$O(Mn)$
Tree-structured Haar Integral	$O(mn) \quad m \ll M$

Recursive Two-function Products Using Haar Bases: Here we consider evaluating two-function products in equation 3.8 using Haar bases. Each two function product using Haar basis takes $O(M)$ [ZHL05]. Consequently, the time complexity of the whole algorithm for evaluating equation 3.8 is $O(Mn)$. Note that for this approach it is prohibitively expensive to apply non-linear approximation to the intermediate two-function products since they are dynamically generated.

Tree-structured Haar Integral: The time complexity of the proposed tree-structured algorithm in section 3.4.3 is $O(mn)$.

This table summarizes the time complexity results of the aforementioned approaches. We conclude that the recursive two-function products approach using spherical harmonics cannot account for high-frequency shadows and high-glossy materials. Pixel domain approach and the recursive two-function products approach using Haar bases are expensive because of

the inherent difficulty in non-linear approximation, while the tree-structured Haar integral approach is computationally favorable and practically attractive due to its capability to efficiently support high-frequency materials. Extensive experiments show that the tree-structured Haar integral approach is nearly 40-100 times faster than the pixel domain product integral and the recursive approach using Haar tripling coefficients.

3.3 Generalized Haar Integral Coefficient Theorem

We focus on the efficient computation of the multi-function product integral (equation 3.7) and the product of multiple functions (equation 3.13). For this purpose, we choose Haar bases as the basis set $\mathcal{B}$. Compared with the pixel domain representation, wavelets allow us to approximate signals at low distortion with a small number of significant coefficients. Haar bases, amazingly, have an interesting property that simplifies the computation as many of the integral coefficients are zero. Therefore, by using only non-zero wavelet coefficients and non-zero integral coefficients, evaluation of both the multi-function product integral and the product of multiple functions can be significantly accelerated.

3.3.1 2D Haar Bases

Nonstandard Haar wavelet transform [SDS96] decomposes a $2^n \times 2^n$ image into a 2D signal with $2^n \times 2^n$ coefficients. Each coefficient corresponds to a basis function defined in the region $\langle j, k, l \rangle$, where j is the *scale* ($0 \leq j < n$), k and l are spatial translations ($0 \leq k, l < 2^j$). In each region $\langle j, k, l \rangle$, four normalized 2D Haar basis functions are defined:

ϕ_{kl}^j *normalized Haar scaling basis function*:

$$\phi_{kl}^j(x, y) = 2^j \phi^0(2^j x - k, 2^j y - l)$$

where ϕ^0 is the mother scaling function.

ψ_{kl}^j *normalized Haar wavelet basis function.* There are three types of wavelets defined in the region $\langle j, k, l \rangle$:

$$\psi_1{}_{kl}^j = 2^j \psi_1{}^0(2^j x - k, 2^j y - l)$$

$$\psi_2{}_{kl}^j = 2^j \psi_2{}^0(2^j x - k, 2^j y - l)$$

$$\psi_3{}_{kl}^j = 2^j \psi_3{}^0(2^j x - k, 2^j y - l)$$

where $\psi_1{}^0$, $\psi_2{}^0$ and $\psi_3{}^0$ are three different mother wavelets, denoting the horizontal, vertical and diagonal differences.

Mother scaling basis function and mother wavelets are illustrated in Figure 2.2. Additional examples of Haar basis functions are illustrated in Figures 2.5 and 3.3. In these figures, functions are positive where white, negative where black and zero where gray. For simplicity, amplitude is ignored in the diagrams.

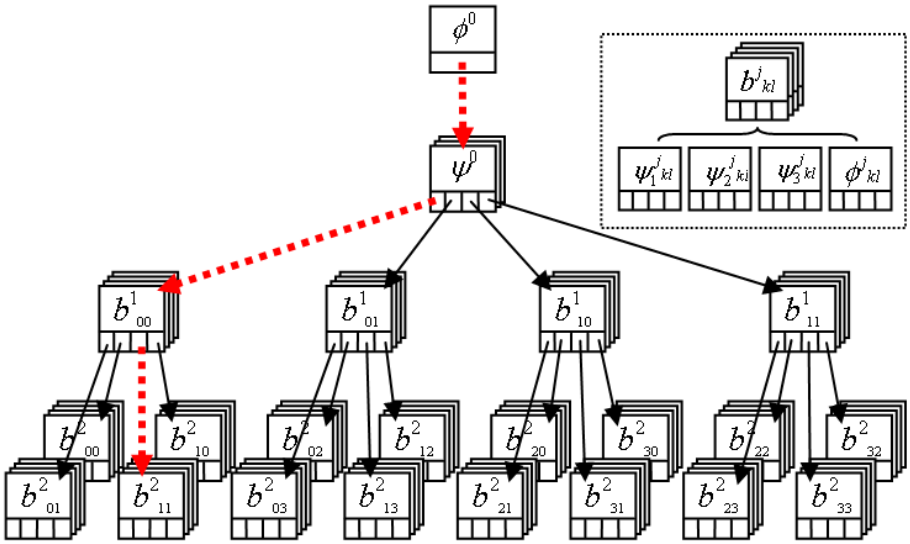


Figure 3.2: Example of a *complete* Haar basis tree. The mother scaling basis function is defined as the parent of all other basis functions. As shown in the upper-right corner, in each node in the tree up to four basis functions (denoted by b_{kl}^{j}) are defined: one scaling basis and three wavelet bases. Meanwhile, each node contains up to four child nodes. The basis tree contains not only the restricted bases, but also child scaling bases. The red dotted lines denote the directed path going through all basis functions in Figure 3.3-e.

From the definition, basis functions are the dilated and translated versions of the mother basis functions. A basis function b_s is said to be the *parent* of another basis function b_t if the support of b_s completely covers the support of b_t, and the scale of b_s is less than the scale of b_t. As a result, basis functions in the region $\langle j, k, l \rangle$ are immediate parent of the basis functions defined at scale $(j+1)$ and spatial locations $(2k, 2l)$, $(2k+1, 2l)$, $(2k, 2l+1)$ and $(2k+1, 2l+1)$. Naturally all basis functions form a directed basis tree. Example of

a complete basis tree is shown in Figure 3.2. For simplicity, we define the mother scaling function as the parent of any other basis function, and it lies in the root node of the tree. We also define the set of basis functions containing only the mother scaling basis function and wavelet basis functions as the *restricted* bases. As can be seen from Figure 3.2, given a certain set of basis functions, for any two basis functions, if one is the parent of the other then there must exist a directed path in the basis tree going through all basis functions in the set.

Without explicit notations, all basis functions in this section refer to the normalized 2D Haar basis functions.

3.3.2 Product Of 2D Haar Bases

In this subsection, we show that the product P^n of arbitrary Haar bases is also a Haar basis, scaled by some signed constant, which we refer to as the *magnitude* of P^n, $|P^n|$. Especially, the magnitude of a normalized basis function is 1. Some examples of the basis products are illustrated in Figure 2.5 and Figure 3.3. As shown in the figures, the magnitude of a basis product is not necessarily a positive value.

We make following observations on the product of two Haar basis functions:

1. The multiplication of basis functions is commutative.

2. The product of a scaling basis function with another basis function of identical support at scale j is the second basis function, with magnitude 2^j.

3. The square of a wavelet basis function at scale j is the scaling basis function of identical support, with magnitude 2^j.

4. The product of two different wavelets of the same support at scale j is the third wavelet of identical support, with magnitude 2^j.

5. The product of a child basis function with a parent basis function at scale $\widetilde{j}$ is the child basis function, with magnitude $\pm 2^{\widetilde{j}}$. The sign of the product is the sign of the sub-region of the parent basis function that the child basis function falls into.

Some examples of the product of two Haar basis functions are shown in Figure 2.5. As illustrated in the figure, the product of two basis functions is also a basis function, scaled by its magnitude, although the magnitude may be zero. From the above observations, we have following conclusion on arbitrary basis product:

Lemma 2 *Basis product P^n is always a basis function, scaled by its magnitude.*

Haar basis is a diadic basis set, and the support of each child basis function is always one quarter of the support of its immediate parent. As shown in Figure 2.5, if the supports of two basis functions overlap, then there exists a directed path in the basis tree originating from one basis function to the other, consequently their product is non-zero. Furthermore, if the product is non-zero, then their support must overlap, consequently in the basis tree there

must exist a directed path originating from one basis function to the other. This observation can be generalized to the product of multiple basis functions:

Lemma 3 *Product P^n is non-zero if and only if there is a directed path in the basis tree going through all operand basis functions.*

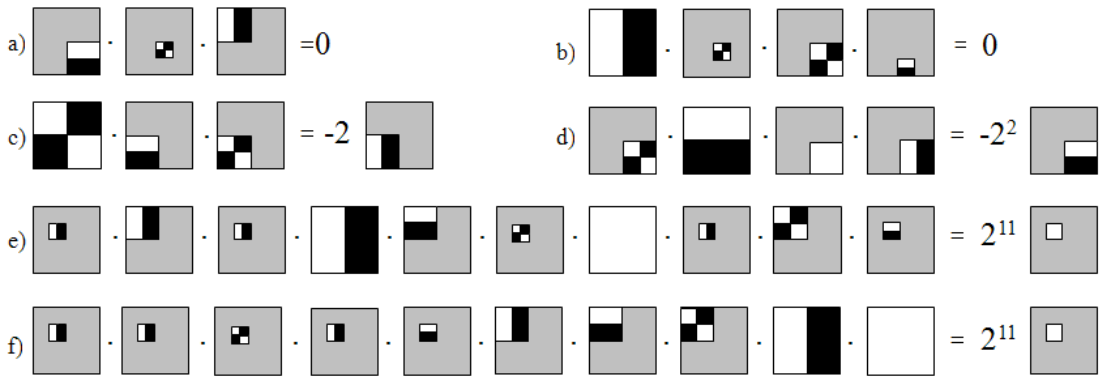


Figure 3.3: Some examples of the product of multiple Haar basis functions. In the first row, basis product is zero. In the second row, the type of the product is a wavelet basis function. In e), the type of the product is a scaling basis function. f) is the ranked version of e). The magnitude of each basis product is also shown in the figure.

Ranking: Sorting basis set $\langle b_1, b_2, \ldots, b_n \rangle$ from the finest scale to the coarsest scale. For basis functions defined at the same scale, wavelets appear first.

For example, sorting operand basis functions in Figure 3.3-e results in the ranked basis set in Figure 3.3-f. From the commutativity rule (observation 1), ranking does not change the magnitude or the type of the product. However, it facilitates the evaluation of the basis

product. For instance, the type of the product of the ten basis functions in Figure 3.3-e is same as the type of the product of the first five bases in Figure 3.3-f.

Given $P^n = P^{n-1} \times b_n$, ranking ensures that the scale of b_n is no more than the scale of P^{n-1}. If b_n is a parent of P^{n-1}, observation 5 shows that P^n and P^{n-1} are of the same type. Recursive applications of this observation result in the following result:

Lemma 4 *The type of the basis product P^n is same as the type of the product of basis functions at the finest scale.*

Given an arbitrary number of basis functions of identical support, we have following result on their product:

Lemma 5 *Product of n ($n \geq 1$) basis functions of the same support at scale j is a basis function of identical support with magnitude $2^{(n-1)j}$. The type of the basis product is determined by the parities of the numbers of three kinds of wavelets as:*

Parity	number of ψ_1	0	0	0	0	1	1	1	1
	number of ψ_2	0	0	1	1	0	0	1	1
	number of ψ_3	0	1	0	1	0	1	0	1
type of the product		ϕ	ψ_3	ψ_2	ψ_1	ψ_1	ψ_2	ψ_3	ϕ

Lemma 5 can be trivially verified using observations 2, 3 and 4. Especially, the type of the basis product P^n is a scaling basis function, if and only if the numbers of the three kinds of wavelets ψ_1, ψ_2 and ψ_3 have the same parity.

On the magnitude of arbitrary basis product, we have following conclusion:

Lemma 6 *The magnitude of the non-zero product P^n is $\pm 2^{\sum_j - j_0}$, where $\sum_j$ is the sum of the scales of all operand basis functions, j_0 is the scale of the finest basis function. The sign of the product is the multiplication of the signs of the sub-regions of all parent basis functions that the child basis function falls into.*

Proof Given $P^n_{b_1,b_2,\cdots,b_n} \neq 0 (n \geq 1)$. After ranking, we have bases $b^{j_0}_{s_1}$, $b^{j_0}_{s_2}$, $\cdots$, $b^{j_0}_{s_m}$, $b^{j_{m+1}}_{s_{m+1}}$, $\cdots$, $b^{j_n}_{s_n}$, where $j_0 > j_{m+1} \geq \cdots \geq j_n$. Lemma 5 shows that the product of $b^{j_0}_{s_1}$, $b^{j_0}_{s_2}$, $\cdots$, $b^{j_0}_{s_m}$ is a basis function of identical support, say, b^{j_0}, with magnitude $2^{(m-1)j_0}$. So, we have:

$$P^n_{b_1,\cdots,b_n} = b^{j_0}_{s_1} \cdots b^{j_0}_{s_m} b^{j_{m+1}}_{s_{m+1}} \cdots b^{j_n}_{s_n} = 2^{(m-1)j_0} \cdot b^{j_0} b^{j_{m+1}}_{s_{m+1}} \cdots b^{j_n}_{s_n} \tag{3.15}$$

Because $P^n_{b_1,b_2,\cdots,b_n} \neq 0$, $b^{j_{m+1}}_{s_{m+1}}$ must be a parent of b^{j_0}. Observation 5 shows that $b^{j_0} \cdot b^{j_{m+1}}_{s_{m+1}} = \pm 2^{j_{m+1}} \cdot b^{j_0}$. Therefore, we have:

$$P^n_{b_1,b_2,\cdots,b_n} = \pm 2^{(m-1)j_0 + j_{m+1}} \cdot b^{j_0} b^{j_{m+2}}_{s_{m+2}} \cdots b^{j_n}_{s_n} \tag{3.16}$$

Repeat the previous step for all the remaining bases, we have:

$$P^n_{b_1,b_2,\cdots,b_n} = \pm 2^{(m-1)j_0 + j_{m+1} + j_{m+2} + \cdots + j_n} b^{j_0} = \pm 2^{\sum_j - j_0} b^{j_0} \tag{3.17}$$

This completes the proof.

These lemmas are sufficient to determine the product of arbitrary basis functions. For instance, Lemma 3 shows that the basis product in Figure 3.3-e is non-zero since there is a directed path in the basis tree going through all basis functions (refer to Figure 3.2). Ranking shows that the finest basis functions are defined in the region $\langle 2, 1, 1 \rangle$, and the numbers of the three kinds of wavelets in the region are 3, 1 and 1, respectively. Therefore, Lemma 5

shows that the type of the basis product is the scaling basis ϕ^2_{21}. Finally, Lemma 6 shows that the magnitude of the basis product is $+2^{11}$.

3.3.3 Haar Integral Coefficients

From the definition of 2D Haar basis functions, following observations hold:

6. The integral of a wavelet basis vanishes. $\iint \psi^j_{kl} dxdy = 0$.

7. The integral of a scaling basis is non-zero. $\iint \phi^j_{kl} dxdy = 2^{-j}$.

We have showed that the product of arbitrary 2D Haar basis functions is always a basis function, scaled by its magnitude. Since there are only two kinds of basis functions: scaling bases and wavelet bases, from observations 6 & 7 we have:

Lemma 7 *Integral coefficient $C^n \neq 0$, if and only if the type of the corresponding non-zero basis product P^n is a scaling basis, and $C^n = 2^{-j} \times |P^n|$, where j is the scale of P^n.*

3.3.4 Haar Integral Coefficients

Based on the previous results, we have following conclusion on the integral coefficients of arbitrary 2D Haar basis functions:

Generalized Haar Integral Coefficient Theorem *The n^{th}-order Haar integral coefficient C^n has a non-zero value, if and only if the numbers of the three kinds of wavelets ψ_1, ψ_2 and ψ_3 at the finest scale have the same parity. In this case, the integral coefficient is $\pm 2^{\sum_j -2j_0}$, where $\sum_j$ is the sum of the scales of all operand basis functions, and j_0 is the scale of the finest basis function. The sign of the integral coefficient is the multiplication of the signs of the sub-regions of all parent basis functions that the child basis function falls into.* ♢

Some examples of the product of multiple Haar basis functions are illustrated in Figure 3.3. Using the theorem, it is trivial to get the corresponding integral coefficients.

The theorem applies to arbitrary Haar integral coefficients, whereas the Haar tripling coefficient theorem [NRH04] is limited to the 3^{rd}-order integral coefficients defined on the *restricted* bases. In this case, we can set all three operand basis functions to the restricted Haar basis functions, and do a simple case analysis using Lemma 5 to get the Haar tripling coefficient theorem as expected. Similar approaches can be used to derive the Haar Quad-coefficient theorem as mentioned in Chapter 4.

3.3.5 A Recursive Perspective

In this subsection, we present an alternative approach to computing the integral coefficients. Results in this subsection supplement the previous theorem, and are used in section 3.4 to

construct two efficient algorithms for evaluating the multi-function product integral and the product of multiple functions.

Corollary 1 Haar integral coefficient $C^n (n \geq 2)$ has a non-zero value, if and only if one of the following cases holds for the ranked basis product $P^n = P^{n-1} \times b_n$:

1. P^{n-1} and b_n are wavelets of the same type, and both have identical support. In this case, $C^n = |P^{n-1}|$.
2. Both P^{n-1} and b_n are the mother scaling function. $C^n = 1$.
3. P^{n-1} is a scaling basis function, and b_n is its parent at scale j. $C^n = \pm 2^j \times C^{n-1}$. The sign of the integral coefficient is the sign of the sub-region of b_n that P^{n-1} falls into. $\diamondsuit$

Proof Ranking does not change the integral coefficient C^n. Because of ranking, the scale of b_n is no more than the scale of P^{n-1}. Since $C^n \neq 0$, Lemma 7 shows that the type of P^n is a scaling basis.

If b_n is the parent of P^{n-1}, then observation 5 shows that P^{n-1} is a scaling basis. Observations 5, 7 and Lemma 7 conclude that $C^n = \pm 2^j \times C^{n-1}$. This establishes case 3.

If b_n is not the parent of P^{n-1}, then they have identical support. If both b_n and P^{n-1} are the mother scaling function, then $C^n = 1$. This establishes case 2. Otherwise, b_n and P^{n-1} must be wavelets of the same type. Observation 3 and Lemma 7 show that $C^n = |P^{n-1}|$. This establishes case 1, completing the proof.

In case 1, the integral coefficient reduces to the magnitude of a ranked basis product of wavelet type, which is given as:

Corollary 2 Ranked Haar basis product $P^n = P^{n-1} \times b_n (n > 1)$ is a wavelet, if and only if one of the following cases holds:

1. P^{n-1}, P^n and b_n have identical support at scale j. P^{n-1} is a scaling basis, and both b_n and P^n are wavelets of identical type. In this case, $|P^n| = 4^j \times C^{n-1}$.

2. P^{n-1}, P^n and b_n have identical support at scale j. b_n is a child scaling basis, and both P^{n-1} and P^n are wavelets of identical type. $|P^n| = 2^j \times |P^{n-1}|$.

3. P^{n-1}, P^n and b_n are wavelets of identical support at scale j, and three are of different wavelet types. $|P^n| = 2^j \times |P^{n-1}|$.

4. P^{n-1} and P^n are wavelets of the same type, both have identical support, and b_n is their parent at scale j. $|P^n| = \pm 2^j \times |P^{n-1}|$. The sign of the product is the sign of the sub-region of b_n that P^{n-1} falls into. $\diamondsuit$

Proof Ranking ensures that the scale of b_n is no more than the scale of P^{n-1}. Because the type of P^n is a wavelet, then one of P^{n-1} and b_n must be a wavelet at the finest scale.

If b_n is a parent of P^{n-1}, then observation 5 shows that P^{n-1} and P^n have identical support, and both are wavelets of the same type. $|P^n| = \pm 2^j \times |P^{n-1}|$. This establishes case 4.

If b_n is not the parent of P^{n-1}, then b_n and P^{n-1} have identical support as P^n. There are three possibilities. If P^{n-1} is a scaling basis, then b_n is a wavelet of identical type as P^n. Observation 2 and Lemma 7 show that $|P^n| = 4^j \times C^{n-1}$. This establishes case 1. If P^{n-1} is a wavelet, then the type of the wavelet is same as that of P^n, and b_n is a child scaling basis. Observation 2 and Lemma 7 show that $|P^n| = 2^j \times |P^{n-1}|$. This establishes case 2. If both

b_n and P^{n-1} are wavelets, then they are different wavelets, and both are of different types from P^n. Observation 4 shows that $|P^n| = 2^j \times |P^{n-1}|$. This establishes case 3. The proof is complete.

A simple recursive application of case 2 in corollary 1 results in the following conclusion:

Lemma 1 *If all operand basis functions are the mother scaling function, then the integral coefficient* $C^n = 1$.

3.4 Algorithms

In this section, we first present an efficient algorithm to evaluate the multi-function product integral, followed by another to compute the product of multiple functions.

Without loss of generality, we assume that the number of operand functions is n, the total number of basis functions is M, and the number of wavelet coefficients used in the non-linear approximation is m. For convenience, we re-write equations 3.7 and 3.13 in terms of n basis functions:

$$B = \sum_{j_1=1}^{M} \sum_{j_2=1}^{M} \cdots \sum_{j_n=1}^{M} [\prod_{i=1}^{n} f_{i,j_i} \cdot C^n_{b_{j_1}, b_{j_2}, \cdots, b_{j_n}}] \tag{3.18}$$

$$t_k = \sum_{j_1=1}^{M} \sum_{j_2=1}^{M} \cdots \sum_{j_n=1}^{M} [\prod_{i=1}^{n} f_{i,j_i} \cdot C^{n+1}_{b_k, b_{j_1}, b_{j_2}, \cdots, b_{j_n}}] \tag{3.19}$$

Evaluating equation 3.18 using the generalized Haar integral coefficient theorem by brute-force results in a simple algorithm with prohibitively expensive time complexity $O(M^n n)$. Here the evaluation of the integral coefficients takes $O(n)$. A natural approach to accelerating computation is to apply non-linear approximation to all operand functions. For this purpose, we use the augmented quadtree as the underlying data structure.

3.4.1 Data Structure

For convenience, we re-write the definition of the augmented quad-tree structure (refer to section 2.2.1) as:

struct augnode

$\psi[3]$: **wavelet coefficients**;

p_{sum} : **signed parent summation**;

$ch[4]$: child pointers;

end

Here, $\psi[3]$ are the three wavelet coefficients of different types with identical support. $ch[4]$ point to four immediate child nodes. The definition of field p_{sum} is given in section 2.2.2. we re-write the definition of the wavelet tree as:

struct augtree

dc : **mother scaling coefficient**;

node : pointer to **struct** augnode;

end

One may notice that a complete wavelet tree resembles a complete basis tree (see Figure 3.2). However, these two trees are different. Wavelet tree stores basis coefficients, while the basis tree is composed of basis functions.

3.4.2 Tree-Structured Algorithm for Multiple Function Product Integral

From Lemmas 3 and 7, we conclude that for non-zero integral coefficients, there must exist a directed path in the basis tree traversing through all operand basis functions. Therefore, by confining operand coefficients to those on a directed path in the wavelet tree, integrating the product of multiple functions can be greatly accelerated since operations involving zero integral coefficients are eliminated. Based on this intuition, we designed a tree-structured rendering algorithm to *synchronously* traverse n wavelet trees, $T_1, T_2, \cdots, T_n$. The entry point of the algorithm is *FunctionProductIntegral*. From Lemma 1, product integral B is initialized as the direct product of the mother scaling coefficients.

As we recursively traverse wavelet trees, the first step is to rank participating basis functions from the finest scale to the coarsest scale. As shown in line 6, all operand bases have the same scale, as a result ranking reduces to choosing non-null nodes. The recursion stops

in line 7 if there is at most one non-null node. In this case, because each node can only contain wavelet coefficients, and the integral of wavelet vanishes, the corresponding integral coefficients must be zero. For the null nodes in line 6, the contribution of their parents are not ignored but accumulated in the variable *cum* as shown in line 9, which gives the product of the signed parent summations. One may notice that here t_i $(k < i \leq n)$ is a null node. The signed parent summation $t_i \rightarrow p_{sum}$ is evaluated from t_i's closest *physical* parent node, which can be passed as a parameter during the traversal. In line 8, p_{sum} field is computed in constant time using the algorithm in subsection 2.2.2.

---

1 **algorithm** FunctionProductIntegral(augtree $T_1, T_2, \cdots, T_n$)

2 $B = \prod_{i=1}^{n} T_i.dc$;

3 traverseAugTrees($1, T_1.node, T_2.node, \cdots, T_n.node$);

4 **end**.

5 **routine** traverseAugTrees(cum, augnode $t_1, t_2, \cdots, t_n$)

6 Ranking, such that t_1, $\cdots$, $t_k \neq null$ and t_{k+1}, $\cdots$, $t_n = null$;

7 **if** $k < 2$ **then** return;

8 update p_{sum};

9 $cum \times= \prod_{i=k+1}^{n} t_i \rightarrow p_{sum}$;

10 B += $cum \times$ getProductIntegral($t_1, \cdots, t_k$);

11 **for** $i = 0$ **to** 3

12 traverseAugTrees(cum, $t_1.ch[i], \cdots, t_k.ch[i]$);

In line 10, product integral of these non-zero coefficients at the lowest traversal depth are computed in subroutine *getProductIntegral*, and multiplied with the accumulated parent summations of these null-nodes at the current traversal depth. Support routine *getProductIntegral* is directly converted from corollary 1, which recursively calculates the value of non-zero integral coefficients:

routine getProductIntegral(augnode $t_1, \cdots, t_n$)

if $n = 1$ **return** 0;

return $t_n.p_{sum} \times$ getProductIntegral$(t_1, \cdots, t_{n-1})$

$+ t_n.\psi[0] \times$ getWaveletProduct$(0, 1, 2, t_1, \cdots, t_{n-1})$

$+ t_n.\psi[1] \times$ getWaveletProduct$(1, 0, 2, t_1, \cdots, t_{n-1})$

$+ t_n.\psi[2] \times$ getWaveletProduct$(2, 0, 1, t_1, \cdots, t_{n-1})$;

This routine refers to *getWaveletProduct*, which is derived from corollary 2. Note that case 2 of corollary 2 is not applicable here because all participating basis functions are limited to the *restricted* bases (due to the nonstandard wavelet transform). Routine *getWaveletProduct* evaluates the magnitude of the product, where the type of the basis product is a wavelet identical to the first input parameter a. In the routine, input parameters a, b and c are used to differentiate three kinds of wavelets.

routine getWaveletProduct$(a, b, c,$ augnode $t_1, \cdots, t_n)$

if $n = 1$ **return** $t_1.\psi[a]$;

return $t_n.\psi[a] \times \prod_{i=1}^{n-1} t_i.p_{sum}$

$$+ t_n.p_{sum} \times \text{getWaveletProduct}(a, b, c, t_1, \cdots, t_{n-1})$$

$$+ 4^{t_n \rightarrow scale} \times t_n.\psi[a] \times \text{getProductIntegral}(t_1, \cdots, t_{n-1})$$

$$+ 2^{t_n \rightarrow scale} \times (t_n.\psi[b] \times \text{getWaveletProduct}(c, a, b, t_1, \cdots, t_{n-1})$$

$$+ t_n.\psi[c] \times \text{getWaveletProduct}(b, a, c, t_1, \cdots, t_{n-1}));$$

The time complexity of routine *getProductIntegral* is $O(4^n)$. In the next subsection, we show that it can be optimized to $O(n)$.

3.4.3 Algorithm Optimization

Support routines *getProductIntegral* and *getWaveletProduct* are designed in a top-down fashion. To compute the product integral of n nodes, they are recursively expanded to calculate the product (integral) of $n-1$ nodes, and so on. The total number of expansions is $\frac{4^n-4}{3}$, and there are many repetitive operations involved. The redundant expansions can be eliminated using dynamic programming technique. The critical idea is to introduce an intermediate data structure: table $\mathcal{T}$, on which the computation is conducted bottom-up. Table $\mathcal{T}$ contains n entries:

struct table $\mathcal{T}[n]$

$\phi, \psi[3]$: magnitude;

cum : cumulative p_{sum};

end

Field $\mathcal{T}[i].\phi$ stores the magnitude of the product from subset $\langle t_1, t_2, \cdots, t_i \rangle \quad (1 \leq i \leq n)$ where the type of the product is a scaling basis function. It is worth noting that although the input coefficients relate to the *restricted* basis functions, the type of their product may be child scaling bases (see Lemma 5). Field $\mathcal{T}[i].\psi[0]$ (or, $\mathcal{T}[i].\psi[1]$, $\mathcal{T}[i].\psi[2]$) stores the magnitude of the product from the subset where the product is a wavelet of type $\psi[0]$ (or $\psi[1]$, $\psi[2]$, respectively). Field $\mathcal{T}[i].cum$ stores the cumulative signed parent summation in the subset. Using table $\mathcal{T}$, routine *getProductIntegral* is optimized as *getProductIntegralDP*:

routine getProductIntegralDP(augnode $t_1, \cdots, t_n$)

if $n = 1$ **return** 0;

$\mathcal{T}[1].\{\phi, \psi[0], \psi[1], \psi[2], cum\} = \{0,\ t_1.\{\psi[0],\ \psi[1],\ \psi[2],\ p_{sum}\}\}$;

for i = 2 **to** n-1

$\mathcal{T}[i].cum = t_i.p_{sum} \times \mathcal{T}[i-1].cum$;

$\mathcal{T}[i].\phi = \text{getP}(i)$; $\mathcal{T}[i].\psi[0] = \text{getW}(0, 1, 2, i)$;

$\mathcal{T}[i].\psi[1] = \text{getW}(1, 0, 2, i)$; $\mathcal{T}[i].\psi[2] = \text{getW}(2, 0, 1, i)$;

return getP(n);

Helper functions *getP* and *getW* are simply derived from *getProductIntegral* and *getWavelet-Product*:

routine getP(i)

if $i = 1$ **return** 0;

return $t_i.p_{sum} \times \mathcal{T}[i-1].\phi \ + \ t_i.\psi[0] \times \mathcal{T}[i-1].\psi[0]$

$$+\ t_i.\psi[1] \times \mathcal{T}[i-1].\psi[1]\ +\ t_i.\psi[2] \times \mathcal{T}[i-1].\psi[2]\ \ ;$$

routine getW(a, b, c, i)

if $\quad i = 1 \quad$ **return** $\quad \mathcal{T}[1].\psi[a];$

return $t_i.\psi[a] \times \mathcal{T}[i-1].cum$

$+\ t_i.p_{sum} \times T[i-1].\psi[a]\ +\ 4^{t_i \to scale} \times t_i.\psi[a] \times \mathcal{T}[i-1].\phi$

$+\ 2^{t_i \to scale} \times (t_i.\psi[b] \times \mathcal{T}[i-1].\psi[c]\ +\ t_i.\psi[c] \times \mathcal{T}[i-1].\psi[b]);$

The time complexity of routine *getProductIntegralDP* is O(n). The time complexity of the whole algorithm for the multi-function product integral is $O(mn)$.

3.4.4 Tree-Structured Algorithm for Multiple Function Product

In this subsection, we present the algorithm for the product of multiple functions. Note the similarity between equation 3.18 and equation 3.19. The algorithm here is very similar to *FunctionProductIntegral*. The input to the algorithm are n functions encoded in the wavelet trees T_1, T_2, $\cdots$, T_n. The output of the algorithm is the product of these functions encoded in the wavelet tree T_0. It is worth noting that with a single call to the algorithm, all non-zero coefficients of the product are evaluated and encoded in the augmented quadtree. The entry point of the algorithm is *FunctionProduct*.

From Lemma 1, the coefficient of the mother scaling basis function $T_0.dc$ is initialized as the product of all *dc* coefficients (line 2). In line 10, product integral of these non-zero coefficients

at the lowest traversal depth are computed using the optimized routine *getProductIntegralDP*. Next the magnitudes of the three kinds of wavelet type products are retrieved from table $\mathcal{T}$ and used to evaluate wavelet coefficients at the current traversal depth (lines 11, 12 and 13). Wavelet coefficients in the parent nodes are evaluated in routine *updateParents*, where the definition of array *sign* is given in subsection 2.2.2.

---

1 **algorithm** FunctionProduct(augtree $T_0, T_1, T_2, \cdots, T_n$)

2 $T_0.dc = \prod_{i=1}^{n} T_i.dc$;

3 getCoefficients($1, T_0.node, T_1.node, T_2.node, \cdots, T_n.node$);

4 **end**.

5 **routine** getCoefficients(cum, augnode $t_0, t_1, t_2, \cdots, t_n$)

6 Ranking, such that t_1, $\cdots$, $t_k \neq null$ and t_{k+1}, $\cdots$, $t_n = null$;

7 **if** $k = 0$ **then** return;

8 update p_{sum};

9 $cum \times= \prod_{i=k+1}^{n} t_i \rightarrow p_{sum}$;

10 updateParents(t_0, $cum \times$ getProductIntegralDP($t_1, \cdots, t_k$));

11 $t_0.\psi[0] = cum \times \text{getW}(0, 1, 2, k)$;

12 $t_0.\psi[1] = cum \times \text{getW}(1, 0, 2, k)$;

13 $t_0.\psi[2] = cum \times \text{getW}(2, 0, 1, k)$;

14 **for** $i = 0$ **to** 3

15 getCoefficients(cum, $t_0.ch[i], t_1.ch[i], \cdots, t_k.ch[i]$);

---

routine updateParents(augnode t_0, val)

$t_0 \rightarrow dc \mathrel{+}= val$;

for scale $j = 0, 1, \cdots,\ t_0.scale - 1$

// t_0 lies in the quadrant (k, l) of its parent t_p at scale j

$t_p.\psi[0] \mathrel{+}= \ \text{sign}(0, k, l) \times 2^j \times val$;

$t_p.\psi[1] \mathrel{+}= \ \text{sign}(1, k, l) \times 2^j \times val$;

$t_p.\psi[2] \mathrel{+}= \ \text{sign}(2, k, l) \times 2^j \times val$;

Practically the total number of scales in routine *updateParents* is up to 8 (assuming up to $6 \times 256 \times 256$ cubemaps). Therefore, the logarithmic complexity factor in the routine is limited to a constant. Consequently, the time complexity of the algorithm *FunctionProduct* is $O(mn)$.

3.5 Implementation

In this section, we present the pre-computation and rendering details. All experiments are conducted on a desktop computer with a single Intel Pentium4 3.2GHz CPU.

3.5.1 Pre-Computation

Visibility: For each scene entity, we sample visibility field at its surface points (termed as *local visibility*) and in the important surrounding regions (termed as *global visibility*). For the global visibilities we use two sampling schemes: *planar sampling* and *spherical sampling*. In the planar sampling scheme, we densely sample visibilities in the form of concentric circles on each entity's *virtual* ground plane, which is the lower plane of its bounding box. The center of the concentric circles is the projection of the object center on the plane. In the spherical sampling scheme, we sample the surrounding region around each scene entity in the form of concentric spheres, which is similar to the object occlusion field [ZHL05]. In our implementation, the center of the concentric spheres coincides with the center of the concentric circles. For planar sampling scheme, we use 100 concentric circles, each being sampled at 200 points. The radiuses of these concentric circles range from $0.05r$ to $10r$, where r is the radius of the projection of the instance on the virtual ground plane. For spherical sampling scheme, we use 20 concentric spheres. The radiuses of these concentric spheres range ranging from $0.2R$ to $6R$, where R is the radius of the bounding sphere. Each concentric sphere is sampled at $6 \times 9 \times 9$. Since we densely sample visibilities on the ground plane, we can account for intricate cast shadows on that plane. The absolute difference of the radius of the neighboring concentric circles/spheres in the planar and spherical sampling schemes increases linearly with increasing distance from the sampling center.

Table 3.2: Pre-computation statistics for generalized wavelet product integral

Entity	Vertices		Sampling Points		Raw	Num. of	Precomp.
	Raw	Fine	Planar	Spherical	Data	Quadnodes	Time
Table	4K	25K	20K	10K	1.3G	15M	20m
Chair	5K	30K	20K	10K	1.5G	16M	23m
Floor	4	62K	0	0	1.5G	49	1s

At each sampling point, visibilities are pre-computed as a cubemap by rasterizing the coarse model using graphics hardware, with the standard super-sampling enabled (up to 8 samples per pixel). Each cubemap is rasterized at resolution $6 \times 64 \times 64$. Note that multiple instances of a scene entity share the same geometry and visibility field. It is worth noting that all vertices of the planar object "Floor" have identical visibilities, therefore this huge dataset can be represented with a very small number of quadnodes.

BRDF: Pre-computing BRDF is quite similar to the previous work [NRH04]. In the implementation, Phong BRDFs (shininess: up to 200) are sampled with a resolution in $\theta_r \times \varphi$ of up to $(6 \times 64 \times 64) \times (6 \times 64 \times 64)$, where θ_r is the reflection vector of the view direction θ about normal N.

Compression and Representation: We project the pre-computed visibilities and BRDFs onto Haar bases using the nonstandard wavelet transform [SDS96], then perform nonlinear approximation to discard insignificant coefficients less than a certain threshold. Practically, for each face of the cubemap, retaining $60 - 120$ significant coefficients is sufficient to render

images close to the references. The resulting data are encoded in the augmented quad-tree structure.

3.5.2 General Rendering Algorithm

Figure 3.4: The right scene is composed in our system through cloning, translating and scaling from the left scene. Both scenes are illuminated as inside the Grace Cathedral, with identical view condition.

Lighting: In the experiments, we sample the high dynamic range illumination [DM97] at resolution $6 \times 64 \times 64$. Note that it is not necessary to sample the illumination at a higher resolution than that of the visibilities and BRDFs. Otherwise, the integral coefficients at the finest scale will be zero (refer to the generalized Haar integral coefficient theorem), and consequently wavelet coefficients at the finest scale will not contribute to the shading integral. Generally, the number of operand functions with the maximum resolution in the shading integral should be no less than two. In our system, lighting is dynamically sampled on-the-fly,

subject to wavelet transform, non-linear approximation and quadtree encoding. Compared to the visibilities and BRDFs, more wavelet coefficients are required to approximate the all-frequency lighting (up to 300 coefficients per face of the cubemap).

Rendering: The sub-linear tree-structured rendering algorithm (see subsection 3.4.3) is capable of computing the color of each vertex in any scene interactively, under arbitrary all-frequency lighting and view conditions. The algorithm iterates six times for each visible vertex to account for each face of the cubemap. Each color channel is processed independently. For each vertex, *local visibility* is readily available. BRDF is interpolated from the tabulated data. *Dynamic occlusions* are interpolated from the tabulated *global visibilities* of the neighboring objects in the scene. If a vertex lies on the virtual ground plane of an occluding object, they are bi-linearly interpolated from the four neighboring points in the occluder's planar sampling field; otherwise they are tri-linearly interpolated from the eight neighboring points in the spherical sampling field. Finally, we rasterize each object in the scene using graphics hardware. Because we have visibility field for all the objects in the scene, our system supports interactive manipulation of any object in the scene (such as cloning, translating and scaling), as shown in Figure 3.4. The maximum memory used by the system is nearly 1.5GB, which is comparable to the previous work [NRH03, NRH04].

One of the nice features of the tree-structured rendering algorithm is that the rendering speed can be controlled interactively by varying the traversal depth, which further controls the number of the wavelet coefficients used in the lighting integral. Figure 3.5 compares the effect of various traversal depths on the rendered images. A smaller depth results in faster

rendering, but filters out the high-frequency components of the generated shadows and view-dependent specularities. As traversal depth increases, more wavelet coefficients are involved, consequently we get more shadow details at the expense of longer computation time. In the figure, the overall rendering time includes rotating/encoding the lighting, interpolating BRDFs and occlusions, shading integration and the rasterization. It demonstrates a linear complexity in terms of the traversal depth of the rendering algorithm. For these images, more wavelet coefficients are used to approximate the high-frequency components of the environment lighting (St. Peter's Basilica) than that of the visibilities and BRDFs.

3.5.3 Just-In-Time Radiance Transfer

By pre-computing radiance transfer at runtime, JRT is capable of generating dynamic realistic all-frequency shadows in real-time. As mentioned in subsection 3.2.2, as long as only one operand function in the multi-function product integral varies (equation 3.9), we can apply JRT to accelerate computation. Practically this assumption is quite reasonable in the lighting design system.

We implement the efficient algorithm for the product of multiple functions (see subsection 3.4.4) to compute the dynamic radiance transfer vectors on-the-fly. Using the algorithm, these radiance transfer vectors are directly encoded in the augmented quad-trees. One may notice that the output of the algorithm may contain many small coefficients in the quad-trees.

For computational efficiency, we explicitly prune these trees by eliminating nodes with small wavelet coefficients. Note that pruning does not introduce any perceived difference between JRT rendered images and the references. By pre-computing radiance transfer at runtime, the approach effectively reduces to the geometry relighting and the image relighting [NRH03]. Note that our approach supports high-frequency materials, while the geometry relighting in [NRH03] is limited to diffuse materials. As demonstrated in the accompanying video, runtime pre-computation of the radiance transfer vectors for moving table takes 1.22s, scaling table takes 1.42s, scaling chair takes 1.56s, cloning table takes 1.86s, cloning chair takes 2.84s, and rotating the lighting takes 1.19 - 2.13s, respectively. Images are rendered at 15-35 fps. Additional examples of the composed scenes are shown in Figures 3.1, 3.6, 3.7 and 3.5.3. For these images, we use more wavelet coefficients to support intricate shadows, consequently the runtime pre-computation is longer (up to fourteen seconds). These images are rendered at 4-11 fps.

Figure 3.5.3 illustrates the realistic cast shadows by the dynamic table onto the floor and the chair. After the designer selects the table, we compute the just-in-time radiance transfer vectors for all vertices affected by the table. At each vertex, it is the product of the lighting, BRDF, local visibility and global occlusions from any neighboring object except for the table. As the designer moves the table, shading on the chair and the floor are evaluated as the dot product between the computed just-in-time transfer vectors and the dynamic occlusion with respect to the table. One may notice that shading on the table in the third and the fourth image (the reference) are slightly different. This is owing to the fact that they are updated

only after the designer completes the movement of the table. However, as shown in the figure, there is no difference between JRT generated cast shadows and the reference image. Extension to support real-time shading on the dynamic object itself remains as future work.

3.6 Conclusions and Future Work

In this chapter, we focused on the efficient computation of two basic problems: product integral of multiple functions and its dual, multi-function product. These two mathematical problems are interrelated and both reduce to the efficient computation of the integral coefficients. Analyses show that the previous recursive approach is computationally expensive and practically infeasible. We address these issues by projecting all operand functions onto the wavelet bases. We proposed a novel generalized Haar integral coefficient theorem, and developed two efficient tree-structured sub-linear algorithms for the two problems.

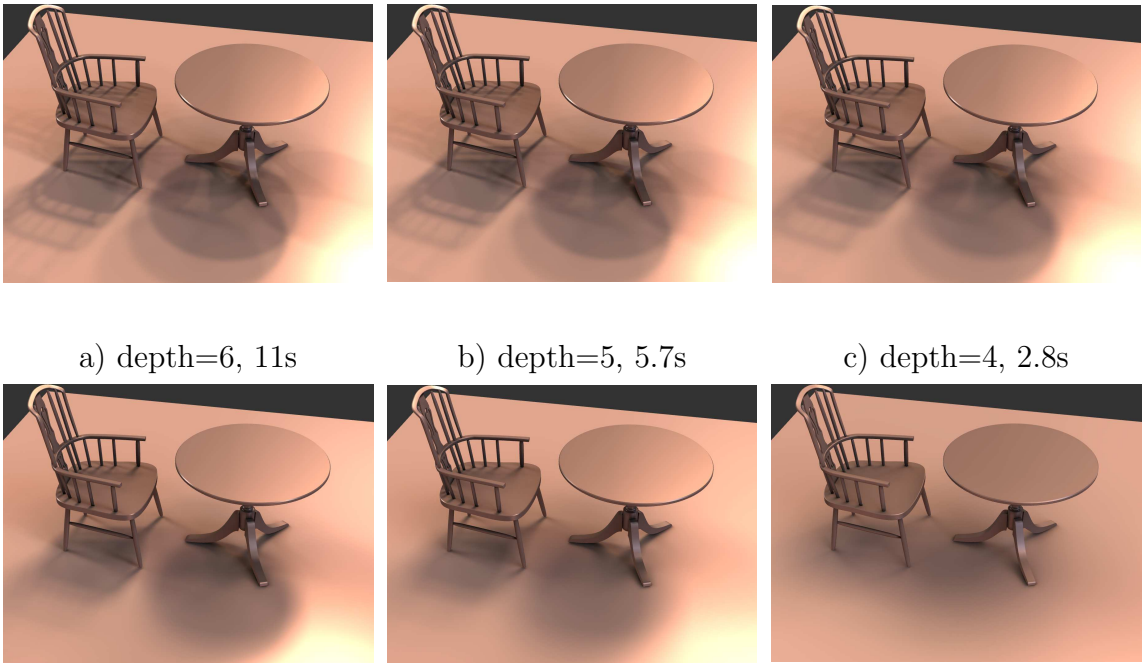
We demonstrated the practical application of the first algorithm in interactive rendering of dynamic glossy objects under distant time-variant all-frequency environment lighting and arbitrary view conditions. We represent the shading integral at each vertex as the product integral of multiple functions, involving the lighting, BRDF, local visibility and dynamic occlusions. Using the sub-linear algorithm for multi-function product integral, we rendered each frame in a few seconds on a commodity CPU. The approach is orders of magnitude faster than previous techniques.

We proposed a novel Just-in-time Radiance Transfer (JRT) technique to further accelerate shadow computation in the dynamic scenes. JRT renders realistic all-frequency shadows in real-time. As a new generalization to PRT, JRT has three main advantages. First, light transport in JRT is captured by a small radiance transfer vector that readily supports high-glossy materials. In comparison, PRT needs a prohibitively massive light transport matrix to support high-glossy materials. Although techniques have been proposed to aggressively compress this massive matrix in PRT by decreasing its dimensionality [LSS04, WTL04], compression band-limits high-glossy materials, and even the compressed dataset is still big compared to that in JRT. Second, radiance transfer in JRT is evaluated on-the-fly and it supports all-frequency cast shadows from dynamic neighboring objects. However, the light transport in PRT is only valid for a single static object. Note that pre-computing light transport for every frame in a dynamic scene using PRT is possible but prohibitively expensive. Finally, the radiance transfer vector in JRT is evaluated interactively at runtime (using the second tree-structured algorithm), while the light-transport in PRT is tabulated off-line in hours. JRT is a fast and flexible approach, and we believe that it has broad applications in computer games, 3D modelling and lighting design. We predict future extensions to this technique, for instance, in rendering dynamic objects with real-time subsurface scattering effects and extending the technique to volume rendering.

Currently the system cannot handle object rotation. This is a prevalent issue for all PRT approaches using wavelets [NRH03, NRH04]. The problem arises because the pre-computed data are defined in a global coordinate system and encoded in the wavelet domain, while

the object is rotated in a local coordinate system. As a result, the problem reduces to the non-trivial rotation in the wavelet domain. Finding efficient solutions to this problem is an important area of future work.

This chapter is a significant step towards efficiently integrating the product of multiple signals, which is of great importance to general numerical analysis and signal processing. We predict much future work in this direction. We are aware of the recent wavelet importance sampling by Clarberg et al. [CJA05] in extending two-dimensional triple product to higher-dimensional triple product. Results in the chapter can also be extended to support efficient computation of higher-dimensional tensor product. Finally, our work focuses on efficient light integration with only the direct lighting. Similar mathematical methodology may be used to incorporate indirect lighting in global illumination.



a) depth=6, 11s b) depth=5, 5.7s c) depth=4, 2.8s

d) depth=3, 1.3s e) depth=2, 0.6s f) depth=1, 0.26s

Figure 3.5: Comparison of the rendered images (resolution: 1200×900) with varying traversal depths of the rendering algorithm. The overall rendering time (including sampling/encoding the lighting, interpolating BRDFs/occlusions, shading integral and the rasterization) is also shown (in seconds), which demonstrates a linear complexity in terms of the traversal depth of the rendering algorithm. Image in a) is indistinguishable from the reference. g) is rendered without global occlusions. The maximum number of wavelets encoded in the augmented quadtrees for each face of the cubemap are: lighting (St. Peter's Basilica) 200, BRDFs 70, visibilities 70.

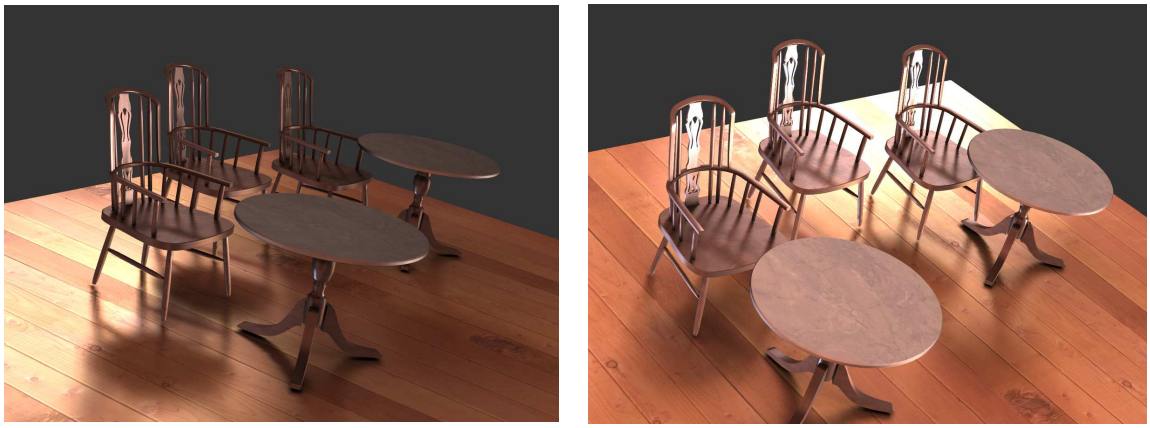


Figure 3.6: More examples of the composed scenes (resolution: 1200×900). Only view point changes (lighting: St. Peters Basilica). Note that shadows on the floor and specularities on the three chairs change dramatically with changing viewpoint. Note that here lighting is fixed. Images are rendered at $4 - 11$ fps.

Figure 3.7: More examples of the composed scenes (resolution: 1200×900). Only lighting varies (upper lighting: Grace Cathedral, lower lighting: Building). Note that view is fixed. Images are rendered at $4 - 11$ fps.

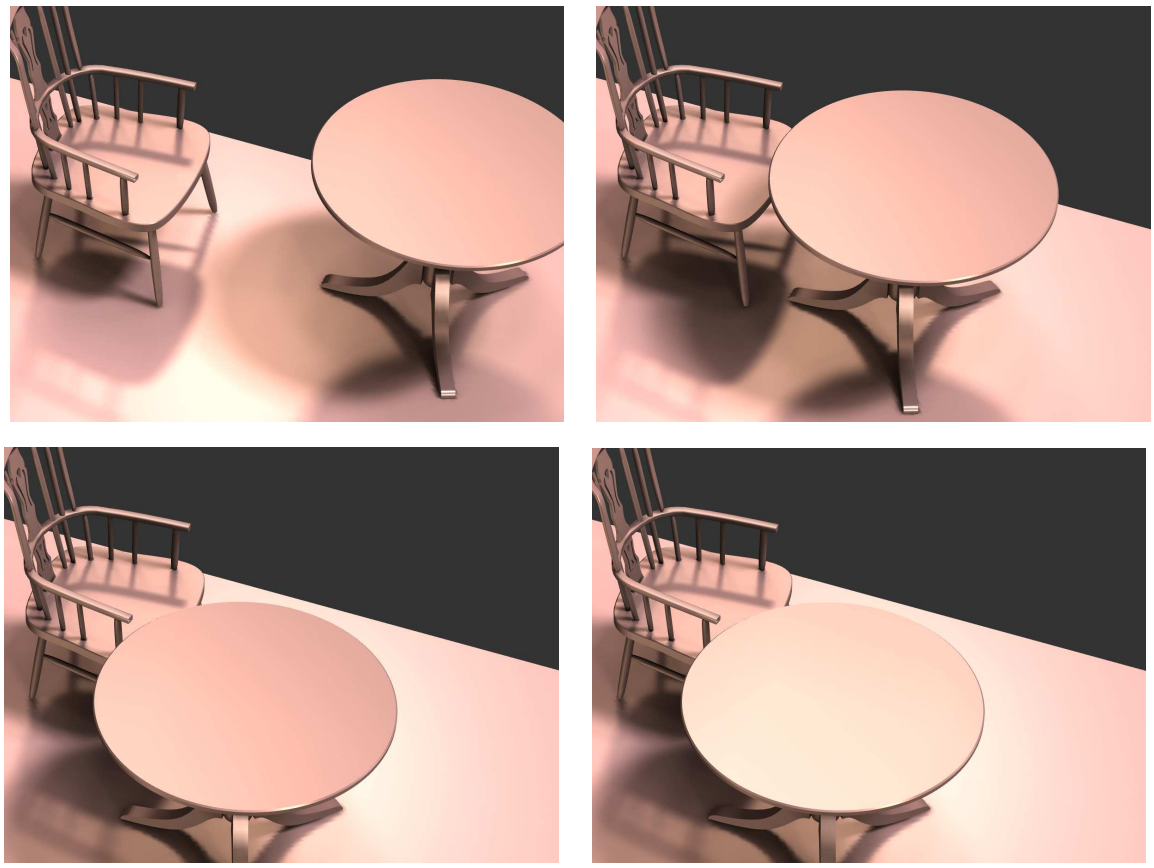


Figure 3.8: More examples of the composed scenes (resolution: 1200×900). The designer moves the table (lighting: Galileo). Note that shadows cast by the table onto the chair and the floor change. The fourth image is generated using the general rendering algorithm, while the first three images are generated using JRT. Images are rendered at nearly 15 fps.

CHAPTER 4

CONCLUSION AND FUTURE WORK

In this dissertation, we presented a new theory of efficiently computing multi-function product and product integral. We analyzed the mathematical problem in the basis domain, especially in the wavelet domain. The theoretical results have been applied in developing a set of efficient algorithms for multi-function product and product integral. We also successfully demonstrated their application in real-time rendering.

The theoretical part of this work has broad implications. In this dissertation, we concentrate on the simplest wavelet filters, Haar wavelets. One prominent question is: are there any other wavelet filters which are smooth yet more efficient in integrating the product of multiple natural signals? We are highly optimistic in the existence of such wavelet filters.

In the dissertation, we only consider the computational aspects of the light integral operation using wavelet technique. We believe similar approaches deserve further investigation in some closely related topics which are also critical to the realistic simulation of natural objects, such as compact yet accurate description of surface material appearance and surface geometry deformation.

LIST OF REFERENCES

[BBS94] Deborah F. Berman, Jason T. Bartell, and David H. Salesin. "Multiresolution Painting and Compositing." In *Proc. SIGGRAPH '94*, pp. 85–90, July 1994.

[BL81] L. C. Biedenharn and J. D. Louck. *Angualr Momentum in Quantum Physics, theory and application*. Addison-Wesley Publishing Company, 1981.

[BN76] J. F. Blinn and M. E. Newell. "Texture and reflection in computer generated images." *Commun. ACM*, **19**(10):542–547, 1976.

[CJA05] Petrik Clarberg, Wojciech Jarosz, Tomas Akenine-Moller, and Henrik Wann Jensen. "Wavelet Importance Sampling: Efficiently Evaluating Products of Complex Functions." *ACM Transactions on Graphics (SIGGRAPH '05)*, **24**(3):1166–1175, 2005.

[DAG95] Julie Dorsey, James Arvo, and Donald Greenberg. "Interactive design of complex time dependent lighting." *IEEE Computer Graphics and Applications*, **15**(2):26–36, 1995.

[Dau92] Ingrid Daubechies. *Ten Lectures on Wavelets. 2nd ed.* SIAM, 1992.

[DBB03] Philip Dutre, Philippe Bekaert, and Kavita Bala. *Advanced Global Illumination*. AK Peters Limited, 2003.

[DM97] P. E. Debevec and J. Malic. "Recovering high dynamic range radiance maps from photographs." In *Proc. SIGGRAPH '97*, pp. 369–378, 1997.

[GSC93] Steven J. Gortler, Peter Schroder, Michael F. Cohen, and Pat Hanrahan. "Wavelet Radiosity." In *Proc. SIGGRAPH '93*, pp. 221–230, 1993.

[GTR] Jinwei Gu, C. Tu, Ravi Ramamoorthi, Peter Belhumeur, Wojciech Matusik, and Shree K. Nayar. "Time-varying Surface Appearance: Acquisition, Modeling, and Rendering." *ACM Transactions on Graphics (SIGGRAPH '06)*, **25**(3).

[JF03] Doug L. James and Kayvon Fatahalian. "Precomputing Interactive Dynamic Deformable Scenes." *ACM Transactions on Graphics (SIGGRAPH '03)*, **22**(3):879–887, 2003.

[Kaj86] James T. Kajiya. "The Rendering Equation." In *Proc. SIGGRAPH '86*, pp. 143–150, 1986.

[KAJ05] Anders Wang Kristensen, Tomas Akenine-Moeller, and Henrik Wann Jensen. "Precomputed Local Radiance Transfer for Real-time Lighting Design." *ACM Transactions on Graphics (SIGGRAPH '05)*, **24**(3):1208 – 1215, 2005.

[KL05] Janne Kontkanen and Samuli Laine. "Ambient Occlusion Fields." In *Proceedings of ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games*, pp. 41–48. ACM Press, 2005.

[KLA04] Jan Kautz, Jaakko Lehtinen, and Timo Aila. "Hemispherical Rasterization for Self-Shadowing of Dynamic Objects." In *Proceedings of Eurographics Symposium on Rendering 2004*, pp. 179–184, 2004.

[KSS02] Jan Kautz, Peter-Pike Sloan, and John Snyder. "Fast Arbitrary BRDF Shading for Low-Frequency Lighting Using Spherical Harmonics." In *Proceedings of the 13th Eurographics Workshop on Rendering*, pp. 291–296, June 2002.

[LH06] Sylvain Lefebvre and Hugues Hoppe. "Perfect Spatial Hashing." *ACM Transactions on Graphics (SIGGRAPH '06)*, **25**(3), 2006.

[LK03] Jaakko Lehtinen and Jan Kautz. "Matrix radiance transfer." In *Proceedings of the 2003 symposium on Interactive 3D graphics*, pp. 59–64, 2003.

[LSS04] Xinguo Liu, Peter-Pike Sloan, Heung-Yeung Shum, and John Snyder. "All-Frequency Precomputed Radiance Transfer for Glossy Objects." In *Proceedings of Eurographics Symposium on Rendering 2004*, pp. 337–344, 2004.

[NRH03] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. "All-Frequency Shadows Using Non-linear Wavelet Lighting." *ACM Transactions on Graphics (SIGGRAPH '03)*, **22**(3):376–381, 2003.

[NRH04] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. "Triple Product Wavelet Integrals for All-Frequency Relighting." *ACM Transactions on Graphics (SIGGRAPH '04)*, **23**(3):477–487, 2004.

[NSD94] J. S. Nimeroff, E. Simoncelli, and J. Dorsey. "Efficient Re-rendering of Naturally Illuminated Environments." *5th Eurographics Rendering Workshop*, pp. 359–374, 1994.

[PVL05] Fabio Pellacini, Kiril Vidimče, Aaron Lefohn, Alex Mohr, Mark Leone, and John Warren. "Lpics: a Hybrid Hardware-Accelerated Relighting Engine for Computer Cinematography." *ACM Transactions on Graphics (SIGGRAPH '05)*, **24**(3):464–470, 2005.

[RWS] Zhong Ren, Rui Wang, John Snyder, Kun Zhou, Xinguo Liu, Bo Sun, Peter-Pike Sloan, Hujun Bao, Qunsheng Peng, and Baining Guo. "Real-time Soft Shadows in Dynamic Scenes using Spherical Harmonic Exponentiation." *ACM Transactions on Graphics (SIGGRAPH '06)*, **25**(3).

[SDS96] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgan-Kaufmann, 1996.

[Sha93] J. M. Shapiro. "Embedded Image Coding Using Zerotrees of Wavelet Coefficients." *IEEE Transactions on Signal Processing*, **SP**(41):3445–3462, December 1993.

[SHH03] Peter-Pike Sloan, Jesse Hall, John Hart, and John Snyder. "Clustered Principal Components for Precomputed Radiance Transfer." *ACM Transactions on Graphics (SIGGRAPH '03)*, **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." *ACM Transactions on Graphics (SIGGRAPH '02)*, **21**(3):527–536, 2002.

[SLS03] Peter-Pike Sloan, Xinguo Liu, Heung-Yeung Shum, and John Snyder. "Bi-scale radiance transfer." *ACM Transactions on Graphics (SIGGRAPH '03)*, **22**(3):370–375, 2003.

[SLS05] Peter-Pike Sloan, Ben Luna, and John Snyder. "Local, Deformable Precomputed Radiance Transfer." *ACM Transactions on Graphics (SIGGRAPH '05)*, **24**(3):1216–1224, 2005.

[SM06a] Weifeng Sun and Amar Mukherjee. "Generalized Wavelet Product Integral for Rendering Dynamic Glossy Objects." *ACM Transactions on Graphics (SIGGRAPH '06)*, **25**(3), 2006.

[SM06b] Weifeng Sun and Amar Mukherjee. "Inside Just-in-time Radiance Transfer." In *ACM SIGGRAPH 2006 Technical Sketch*, 2006.

[TS] Yu-Ting Tsai and Zen-Chung Shih. "All-Frequency Precomputed Radiance Transfer using Spherical Radial Basis Functions and Clustered Tensor Approximation." *ACM Transactions on Graphics (SIGGRAPH '06)*, **25**(3).

[WTL] Jiaping Wang, Xin Tong, Stephen Lin, Chao Wang, Minghao Pan, Hujun Bao, Baining Guo, and Heung-Yeung Shum. "Appearance Manifolds for Modeling Time-Variant Appearance of Materials." *ACM Transactions on Graphics (SIGGRAPH '06)*, **25**(3).

[WTL04] Rui Wang, John Tran, and David Luebke. "All-Frequency Relighting of Non-Diffuse Objects using Separable BRDF Approximation." In *Proceedings of Eurographics Symposium on Rendering 2004*, pp. 345–354, 2004.

[ZHL05] Kun Zhou, Yaohua Hu, Stephen Lin, Baining Guo, and Heung-Yeung Shum. "Precomputed Shadow Fields for Dynamic Scenes." *ACM Transactions on Graphics (SIGGRAPH '05)*, **24**(3):1196 – 1201, 2005.