Wavelet Basis Functions for Precomputed Radiance Transfer

Ian Vollick and Christian Lessig

Abstract

Precomputed Radiance Transfer (PRT) aims at computing global illumination effects such as soft shadows and object interreflection in real-time. This is accomplished by precomputing the factors of the Rendering Equation and projecting them into a suitable basis which allows to combine them efficiently at runtime. Several basis function have been proposed for PRT. Recently, Haar wavelets have been employed which provide superior results, especially for high frequency effects.
In this report, the computation of the triple product integral of lighting, BRDF and visibility in the Haar basis is detailed and an efficient implementation of this algorithm is presented. The influence of the input signal resolution and of nonlinear approximations of the basis representation of the input signals on the quality of the renderings is examined.

1 Introduction

Computing global illumination effects in real-time is a difficult task. Especially effects which require a large number of samples such as soft shadows are hard to compute efficiently. The problem is described by the Rendering Equation [Kajiya 1986]. For non-emmissive surfaces it is defined as

$$B(x, \omega_0) = \int_\Omega L(x, \omega) V(x, \omega) \rho(x, \omega, \omega_0)(\omega \cdot n(x)) d\omega, \tag{1}$$

where B is the reflected light at a surface location x in the scene, ω_0 is the viewing direction and the integral is computed over all possible direction of incident lighting Ω. For convenience we absorb the cosine term of the incident lighting, $(\omega \cdot n(x))$, into the BRDF. The binary visibility function $V(x, \omega)$ is 0 if the surface point x is in shadow for light incident from direction ω, otherwise it is 1. For a scene with n objects, the visibility term can be written as the product of n dynamic occlusion fields O_i [Sun 2006]

$$V(x, \omega) = O_1(x, \omega) \prod_{i=2}^{n} O_i(x, \omega), \tag{2}$$

where O_1 is the local visibility at x due to self occlusion, and each of the O_i represents the potential occlusion by object i in the scene. Inserting Equation 2 into Equation 1 yields

$$B(\omega_0) = \int_\Omega L(\omega) O_1(\omega) \prod_{i=2}^{n} O_i(\omega) \rho(\omega, \omega_0) d\omega. \tag{3}$$

Equation 3 does no longer depend on x because only a fixed surface location and distant illumination are considered. The integrand for computing the reflected light is now a product of $n+2$ functions $F_i(\omega)$ — the lighting, the BRDF and the n dynamic occlusion fields. For PRT, these factors are precomputed and combined at runtime.

To perform the runtime computations efficiently, all factors $F_i(\omega)$ are projected into an orthonormal basis $\mathfrak{B}$

$$F_i(\omega) = \sum_{k=1}^{M} f_{i,k} b_k(\omega), \tag{4}$$

where the $b_k(\omega)$ are the M basis functions of $\mathfrak{B}$ and the $f_{i,k}$ are the basis coefficients of $F_i(\omega)$ in $\mathfrak{B}$.
The integral of $n+2$ functions represented in a common basis is

$$B(\omega) = \int \prod_{i=1}^{n+2} \sum_{k_i=1}^{M} f_{i,k} b_k(\omega) d\omega. \tag{5}$$

Note that only the basis functions depend on ω. Therefore, the basis coefficients can be factored out of the integral yielding

$$B(\omega) = \prod_{i=1}^{n+2} \sum_{k=1}^{M} f_{i,k_j} \int b_k(\omega) d\omega. \tag{6}$$

The product of n basis functions $b_0(\omega), b_1(\omega), ..., b_n(\omega)$, the *integral coefficient* $C_n(b_0, b_1, ..., b_n)$, is given by

$$C_n(b_1, b_2, ..., b_n) = \int \prod_{k=1}^{n} b_k(\omega) d\omega. \tag{7}$$

Multiplying out the product and rearranging the coefficients yields

$$B(\omega) = \sum_{k_1=1}^{M} \sum_{k_2=1}^{M} ... \sum_{k_{n+1}=1}^{M} \prod_{i=1}^{n+2} f_{i,k} C_{n+2}(b_{k_1}, b_{k_2}, ..., b_{k_{n+1}}). \tag{8}$$

The integral in its basis representation can therefore be determined by computing the sum of all products of basis function coefficients and integral coefficients.
Equation 7 and Equation 8 show that the integral coefficients C_n solely depend on the basis function but not on the signals. Thus, the C_n can be precomputed. Moreover, for many bases a significant number of integral coefficients are 0. For a fixed basis $\mathfrak{B}$, this sparsity of the C_n can be exploited to optimize the computation of Equation 8.
However, even with a moderate number of factors in the product the computation of $B(\omega)$ remains expensive because the sparsity in the integral coefficients decreases rapidly with increasing n. Therefore, the integrand is usually reduced to a product of two or three factors.

Most PRT techniques use a two factor integral to compute $B(\omega)$ efficiently. Except [Zhou 2005] and [Sun 2006], all approaches assume a static scene so that the configuration of the occlusion fields is fixed and can be represented by one factor $V(x, \omega)$. Combining the visibility and the BRDF yields the transport operator

$$T(\omega) = V(\omega)\rho(\omega, \omega_0). \tag{9}$$

Equation 3 then reduces to a two factor product

$$B(\omega_0) = \int_\Omega L(\omega)T(\omega, \omega_0)d\omega. \tag{10}$$

For an orthonormal basis $\mathfrak{B}$, computing Equation 10 in its basis representation is efficient because the integral coefficients reduce to Kronecker deltas, i.e.

$$C_2(b_{k_0}, b_{k_1}) = \delta_{k_0,k_1} = \begin{cases} 1 & \text{if } k_0 = k_1 \\ 0 & \text{otherwise} \end{cases}, \forall (k_0, k_1), k_0, k_1 \in \mathfrak{B}.$$

$B(\omega)$ is therefore the dot product of the basis coefficients

$$\begin{aligned} B(\omega) &= \sum_{k_0=1}^{M} \sum_{k_1=1}^{M} C_2(b_{k_0}, b_{k_1}) f_{0,k_0} f_{1,k_1} && (11) \\ &= \sum_{k_0=1}^{M} \sum_{k_1=1}^{M} \delta(k_0, k_1) f_{0,k_0} f_{1,k_1} && (12) \\ &= f_0 \cdot f_1, && (13) \\ & && (14) \end{aligned}$$

where f_0 and f_1 are the vectors containing all basis coefficients of $F_0(\omega)$ and $F_1(\omega)$, respectively. The computation of Equation 10 in an orthonormal basis with only diffuse materials reduces to

$$B(\omega) = l \cdot t, \tag{15}$$

where l and t are the vectors containing all basis coefficients of $L(\omega)$ and $T(\omega)$, respectively. For view depend effects such as glossy BRDFs t becomes a matrix.
Spherical Harmonics (SH) [MacRobert 1948] are a popular basis for computing Equation 15. This basis can efficiently represent diffuse, low frequency lighting yielding real-time frame rates even for complex scenes [Sloan 2002] [Sloan 2003a] [Sloan 2003b]. However, for high frequency lighting such as sharp shadows or glossy materials an intractable number of SH coefficients would be necessary [Ng 2003]. Piecewise bi-linear functions defined on the hemisphere [Lehtinen 2003], hemispherical basis function derived from associated Legendre polynomials [Gautron 2004] and different clustering techniques [Sloan 2003b] have been proposed to improve or replace Spherical Harmonics. However, none of these approaches is efficient for high frequency effects.
Ng et al. [Ng 2003] compute Equation 15 in the Haar basis. They demonstrate that Haar wavelets can efficiently represent both low and high frequency effects. However, computing $B(\omega)$ is still more expensive than for low frequency effects and interactive frame rates are only possible with either fixed view or only diffuse materials.

To be able to change the BRDF at runtime, Ng et al. [Ng 2004] do not combine visibility and BRDF. Equation 3 then becomes

$$B(\omega_0) = \int_\Omega L(\omega)V(\omega)\rho(\omega, \omega_0)d\omega. \tag{16}$$

In contrast to Equation 10, no interreflection between objects can be taken into account in this case. Moreover, computing Equation 16 in its basis representation in an arbitrary orthonormal basis $\mathfrak{B}$ does not simplify as in Equation 15. The triple product integral is therefore

$$B(\omega) = \sum_{k_0=0}^{M}\sum_{k_1=0}^{M}\sum_{k_2=0}^{M} C_3(b_{k_0}, b_{k_1}, b_{k_2})L_{k_0}V_{k_1}\rho_{k_2}. \tag{17}$$

Here, the integral coefficients $C_3(b_{k_0}, b_{k_1}, b_{k_2})$ are denoted as *tripling coefficients*. Ng et al. [Ng 2004] show that the tripling coefficients are sparse for many common bases such as Dirac impulses, the 2D Fourier Series, Spherical Harmonics and Haar wavelets.
For Dirac impulses, the computation of Equation 17 is efficient because the C_3 are generalized Kronecker deltas

$$C_3(b_{k_0}, b_{k_1}, b_{k_2}) = \delta_{k_0,k_1,k_2} = \begin{cases} 1 & \text{if } k_0 = k_1 = k_2 \\ 0 & \text{otherwise} \end{cases}.$$

However, Dirac impulses, or point samples, provide very poor compression because they localize only in space but not in frequency. Therefore, a very large number of coefficients are necessary to represent low-frequency lighting effects such as soft shadows [Ng 2004]. Ng et al. also show that the Haar basis is the most efficient basis form the set mentioned above. In fact, Haar wavelets are well suited for nonlinear approximation so that only the $k\%$ largest coefficients have to be to compute Equation 17. This yields an algorithm with sublinear time complexity. Usually about 0.1 - 1% of the coefficients are sufficient to achieve results which are visually almost indistinguishable from the full solution [Ng 2004]. Because the integral coefficients are far less sparse than for the two product integral rendering still takes several minutes.

In the remainder of the report, an derivation and explanation of using Haar wavelets for efficiently computing n factor and triple product integrals is provided, detailing some of the points left open in the original papers by Ng et al. [Ng 2004] and Sun and Mukgerjee [Sun 2006]. We also analyze the influence of different input signal resolutions and of nonlinear approximation on the quality of the renderings.
The next section presents an introduction into Haar wavelets and their properties. This is followed by an explanation of the use of the Haar basis for PRT. The report concludes with a presentation of results of our implementation and a discussion of areas for future work.

2 Haar Wavelets

2.1 Multiresolution Analysis

It is often desirable to work with signals on a number of scales. For example, one may want to separate high frequency noise from a smooth signal or change the low frequency structure of a signal while maintaining the high frequency characteristics. A multiresolution analysis of the signal is often used to facilitate these sorts of operations. It is obtained by representing the signal as a linear combination of basis functions. Each basis function

is defined so that it captures only certain spatial or frequency-dependent characteristics of the signal. More formally, we seek a basis $\mathfrak{B}$ with basis functions $b_i(t)$ such that a signal $y(t)$ can be represented as

$$y(t) = \sum_{i=0}^{N} f_i b_i(t). \tag{18}$$

One very commonly used basis is the Fourier basis which yields a frequency analysis of the input signal. However, the Fourier basis does not localize the signal in space and a Fourier representation of a signal is rarely sparse. That is, in Equation 18, a large number of coefficients f_i are significantly nonzero. In this report we are interested in finding a basis $\mathfrak{B}$ which yields a sparse representation. This may be accomplished with wavelet bases.

Wavelet bases have become a popular alternative for multiresolution analysis. Like the Fourier basis, wavelets allow a perfect reconstruction of the original signal and all basis functions are scaled copies of a small set of parent functions. For the Fourier transform these functions are $\cos nx$ and $\sin nx$. For wavelets a variety of different basis functions exist. One crucial difference between the Fourier and wavelet basis functions is that the wavelet basis functions are scaled *and* translated versions of the parent basis functions. In contrast, the Fourier basis functions are only scaled versions of $\cos nx$ and $\sin nx$. Another difference is that the effective support of wavelet basis functions, i.e. the domain over which they are significantly nonzero, is finite. These differences allow a wavelet basis to localize a signal in space *and* frequency. Wavelet decompositions also tend to be sparse for typical signals and so wavelet bases are useful for compression or to accelerate algorithms based on basis function coefficients [Stollnitz 1996].
The simplest wavelet basis is the Haar basis introduced by Alfred Haar in 1909 [Stollnitz 1996].

2.2 1D Haar Transform

2.2.1 Introduction

Consider a discrete signal f with 2^5 elements, for example the pixel values of one row of an image. The signal is decomposed in to a hierarchy of successively coarse approximations H, with $H = \{f^l\}_{i=0}^4$, where f^l is an approximation of f of length 2^l.
A simple way of obtaining f^4 is to take the average of each pair of adjacent values in f. The approximation f^4 is therefore given by

$$f^4[i] = \frac{1}{2}(f[2i] + f[2i+1]). \tag{19}$$

The information that is lost due to averaging can be formalized as

$$f_d^4[i] = \frac{1}{2}(f[2i] - f[2i+1]). \tag{20}$$

Given f_d^4 and f^4, we can restore f as follows

$$f[2i] = f^4[i] + f_d^4[i] \tag{21}$$

$$f[2i+1] = f^4[i] - f_d^4[i]. \tag{22}$$

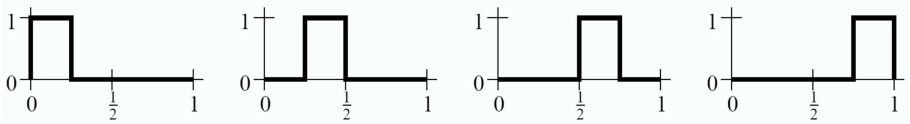

Figure 1: $\phi^{(2,0)}, \phi^{(2,1)}, \phi^{(2,2)}$ and $\phi^{(2,3)}$ (Courtesy [Stollnitz 1996])

Thus, Equation 19 and Equation 20 describe a lossless decomposition of f into two parts. The approximation $f^4[i]$ captures the low frequency content of the signal f while $f_d^4[i]$ captures the high frequency content. It should be noted that the low frequency or average part of a signal usually contains most of the significant information. For example removing the high frequency part of an audio recording of a speech will produce an audio signal that may sound unnatural, but in which the spoken words are still understandable. On the other hand, removing the low frequency part will result in an incomprehensible signal [Stollnitz 1996].

In the same way f has been decomposed into f^4 and f_d^4, we can decompose the average signal f^4, into two signals f^3 and f_d^3. This process can be repeated until f^0 is reached. Now given f^l and f_d^l we can reconstruct f^{l+1} as follows

$$f^{l+1}[2i] = f^l[i] + f_d^l[i] \tag{23}$$

$$f^{l+1}[2i+1] = f^l[i] - f_d^l[i]. \tag{24}$$

It follows that any of the approximate signals f^l can be reconstructed using f^0 and $f_d^1, \ldots, f_d^4$ by repeated application of Equation 23 and Equation 24. Since we can reconstruct f^4 using $f_d^1, \ldots, f_d^4$, we can also reconstruct f by Equation 21 and Equation 22. Hence, $f^0, f_d^1, \ldots, f_d^4$ are the coefficients of a lossless decomposition of f. Moreover, for typical signals f, many of the detail coefficients f_d^l will be zero or close to zero. This is because the detail signals capture differences in adjacent elements of the approximations f^i and for typical signals most adjacent values are strongly correlated.

For many applications, the coefficients $f^0, f_d^1, \ldots, f_d^4$ are represented in one vector f'. This vector has the same number of elements as f. The signal f' is an unnormalized representation of the 1D discrete Haar transform of f. The vector f' may be compressed using RLE. Additionally, also coefficients which are close to zero can be set to zero before performing the compression which improves its efficiency. This is called *nonlinear approximation*.

2.2.2 Haar Wavelets

In order to better understand the notion of a Haar wavelet, we will reformulate the transform in terms of hierarchical vector spaces rather than a hierarchy of signals. The derivations below are based on [Stollnitz 1996].

Let V^n represent the subspace of all piecewise constant functions over 2^n, non-overlapping subintervals of $[0, 1)$ of equal size. An orthonormal basis for V^n is $\{\phi^{(n,i)}\}_{i=0}^{2^n-1}$, where $\phi^{(l,i)}(t) := \sqrt{2^l}\phi(2^l t - i)$ and

$$\phi(t) := \begin{cases} 1 & \text{if } t \in [0,1) \\ 0 & \text{otherwise} \end{cases}.$$

The basis functions $\phi^{(l,i)}(t)$ are called scaling functions. They are scaled and translated copies of the function $\phi(t)$ which is called the father wavelet. The parameter l governs the scale of the function, and the parameter i controls its translation. The basis functions for V^2 are illustrated in Figure 2.2.2.

The function $\phi(t)$, the basis function for V^0, can be written as a linear combination of the basis functions for V^1

$$\begin{aligned}\phi(t) &= \frac{1}{\sqrt{2}}\left(\phi^{(1,0)}(t) + \phi^{(1,1)}(t)\right) \\ &= \phi(2t) + \phi(2t-1).\end{aligned}$$

This is true since

$$\phi(2t) := \begin{cases} 1 & \text{if } t \in [0, 0.5) \\ 0 & \text{otherwise} \end{cases}$$

and

$$\phi(2t-1) := \begin{cases} 1 & \text{if } t \in [0.5, 1) \\ 0 & \text{otherwise} \end{cases} .$$

This means that $\phi(t) \in V^1$. Since every function $g \in V^0$ is a multiple of $\phi(t)$, g is also an element of V^1. This implies that $V^0 \subset V^1$. Similarly, it can be shown that $V^1 \subset V^2$, $V^2 \subset V^3$ and so on. Hence, the vector spaces V^i in which the approximations lie are nested and form a hierarchy.

A discrete signal f is equivalent to a piecewise constant function over 2^5 equally sized subdivisions of $[0, 1)$. Therefore f is an element of V^5. Let f^n be the projection of f onto V^n. Since the basis for V^l is orthonormal, this projection is

$$f^l(t) = \sum_{i=0}^{2^l} f_i \phi^{(l,i)}(t),$$

where the coefficients f_i are given by $\left\langle f, \phi^{(i,j)} \right\rangle$. Here, $\langle u, v \rangle := \int_0^1 u(x)v(x)dx$ is the standard inner product. All coefficients of the hierarchical decomposition can again be represented in one vector

$$\{\left\langle f, \phi^{(0,0)} \right\rangle, \left\langle f, \phi^{(1,0)} \right\rangle, \left\langle f, \phi^{(1,1)} \right\rangle, \ldots, \left\langle f, \phi^{(4,2^4-1)} \right\rangle\}.$$

As mentioned above, these coefficients are not likely to be sparse.

Now, let W^n be a subset of V^{n+1} such that $W^n \neq V^{n+1}$, and $W^n \cup V^n = V^{n+1}$. Wavelets are defined as a linearly independent set of functions spanning W^n [Stollnitz 1996]. The wavelet basis functions corresponding to the vector spaces V^n above are known as the Haar wavelet basis functions. These basis functions can be defined to be orthonormal, and so W^n is the orthogonal complement of V^n in V^{n+1}. The normalized Haar basis functions for W^n are $\{\psi^{(n,i)}\}_{i=0}^{2^n-1}$, where $\psi^{(l,i)}(t) := \sqrt{2^l}\psi(2^l t + i)$ with

$$\psi(t) = \begin{cases} 1 & \text{if } t \in [0, 0.5) \\ -1 & \text{if } t \in [0.5, 1) \\ 0 & \text{otherwise} \end{cases} .$$

The functions ϕ and ψ, are referred to as the father and mother wavelets, respectively. Also, the bases for the various V^i are referred to as scaling functions.

Given that $W^n \cup V^n$ span V^{n+1}[Stollnitz 1996], and hence $V^0 \cup W^0 \cup \cdots \cup W^n$ span V^{n+1}, the basis functions for V^i for $i > 0$ are linear combinations of wavelets and the father wavelet. This implies that any function f_i can be reconstructed using f_0 and the projection of f onto the wavelet spaces W^i. Therefore, to represent the hierarchical decomposition one only needs to retain the vector of coefficients f' resulting from the projection of f onto $V^0, W^0, \ldots, W^n$. Specifically,

$$f' = (\left\langle f, \phi^{(0,0)} \right\rangle, \left\langle f, \psi^{(0,0)} \right\rangle, \left\langle f, \psi^{(1,0)} \right\rangle, \ldots, \left\langle f, \psi^{(4,2^4-1)} \right\rangle).$$

This result is equivalent to the one obtained in Section 2.2.1. Since the basis functions are normalized, f' is referred to as the normalized Haar transform of f, or simply the Haar transform of f. Analogous to the unnormalized case, many of the coefficients will be zero or close to zero.
In general, if $g \in V^n$ then $H(g)$, the Haar transform of g, is

$$H(g) = (\left\langle g, \phi^{(0,0)} \right\rangle, \left\langle g, \psi^{(0,0)} \right\rangle, \left\langle g, \psi^{(1,0)} \right\rangle, \ldots, \left\langle g, \psi^{(n,2^n-1)} \right\rangle).$$

2.2.3 Matrix Representation

The Haar transform is a linear operations. Therefore, if $g \in V^n$ then there exists a matrix T such that $H(g) = T(V^n(g))$, where $V^n(g)$ denotes the coordinates g in V^n. The columns of T are given by the Haar transforms of the basis functions for V^n [Jensen 2000]. Therefore the size of T depends on n. For example, if $n = 2$ and we use the unnormalized Haar transform, we have

$$T = \begin{pmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} \\ \frac{1}{2} & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & -\frac{1}{2} \end{pmatrix}.$$

Note that the inverse Haar transform used to reconstruct the original signal is simply T^{-1}. Although it is mathematically convenient to work with the matrix representation of the wavelet transform, it is not used in practice since T becomes very large as g grows in length. Instead, efficient implementations make use of the fast wavelet transform (provided the wavelets are orthogonal), or the filter or lifting scheme representations of the transform.

2.2.4 Filter Representation

Matlab uses convolution of filters with the discrete signal g of length 2^n to obtain g'. Notice that if g is convolved with the vector $(1/\sqrt{(2)}, 1/\sqrt{(2)})$, one obtains a signal of length $2^n - 1$ consisting of the averages of every pair of values. Taking the odd-indexed elements of this signal gives g^{n-1}. Similarly, the odd-indexed values from the convolution of g with $(1/\sqrt{(2)}, -1/\sqrt{(2)})$ yields g_d^{n-1}. This process may be repeated n times to obtain g'. The vectors $(1/\sqrt{(2)}, 1/\sqrt{(2)})$ and $(1/\sqrt{(2)}, -1/\sqrt{(2)})$ are often written in an unnormalized form: $[1, 1]$ and $[1, -1]$ [1]. In this form they are called filter tab vectors.

[1]Matlab uses the filter tab $[-1, 1]$. However, because we did not use the Wavelet Toolbox, our implementation follows [Stollnitz 1996].

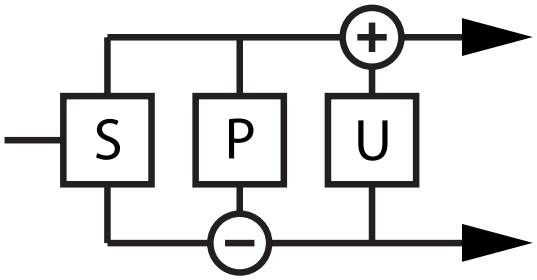

Figure 2: Standard diagram of a one-step lifting scheme.

2.2.5 Lifting

Lifting was first proposed by W. Sweldens as another method for performing the discrete wavelet transform [Jensen 2000]. One lifting step (in its basic form) has three parts:

1. **Split** The input, a discrete signal g, is split into two halves, s and d, where s contains the odd-indexed elements of g, and d contains the even.
2. **Predict** The values of s are used to "predict" the values of d and the values of d are altered to reflect how they differ from the prediction.
3. **Update**. Finally, the altered version of d is used to update the signal s to maintain a desired property. Usually this is done to ensure that s and g share the same mean value.

The lifting scheme is often described graphically as in Figure 2. In general, one lifting step may involve many predictions and updates as well as uniform scalings of s and d. A hierarchical decomposition using the Lifting Scheme is performed by using the output s signal from one lifting step as the input signal for the next. The final result of lifting is the concatenation of all outputs from all lifting steps. Lifting is memory efficient since the transform can be computed in place. Also, since predictions, updates and scales can be reversed, it is trivial to obtain an inverse lifting transform.

For the discrete Haar transform the prediction is that the signal is constant [Jensen 2000], i.e. it is expected that $s[i]$ equals $d[i]$. Consequently, in the prediction step $d[i]$ is set to $d[i] - s[i]$. In the update step, $s[i]$ is set to $s[i] + (d[i]/2)$ in order for s and g to have the same mean. This lifting step is repeatedly applied to the signal s from the previous lifting step until the output is of length one. At this point, the concatenation of s with the detail coefficients d from all steps gives an unnormalized discrete Haar transform with all but the first coefficient scaled by a factor of 2. Normalization can be done by uniformly scaling s and d.

2.3 2D Haar Transform

The simplest way to perform the Haar transform on a 2D discrete signal a 1D transform on the columns. This is known as the standard decomposition such as an image is to first perform a 1D transform on the rows and then perform a 1D transform on the columns. This is known as the standard decomposition [Stollnitz 1996]. The non-standard decomposition involves performing one step of the Haar transform alternately on the rows and columns. The non-standard transform is used in most applications because it better adapts to the 2D domain. The filter representation of the non-standard transform consists of 4 filter tab matricies

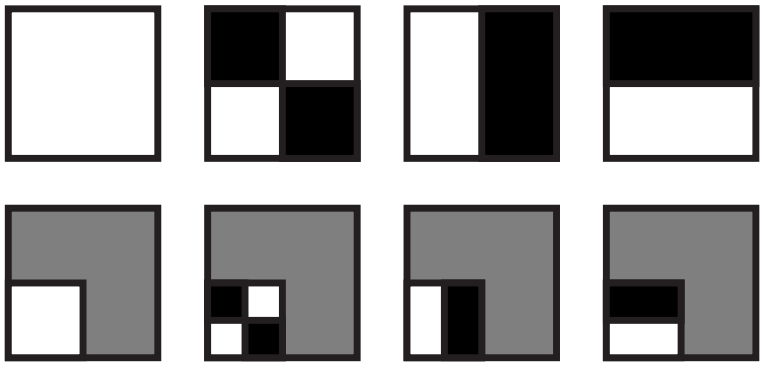

Figure 3: Visualization of $\phi^{(0,0,0)}$, $\psi_{11}^{(0,0,0)}$, $\psi_{10}^{(0,0,0)}$, $\psi_{01}^{(0,0,0)}$, $\phi^{(1,0,0)}$, $\psi_{11}^{(1,0,0)}$, $\psi_{10}^{(1,0,0)}$, and $\psi_{01}^{(1,0,0)}$. Ignoring normalization, black squares indicate regions where the functions equal -1, white 1, and grey 0.

1	1
1	1

,

1	-1
1	-1

,

1	1
-1	-1

, and

1	-1
-1	1

.

Convolving with the first filter gives the coarse approximation coefficients. Convolving with the second gives the detail coefficients corresponding to horizontal differences in the 2D signal. Similarly, convolving with the third gives those coefficients corresponding to vertical differences in the 2D signal. The fourth filter tab accounts for diagonal differences. The coefficients obtained by convolving with these filters correspond again to the coefficients obtained by projecting the image onto basis functions. In order to project the image, we will treat it as a piecewise constant function over equal-sized, square subdomains of $[0,1) \times [0,1)$. Just as in the case of 1D signals, the approximations of this image will lie in a hierarchy of vector spaces, and these vector spaces will be spanned by a hierarchy of wavelets.

The wavelet basis functions of the 2D Haar transform are

$$\begin{aligned}
\psi_{01}^{(l,i,j)} &= 2^l \phi(2^l x + i)\psi(2^l y + j) \\
\psi_{10}^{(l,i,j)} &= 2^l \psi(2^l x + i)\phi(2^l y + j) \\
\psi_{11}^{(l,i,j)} &= 2^l \psi(2^l x + i)\psi(2^l y + j),
\end{aligned}$$

where l denotes the level of the wavelet and (i, j) indicate the position of the wavelet. The scaling functions are given by

$$\phi^{(l,i,j)} = 2^l \phi(2^l x + i)\phi(2^l y + j)$$

Some examples of these wavelets are shown in Figure 2.3. Since these wavelets are based on the product of 1D Haar wavelet basis functions, they are referred to as tensor-product wavelets.

Unlike the standard and non-standard decompositions, there are some wavelet transformations that are truly 2D. That is, they cannot be expressed as a sequence of 1D wavelet

transform steps and corresponding basis functions are therefore not products of 1D basis functions. One example is the quincunx transformation which is based on the Lifting scheme [Jensen 2000].

2.4 Special Properties of Haar Wavelets

Haar wavelets exhibit a large set of desirable properties. Specifically,

- **Orthonormality** For orthogonal wavelets, the wavelet transform can be performed in linear time using the Fast wavelet transform [Cody 1993].
- **Symmetry** Symmetry is a valuable property for many applications [Stollnitz 1996]. For example, it means that the wavelet will be better at maintaining time localization [Jensen 2000].
- **Minimal Support** Haar wavelets have compact support. Moreover, this support is smaller than that of any other wavelet. The efficiency of many computations depends on the size of the support of a basis.
- **Maximal Vanishing Moments** The number of vanishing moments of a wavelet ψ is the highest number k such that for all $0 \leq j < k, \int_0^1 \psi(x)x^j dx = 0$ [Stollnitz 1996]. Haar wavelets have one vanishing moment, the highest number possible given their support size. Having k vanishing moments means that the wavelet will have a very sparse representations of signals which are polynomials of degree less than or equal to k [Mathworks].

Haar wavelets are unique in that they are the only wavelets which are at once symmetric, orthonormal and have finite support. Unfortunately, Haar wavelets are neither smooth nor continuous, and cannot be differentiated.

2.5 Other Wavelets

Haar wavelets are only one of many wavelet bases. In general, wavelet bases fall into three main categories.

2.5.1 Orthonormal Wavelets

There are three requirements for wavelet basis to be an orthogonal multiresolution basis [Stollnitz 1996].

1. The basis functions for each approximation space V^i, the scaling functions, must be orthogonal.
2. The basis functions for each wavelet space W^i, the wavelets, must be orthogonal.
3. Every wavelet basis function must be orthogonal to every scaling basis function defined at a coarser scale.

It is possible, through scaling, to convert an orthogonal basis into an orthonormal basis. The Daubechies wavelet bases, of which Haar is a special case, are all examples of orthogonal wavelet bases [Wikipedia a].

2.5.2 Semiorthogonal Wavelets

Like orthogonal wavelets, semiorthogonal wavelets are orthogonal to coarser scaling functions, but are not necessarily orthogonal to each other [Stollnitz 1996]. For some combinations of properties such as smoothness, compact support, and symmetry, one must resort to semiorthogonal wavelets [Stollnitz 1996]. One example of semiorthogonal wavelets are the spline wavelets.

2.5.3 Biorthogonal Wavelets

Biorthogonal wavelets differ from semiorthogonal wavelets in that they are not orthogonal to coarser scaling functions, or to each other. The scaling functions also need not be orthogonal. One useful feature of biorthogonal wavelets is that they can be constructed through lifting [Stollnitz 1996].

It is possible to convert any basis to an orthonormal basis using Gram-Schmidt normalization. Performing Gram-Schmidt normalization on wavelets and scaling functions gives their respective dual bases. Biorthogonal wavelet bases are such that [Stollnitz 1996]:

1. The dual wavelet bases are orthogonal to the scaling function bases.
2. The dual scaling function bases are orthogonal to the wavelet bases.

3 Precomputed Radiance Transfer using Haar Wavelets

In this section different bases for PRT are evaluated and it is shown that the Haar basis is well suited for computing an n factor product integral such as Equation 3. The discussion in this section is mainly based on [Ng 2004] and [Sun 2006].

3.1 Comparison of Basis Functions for PRT

Computing an n factor product integral in a basis representation is efficient because the integral coefficients C_n are sparse, i.e. most of them are zero or close to zero (cf. Section 1). However, the degree of sparsity depends on the chosen basis. Additionally, bases also differ with respect to other desirable properties such as the possibility to perform nonlinear approximations. In the following, we therefore analysis the efficiency of different bases for computing the n factor product integral.

General Basis A general basis is efficient for representing a signal with a small number of coefficients. The basis functions are chosen depending on the input signal. For example, Liu et al. [Liu 2004] use Singular Value Decomposition (SVD) to project the BRDF into a custom basis. The singular values are the basis coefficients and the basis functions are represented by the rows of the two triangular matricies resulting from the SVD. Liu et al. use nonlinear approximation to reduce the storage requirements and the runtime computations. They show that the 10 largest singular values are sufficient to achieve the same accuracy than with 6-th or 7-th order spherical harmonics which require 36 and 49 basis coefficients, respectively.
However, the basis depends on the input signal and is not known a priori. Therefore, the sparsity in the integral coefficients, if it exists, cannot be employed to make the computations more efficient. Thus, the integral has to be computed by brute force yielding a time complexity of $O(M^n O_{ic}^{(n)})$, where $O_{ic}^{(n)}$ is the complexity to compute a n-th order integral coefficient.

Dirac Basis The representation of a signal S in the Dirac basis δ is a discretized version of S. The Dirac basis functions have minimal local support and are non-overlapping yielding integral coefficients C_n^δ which are generalized Kronecker deltas,

$$C_n^\delta(b_{k_1}, b_{k_2}, ..., b_{k_n}) = \delta_{b_{k_1}, b_{k_2}, ..., b_n} = \begin{cases} 1 & \text{if } k_1 = k_2 = ... = k_n \\ 0 & \text{otherwise} \end{cases} . \quad (25)$$

The complexity for computing the n factor product integral in the Dirac basis is $O(Mn)$. However, this basis only localizes in the spatial domain and not in the frequency domain

Basis	Time Complexity	Comment
General basis	$O(M^n O_{IC})$	No a priori knowledge can be exploited
Dirac basis	$O(Mn)$	Nonlinear approximation not possible.
Recursive spherical harmonics	$O(M^{5/2}n)$	For low frequency lighting only $O(m^{5/2}n)$
Haar basis (recursive)	$O(Mn)$	Nonlinear approximation cannot be exploited
Haar basis (tree structured)	$O(mn)$	

Table 1: Comparison of computing the n factor product integral in different basis functions with M basis coefficients, m denotes the number of coefficients retained after nonlinear approximation and $m \ll M$.

[Ng 2004]. Thus, a large number of coefficients has to be retained to capture low frequency effects. Using nonlinear approximation in the Dirac basis therefore leads to significant artifacts.

Spherical Harmonics Spherical Harmonics (SH) [MacRobert 1948] are a basis function defined on the sphere and the equivalent of the Fourier series in this domain. The n factor product integral in the SH basis can be evaluated recursively [Sun 2006]. Then, at a single time only two factor products have to be computed. Ng et al. [Ng 2004] show that computing the two factor product using Spherical Harmonics has a complexity of $O(M^{5/2})$. Computing the n factor product has therefore a complexity of $O(M^{5/2}n)$. In practice, often only low order Spherical Harmonics are used, i.e. $m \ll M$, yielding a complexity of $O(m^{5/2})$. However, using only a very small number of SH coefficients significantly bandlimits the effects which can be represented.

Haar Basis, Recursive Computation Computing a two factor product integral in the Haar basis has a complexity of $O(M)$. Recursively evaluating the n factor product has therefore a complexity of $O(Mn)$. Because wavelets are localized in both, the space and the frequency domain, nonlinear approximation can be employed efficiently yielding an algorithm with sub-linear time complexity.
However, using nonlinear approximation is expensive because it has to be performed at runtime for each two factor product which has been computed. Thereby sorting the coefficients and selecting the $k\%$ largest has a best case complexity of $O(n \log n)$. Thus, it is not applicable for the efficient computation of the n factor product integral. The complexity to compute the n factor product in the Haar basis using a recursive computation therefore remains $O(Mn)$.

Haar Basis, Tree-based Computation Sun et al. [Sun 2006] propose a tree-structured Haar integral algorithm to compute the n factor product integral. This algorithm exploits the predetermined sparsity in the integral coefficients yielding a complexity of $O(mn)$, with $m \ll M$. A detailed description of the tree-structured Haar integral algorithm can be found in Section 3.2, Section 3.3 and Section 3.4

The complexity of the different basis functions to compute the n factor product integral is summarized in Table 1. The table shows that the Haar basis, with the tree-structured Haar integral algorithm, is the most efficient basis from those taken into account for this comparison.

In the remainder of this section we first discuss some mathematical properties of the n factor product in the Haar basis before detailing how the computations can be implemented efficiently.

3.2 Haar Basis Function Products

Let $\varphi^{(l,i,j)}$ be a Haar wavelet basis function

$$\varphi^{(l,i,j)} \in \{\phi^{(l,i,j)}, \psi_{01}^{(l,i,j)}, \psi_{10}^{(l,i,j)}, \psi_{11}^{(l,i,j)}\}.$$

Then the product of two Haar wavelet functions is

$$2^{\alpha}\varphi_r^{(l,a,b)} = \varphi_1^{(k,i,j)}(u,v) \otimes \varphi_2^{(l,a,b)}(u,v) = \prod_{(u,v)\in Im_{k+1}} \varphi_1^{(k,i,j)}(u,v)\varphi_2^{(l,a,b)}(u,v), \quad (26)$$

where

$$Im = (1,1), ..., (2^{k+1}, 2^{k+1}), k > l$$

is the set of all squares at level $k+1$. For example in Figure 4, $k = 2$ and the finest scale has 4x4 wavelet squares. The basis functions at level k are used to analyse a signal of 8x8 pixels at level $k+1$.

The following theorems show that the product of two Haar basis functions is again a Haar basis function, with magnitude 2^{α}. The proofs of the theorems are straightforward using the filter tab matrices of the Haar basis functions and Equation 26. For the limited number of cases the correctness can be shown by iterating over the set of all possible combinations. For each theorem we provide one part of the proof as example. If the wavelets are at different scales, then the wavelet at the coarser scale is projected onto the finer scale and the multiplication is performed there. See Theorem 4 for an example.
It should be noted that the multiplication of Haar basis functions is commutative.

Theorem 1 *The product of two non-overlapping Haar basis functions $\varphi_1^{(k,m,n)}$ and $\varphi_2^{(l,i,j)}$ at arbitrary scales is* 0.

Example Theorem 1: Multiplication of two Haar basis functions at the same scale with different translation.

$$2^l \begin{array}{|c|c|c|c|}\hline 1 & -1 & 0 & 0 \\ \hline 1 & -1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array} \otimes 2^l \begin{array}{|c|c|c|c|}\hline 0 & 0 & 1 & 1 \\ \hline 0 & 0 & -1 & -1 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array} = 4^l \begin{array}{|c|c|c|c|}\hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array} = 0$$

Theorem 2 *Product of two Haar basis functions $\varphi_1^{(l,i,j)}$ and $\varphi_2^{(l,i,j)}$ at level l with identical support.*

1. *If φ_1^l and φ_2^l are wavelets basis functions of the identical type then φ_r^l is the scaling function at (l,i,j), with magnitude 4^l.*
2. *If φ_1^l and φ_2^l are wavelets basis functions of different types then φ_r^l is the third wavelet basis function at level l, with magnitude 2^l.*

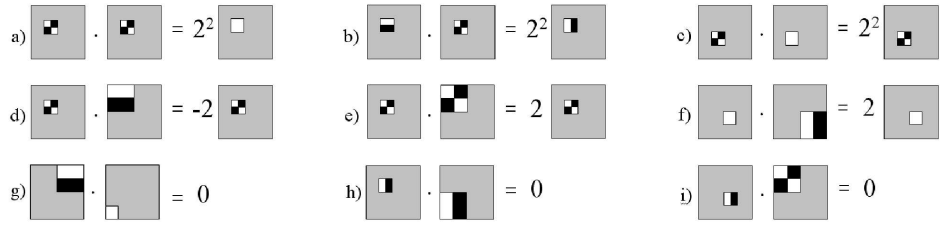

Figure 4: Different examples for the multiplication of two Haar basis functions (Courtesy [Sun 2006])

3. *If φ_1^l and φ_2^l are both the scaling function then φ_r^l is also the scaling function at level l, with magnitude 2^l*

4. *If φ_1^l is a wavelet basis function and φ_2^l is the scaling function then φ_r^l is again the wavelet basis function φ_1^l, with magnitude 2^l.*

Example Theorem 2: Multiplication of two identical wavelets basis functions at level $l = 0$.

$$2^l \begin{array}{|c|c|}\hline 1 & -1 \\ \hline -1 & 1 \\ \hline \end{array} \otimes 2^l \begin{array}{|c|c|}\hline 1 & -1 \\ \hline -1 & 1 \\ \hline \end{array} = 4^l \begin{array}{|c|c|}\hline 1 & 1 \\ \hline 1 & 1 \\ \hline \end{array},$$

Theorem 3 *Product of two overlapping Haar basis functions $\varphi_1^{(k,m,n)}$ and $\psi_2^{(l,i,j)}$ at different scales.*

The product of two Haar basis functions $\varphi_1^{(k,m,n)}$ and $\varphi_2^{(k,m,n)}$, with $k > l$, is the wavelet basis function defined at the finer scale, φ_1, scaled by $\pm 2^k$. The sign is determined by the quadrant of φ_2 in which φ_1 lies. For Haar basis functions φ_1 overlaps φ_2 always in exactly one quadrant.

Example Theorem 3: Product of two wavelets basis functions. The first factor is defined at level 1 and the second at level 2. Note that the first factor is projected onto level before performing the multiplication.

$$2^l \begin{array}{|c|c|c|c|}\hline 1 & -1 & 0 & 0 \\ \hline 1 & -1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array} \otimes 2^k \begin{array}{|c|c|c|c|}\hline 1 & 1 & -1 & -1 \\ \hline 1 & 1 & -1 & -1 \\ \hline 1 & 1 & -1 & -1 \\ \hline 1 & 1 & -1 & -1 \\ \hline \end{array} = 2^k \left(2^l \begin{array}{|c|c|c|c|}\hline 1 & -1 & 0 & 0 \\ \hline 1 & -1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array} \right).$$

Figure 4 shows further examples of the product of two wavelets.

3.3 Integral of Haar basis function

The integral of a Haar basis function is

$$I = \int \int \varphi^{(l,i,j)}\, di\, dj = \frac{1}{4^l} \sum_j \sum_i \Psi^{(l,i,j)}(i,j), \tag{27}$$

where $\Psi^{(l,i,j)}$ is the filter tab matrix of $\varphi^{(l,i,j)}$. The result of the integration is given in Theorem 4.

Theorem 4 *Integral of Haar basis functions.*

1. *The integral of a wavelet basis function* $I = \int\int \psi(l,i,j)didj = 0$.
2. *The integral of a scaling basis function is* $I = \int\int \phi(l,i,j)didj = 2^{-l}$

Theorem 4 can be shown using the filter tab matricies of the Haar basis functions and Equation 27. For example the integral of $\phi^{(1,0,1)}$

$$I = \int\int 2^l \begin{array}{|c|c|c|c|}\hline 1&1&0&0\\\hline 1&1&0&0\\\hline 0&0&0&0\\\hline 0&0&0&0\\\hline\end{array} didj \tag{28}$$

$$= 2^l \frac{1}{4^l} \sum_j \sum_i \begin{array}{|c|c|c|c|}\hline 1&1&0&0\\\hline 1&1&0&0\\\hline 0&0&0&0\\\hline 0&0&0&0\\\hline\end{array} \tag{29}$$

$$= \left(\frac{1}{2^l}\frac{1}{2^l}\right) \cdot 2^l \cdot 4 = 2^{-l} \cdot 4 \tag{30}$$

3.4 n Factor Product Integral

The theorems in the previous subsection show that the product of two Haar basis functions is again a Haar basis function. The product of n Haar basis functions can then be computed as a sequence of two factor products.
The integral coefficients C_n are defined in Equation 7 as,

$$C_n(b_1, b_2, ..., b_n) = \int \prod_{k=1}^{n} b_k(\omega) d\omega. \tag{31}$$

It can be seen easily that the C_n are n factor product integrals. Theorem 5 summarizes the previous theorems and defines the value of the integral coefficients for the Haar basis.

Theorem 5 *Generalized Haar Integral Coefficient Theorem [Sun 2006]*

The n-th order Haar integral coefficient C_n has a nonzero value, if and only if the number of three kinds of wavelet basis functions $\psi_{01}^{(l,i,j)}$, $\psi_{10}^{(l,i,j)}$ and $\psi_{11}^{(l,i,j)}$ at the finest scale have the same parity. In this case, the integral coefficient is $\pm 2^{\sum_l - 2l_o}$ where $\sum_l$ is the sum of the scales of all operand basis functions and l_0 is the scale of the finest basis function. The sign of the integral coefficient is the result of the multiplications of the sign of the squares of all but the wavelets at the finest scale where the wavelets at the finest scale fall into.

The full proof of Theorem 5 can be found in [Sun 2006]. Here, only some important ideas of the proof are presented. It should be emphasized that all wavelets at the finest scale l_0 have to be defined in the same square (l_0, i, j) for the product to be nonzero (Theorem 1). Initially, any scalar scaling factor will be ignored.
The parity of the multiplicity of the wavelet basis function at the finest scale has to be identical. Only in this case is the product of all factors a scaling function and the integral is nonzero (Theorem 4). If the multiplicities of all wavelet basis functions are even, e.g. there are two wavelets of each type, then the product of each pair of wavelet basis functions

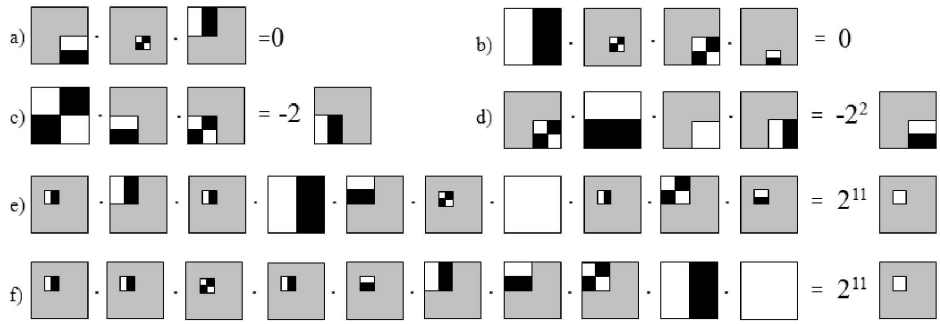

Figure 5: Visual multiplication of multiple Haar basis functions over different scales (Courtesy [Sun 2006])

of the same type is a scaling function (Theorem 2.1). Repeatedly applying Theorem 2.1 shows that the result is again the scaling function (Theorem 2.3). If the multiplicities of all wavelet basis functions at the finest scale are odd, e.g. there are three wavelets of each type, then the product of each pair of wavelet basis functions of the same type is the scaling function (Theorem 2.1). Exactly one wavelet basis function of each wavelet type remains at the finest scale. The product of two of these wavelets basis function is the third wavelet basis function (Theorem 2.2) which, multiplied with the remaining wavelet basis function, forms a scaling function. Any scaling function at the finest scale does not change these results (Theorem 2.3).
The normalization factors can be treated separately by factoring them out. The product of all factors yields $2^{\sum_\iota}$. The second part of the total scaling factor, 2^{-2l_0}, consists of the normalization factor of the integration and the normalization factor of the scaling function resulting from n factor product. The sign of the integral coefficient results directly from the repeated application of Theorem 3. Figure 5 depicts the computation of different integral coefficients.

3.5 Haar Basis Tree and Efficient Computation of the Integral Coefficients

The time complexity for computing Theorem 5 with a brute force implementation is $O(M^n n)$, when one assumes that the complexity of computing an integral coefficients is $O(n)$. In this subsection we show how the complexity of the computations can be reduced to $O(mn)$, with $m \ll M$.
Theorem 6 summarizes the properties of the basis function squares (l, i, j) for the Haar basis.

Theorem 6 *Multi-resolution properties of squares for the Haar basis.*

- *Each parent square at level l has 4 child squares at level $l + 1$.*
- *The child squares are fully contained in the support of the parent square.*
- *The union of all child squares is the parent square.*
- *All child squares are disjoint.*

Based on Theorem 6, all squares (l, i, j) and the corresponding Haar basis functions can be represented in a *Haar basis tree* as shown in Figure 6. It follows from Theorem 6 that only squares in the same branch of the tree overlap. It then can be concluded from Theorem 5

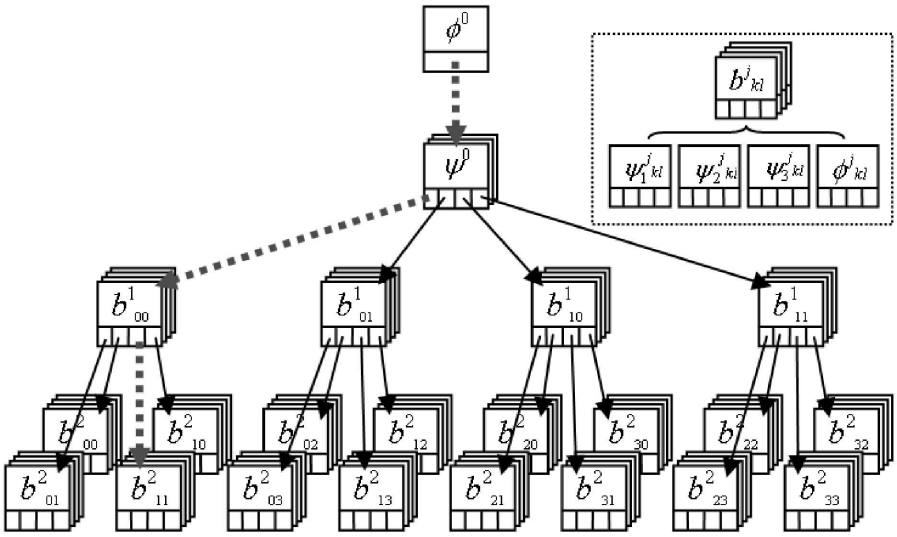

Figure 6: Haar basis tree (Courtesy [Sun 2006])

that nonzero integral coefficients occur only if all factors are Haar basis functions along one branch in the tree. The n factor product integral can therefore be computed efficiently by traversing the Haar basis tree.
In an implementation, the Haar basis tree can be stored in an augmented quadtree which closely resembles the tree shown in Figure 6 [Sun 2006]. One important optimization is that the scalar factor and the sign in Theorem 5 can be computed incrementally while traversing the tree. Then, when computing the n factor product for a basis function defined at (l, i, j), only the signed parent sums `psum` and the Haar basis functions on level l in the same branch of the tree have to be combined. Using the parent sum to compute the n factor product integral yields an algorithm with linear complexity. Nonlinear approximation can be used to further accelerate the computation. The tree then only has to be traversed until all but one coefficients are nonzero. Note that for this algorithm the nonlinear approximation can be precomputed. The final algorithm to compute the n factor product integral has therefore a sublinear complexity of $O(mn)$. The complete algorithm for efficiently computing the n factor product integral is given in [Sun 2006].

3.6 Efficient Computation of Triple Integral Products

The remainder of this section will detail the efficient computation of the triple product integral of lighting, BRDF and visibility. We also explain our implementation of this algorithm. The following discussion is based on [Ng 2004].

The inputs to the algorithm are three signals—lighting, BRDF and visibility. These are parametrized as cubemaps with identical resolution and projected into the Haar basis. The resulting scalar approximation coefficients, the coefficients corresponding to the scaling function, is stored in `scale_*`. The detail coefficients over the different levels for horizontal, vertical and diagonal wavelets, ψ^*_{01}, ψ^*_{10}, and ψ^*_{11} are stored in the detail coefficient vectors `dh_*`, `dv_*` and `dd_*`, respectively. All detail coefficient vectors for a signal have the same length.

For example, for an 16x16 input signal, there are three levels and 8, 4 and 1 wavelet coefficients, respectively. The detail vector for the horizontal decomposition then takes the form

$$dh = [d_h^{(0,0,0)}, d_h^{(1,0,0)}, d_h^{(1,0,1)}, d_h^{(1,1,0)}, d_h^{(1,1,1)}, d_h^{(2,0,0)}, d_h^{(2,0,1)}, ..., d_h^{(2,3,3)}]$$

where coefficient $d_h^{(l,i,j)}$ is the coefficient corresponding to the wavelet basis function $\psi_{01}^{(l,i,j)}$. Each index into the detail coefficient vectors therefore corresponds to one wavelet square.

For computing the triple product integral efficiently Theorem 5 can be further specialized (cf. also Theorem 1 to Theorem 2).

Theorem 7 *Haar Tripling Coefficient Theorem [Ng 2004]*

The tripling coefficient is nonzero, if and only if

1. *all three basis functions are the scaling function, in this case* $C_3 = 1$.
2. *all three basis functions occupy the same square and all are different wavelet types.* $C_3 = 2^l$*, where the square is at level l.*
3. *two basis functions are the identical wavelets basis function, and the third is either the scaling function or a wavelet that overlaps at a strictly coarser level.* $C_3 = \pm 2^l$*, where the third function exists at level l.*

For efficiency, the contributions from each of the three cases in Theorem 7 are computed separately and the results are combined:

```
% initialize with contribution from case 1
integral = scale_l * scale_p * scale_v;

% add contribution from case 2
integral = integral + case2( dh_l, dv_l, dd_l, ...
                             dh_p, dv_p, dd_p, ...
                             dh_v, dv_v, dd_v, ...
                             N );

% add contribution from case 3
integral = integral + case3( scale_l, dh_l, dv_l, dd_l, ...
                             scale_p, dh_p, dv_p, dd_p, ...
                             scale_v, dh_v, dv_v, dd_v, ...
                             N );
```

Given the representation for the approximation coefficients and the detail coefficient vectors, the contribution of case two of Theorem 7 can be computed by iterating over all elements of the detail vectors.

```
function integral = case2( dh_l, dv_l, dd_l, ...
                           dh_p, dv_p, dd_p, ...
                           dh_v, dv_v, dd_v, ...
                           N )

  % level counter
  level = 0;
  % tripling coefficient for the current level
```

```
  tcoeff = 2^level;
  % number of coefficients processed on all previously processed levels
  coeffs_processed = 0;
  % number of coefficients on the current level
  coeffs_level = 1;

  % result
  integral = 0.0;

  % do for all squares at all levels
  for i = 1:length( dh_l)

    integral = integral + tcoeff * ...
                (  dh_l(i) * dv_p(i) * dd_v( i) ...
                 + dh_l(i) * dd_p(i) * dv_v( i) ...
                 + dv_l(i) * dh_p(i) * dd_v( i) ...
                 + dv_l(i) * dd_p(i) * dh_v( i) ...
                 + dd_l(i) * dh_p(i) * dv_v( i) ...
                 + dd_l(i) * dv_p(i) * dh_v( i) );

    % update values then following detail coefficients correspond
    % to the next finer decomposition level
    if(( i - coeffs_processed) == coeffs_level)

      level = level + 1;
      tcoeff = 2^level;
      coeffs_processed = coeffs_processed + coeffs_level;
      coeffs_level = 4^level;
    end
  end

% end function
end
```

Efficiently computing case three of Theorem 7 is more complex because for each square all overlapping squares at coarser levels have to be determined. First it can be noted that, for a fixed $\psi^{(l,i,j)}$, the parent sum is independent of the wavelet type yielding the following implementation

```
function c = case3( scale_l, dh_l, dv_l, dd_l, ...
                    scale_p, dh_p, dv_p, dd_p, ...
                    scale_v, dh_v, dv_v, dd_v, ...
                    N )

  c = 0.0;

  % all translations at all scales
  for i = 1:length( dh_l)

    % psum is independent of the wavelet type
    psum_v = psum( scale_v, details_v, i);
    psum_p = psum( scale_p, details_p, i);
    psum_l = psum( scale_l, details_l, i);

    % for every wavelet type

    % horizontal
    c = c + dh_l(i) * dh_p(i) * psum_v;
```

```
    c = c + dh_v(i) * dh_p(i) * psum_l;
    c = c + dh_l(i) * dh_v(i) * psum_p;

    % vertical
    c = c + dv_l(i) * dv_p(i) * psum_v;
    c = c + dv_l(i) * dv_v(i) * psum_p;
    c = c + dv_v(i) * dv_p(i) * psum_l;

    % diagonal
    c = c + dd_l(i) * dd_p(i) * psum_v;
    c = c + dd_l(i) * dd_v(i) * psum_p;
    c = c + dd_v(i) * dd_p(i) * psum_l;

  end

end
```

The helper function `psum()` computes the sum of all nonzero tripling coefficients resulting from case 3 of Theorem 7. A naive implementation is listed below:

```
function ps = psum( scale, details, i)
% @param scale    1x1  approximation / scaling function coefficient
% @param details  3xN  detail coefficients of all three wavelet types, each
%                      detail vector has N elements
% @param i        1x1  linear index of wavelet square (l,i,j)

  % scaling fucntion has to be always taken into account
  ps = scale;

  % get the index of i in the 2D coefficient matrix at the level of i
  % i.rel is the linear, relative index of square i within all squares at level
  % i.level
  [x_i y_i] = getIndex2DRel( i.rel, i.level);

  % do for all levels which are strictly coarser
  for l = 0 : (i.level - 1)

    % do for all wavelets translations in the current level
    for k = 1 : cnum_coeffs

      % get 2D index of k in the 2D coefficient matrix of l
      [x_c, y_c] = getIndex2DRel( k, l);

      % only squares which overlap i are relevant
      if( doesOverlap( x_i, y_i, x_c, y_c))

        % effective index in the linearized, multi-level representation
        index = coeffs_processed + k;

        % do for all wavelet types at the current level and translation
        for m = 1:3
          % the quadrant of i w.r.t. l-k
          c_uvw = getSignQuadrant( m, x_i, y_i, x_c, y_c);
          c_uvw = c_uvw * (2^l);

          % add contribution
          ps = ps + c_uvw * details( m, index);
        end % end for all three wavelets
```

```
      end  % there is an overlap
    end % end for each wavelet at the current level

    coeffs_processed = coeffs_processed + cnum_coeffs;

  end % end for all levels
```

As already mentioned, this algorithm can be optimized. First, `psum` can be updated incrementally from the values of the previous level as discussed above; second, also the tripling coefficients can be precomputed for a fixed size of the input signals. A more efficient implementation for computing `psum` is presented in Section 4.

3.7 Coordinate frame of computation

The BRDF, $\rho(\omega, \omega_0)$, in Equation 3 is defined in the local coordinate frame of a surface location. The lighting environment, however, is defined in the global coordinate frame. To compute the product of these functions and the visibility, all factors have to be defined in the same coordinate frame. The visibility can be computed easily in a local or global coordinate frame. However, either the lighting environment or the BRDF have to be rotated for computing the product.
We follow the approach proposed by Ng et al. [Ng 2004] and represent the BRDF in the global coordinate frame. The BRDF is then a 6D function, $\hat{\rho}(\omega, \omega_0, n)$, where n is the surface normal at a surface location. For the special case of a diffuse BRDF $\hat{\rho}$ is independent of the view direction ω_0 so that vertex colors computed for a scene containing only diffuse BRDFs are correct independent of the view direction. We employ this property in our plots in Appendix 8.

4 Implementation

We implemented a full rendering pipeline for computing the triple integral product of lighting, BRDF and visibility. For rendering, Equation 17 is computed for each vertex of the model. The final images are obtained with a standard rendering pipeline where the vertex colors are interpolated across the surfaces of the model.
The physically-motivated ray tracer pbrt [Pharr 2004] has been extended and used for the visibility precomputation. Most of the remaining functionality to compute the triple product integral is implemented in Matlab [Mathworks]. In the first phase of the project we also used the Wavelet Toolbox for Matlab which allows to compute the wavelet decomposition and nonlinear approximations for many of the standard wavelets such as Haar, Daubechies and Symlet, in 1D and 2D. Although this functionality could not be used for the final implementation[2], it was valuable to understand some of the concepts of wavelets and to verify our own implementation.

4.1 Lighting

High Dynamic Range images and and analytic light sources are used as lighting environments. We use HDR images from [Debevec 1997] and generate analytic light sources using an image editing program. All lighting environments are parametrized as cross maps. The original images are converted to the `pfm` format. We used PFSTools [PFStools] for this and implemented a `pfm` file reader in Matlab.
Five lighting environments have been used for the experiments discussed in Section 6. The first one is an HDR image of St. Peters Basilica in Rome. The remaining four lighting environments are analytic light source with different solid angles varying from an almost

[2]Currently, only one license for the wavelet toolbox is available and these only on the cdf system.

point-like light source to a big area light source. For computing the triple product integral with input signal resolutions of 64x64, 128x128 and 512x512, the sides of the cross maps have to be resampled. This can locally and globally shift the mean of the image color. At the moment we correct only the global shift per face. To avoid the costly resampling, mean correction and projection into the Haar basis for each new rendering, these computations are performed only once and the result is cached.

4.2 BRDF

The BRDF is defined in a global coordinate frame. Ng et al. [Ng 2004] compute the BRDF by interpolating a precomputed material field. In contrast, we compute the BRDF for each vertex. This is more accurate but also more expensive. The computation of the BRDF parametrized as cubemap is performed directly in Matlab. Currently only monochrome, diffuse BRDFs are supported because we were mainly interested in the frequency characteristic of the rendered effects and used shadows to analyse these. It would be straightforward to extend the current implementation to also handle colored and glossy BRDFs.

4.3 Visibility

Two techniques to sample the visibility for a surface location have been implemented as `pbrt` plugins. First, we have written a visibility ray caster which samples the hemisphere above a surface point. This extension naturally integrates into `pbrt`. Second, we added a graphics hardware accelerated renderer based on OpenGL to `pbrt`. This extension computes visibility cubemaps either in a local or a global coordinate frame. Because `pbrt` is by design a ray tracing system, adding this functionality was not trivial. A more detailed discussion of the extensions and its implementations can be found in a design document which is provide along with this report. The precomputed visibility maps are RLE encoded and stored in files. The compression is efficient because the maps contain only binary information and the visibility is for most scenes a piecewise constant function. For computing the triple product integral, the maps are read into Matlab, decoded and projected into the Haar basis. At the moment, we precompute the visibility maps for different resolutions separately. Precomputing only the highest resolution and downsampling these maps to obtain lower resolution versions should be possible with no or negligible distortion.

4.4 Triple Integral Product

In the following, we will detail an efficient implementation to compute the value of the parent sum `psum` (cf. Section 3). The implementation discussed here differs from those in [Ng 2004], mainly because we used Matlab instead of C++ [3].

The optimized algorithm to compute `psum` is based on the tree-structured Haar integral algorithm described in Section 3 and consists of three phases. In the first phase, a precomputation step, the aggregated value of the non-zero tripling coefficients for each square (l, i, j) from case three of Theorem 7 is determined. As mentioned earlier, psum can be accumulated while traversing the Haar basis tree and therefore only the contribution from the direct parent level has to be precomputed. This computation depends solely on the the size of the input signal and therefore the number of wavelet coefficients. Hence, we precompute the coefficients once for the different resolutions of the input signals which are used. Given the tripling coefficients at the direct parent level and the detail coefficient vectors `details_*`,

[3] To achieve optimal performance in Matlab the implementation has to be vectorized. Vectorization in the context of Matlab means that all operations should be performed as vector or matrix operations; constructs such as loops and branching are in contrast very expensive.

```
details_* = [dh_*, dv_*, dd_*];
```

of the three signals, the value of `psum` can be computed for each square and each input signal. This computation involves only one loop over all decomposition levels and is therefore efficient in Matlab.
The `index_map`, computed in the code listing below, contains the indicies in `details_*` of the wavelet coefficients at the direct parent level.

```
% linear list of indicies at the last scale
% two rows so that the each index is repeated once in the final list
index_map = repmat( start_coeffs_ll : end_coeffs_ll, 2, 1);

% generate the rows in the index list matrix, thereby the indicies in the
% the first column correspond to the indicies necessary to access the
% parent sums of the squares in the first row of the filter tab matrix
% representation
index_map = reshape( index_map, elems_side, size(index_map, 2) / helems_side);

% duplicate each column of index_map
% this generates a square matrix because the number of wavelts in x
% and y direction is identical for tensor product wavelets
ii_map = ceil( 0.5 : 0.5 : helems_side);
index_map = index_map( :, ii_map);
```

The parent sums for one level can therefore be computed as

```
% linearize indicies so that it can be used to index wavelet detail
% coefficient vectors
index_map = index_map(:);

% triple coefficients relevant for the current level
ccoeffs = start_coeffs_current : end_coeffs_current;
tcoeffs = tcoeffs_all( ccoeffs, : );

% compute parent sum
cproduct = dot( tcoeffs', details_l(index_map, :)');
psum_parent_l(ccoeffs) = psum_parent_l( index_map) + cproduct;

cproduct = dot( tcoeffs', details_p(index_map, :)');
psum_parent_p(ccoeffs) = psum_parent_p( index_map) + cproduct;

cproduct = dot( tcoeffs', details_v(index_map, :)');
psum_parent_v(ccoeffs) = psum_parent_v( index_map) + cproduct;
```

Performing the computations outlined above for each decomposition level, starting at the coarsest scale, efficiently computes the value of `psum` for all squares (l, i, j). The contributions of case three of Theorem 7 can then be computed as sum of dot products:

```
% horizontal
c = c + details_l(:,1) .* details_p(:,1) .* psum_parent_v;
c = c + details_v(:,1) .* details_p(:,1) .* psum_parent_l;
c = c + details_l(:,1) .* details_v(:,1) .* psum_parent_p;

% vertical
c = c + details_l(:,2) .* details_p(:,2) .* psum_parent_v;
c = c + details_v(:,2) .* details_p(:,2) .* psum_parent_l;
c = c + details_l(:,2) .* details_v(:,2) .* psum_parent_p;
```

```
% diagonal
c = c + details_l(:,3) .* details_p(:,3) .* psum_parent_v;
c = c + details_v(:,3) .* details_p(:,3) .* psum_parent_l;
c = c + details_l(:,3) .* details_v(:,3) .* psum_parent_p;

c = sum( c);
```

To perform the nonlinear approximation, we precompute an index list `index_approx_*` of the $k\%$ largest coefficients for each signal. In contrast to [Ng 2004] where all coefficients which are not part of the approximation are discarded, we also keep the full coefficient vectors. Therefore, our implementation provides more accuracy when combining the different factors because `index_approx_*` can be used to index into the full coefficient vectors of all three factors. However, this comes at the price of considerably larger memory requirements. Similar to [Ng 2004] we use the index list of the BRDF to compute the contribution of Theorem 7.2. For the contribution from Theorem 7.3 the index list of the BRDF is used for those summands which involve `psum_v` and `psum_l`. The index list of the lighting is used for the summands which involve `psum_p`.

4.5 Runtime Environment

We performed our computations on a cluster with 20 nodes. The computations for each vertex are independent from those of any other vertex and therefore each node can independently process a subset of the vertices. We use a python script to distribute the computation across several nodes.

5 Error Measures

For this project we are investigating the effect of using different degrees of nonlinear approximation and different resolutions of the input signals on PRT rendering with Haar wavelets. Hence, we require a ground truth rendering and an error measure by which to compare the approximate reconstructions with the ground truth. For our ground truth, we use a rendering of the scene using all wavelet coefficients and a high resolution of the input signals, either $256\text{x}256$ or $512\text{x}512$.

The error measures should meet the following goals.

- They should be easy to interpret.
- They should reflect the perceived difference between the reconstructions.
- They should localize the errors. It is important to know where in the scene the errors are occurring.
- They should be applicable early in the rendering pipeline.

There are several points in the pipeline at which the difference between the baseline and an approximation may be measured. In the interest of efficiency it is desirable to make this comparison at an early stage in the pipeline. Ideally, an error measure in the coefficient space of the approximated BRDF, environment, and visibility maps would serve as a measure of how different the respective renderings would appear. We use the average L^2 performance as an error measure of nonlinear approximation in coefficient space. The L^2 performance is the percentage of squared signal energy retained after compression. If X is the original signal, and X_a is the approximated signal, the L^2 error is given by [Mathworks]

$$\frac{100\|X_a\|^2}{\|X\|^2}.$$

In our case, X are the original wavelet coefficients obtained by transforming a *monochrome* version of the input signal and X_a are the coefficients after nonlinear approximation. L^2 performance values near 100% imply that reconstruction will be faithful. The L^2 performance measures the fidelity of the approximate maps and we have used this measure to test the hypothesis that accurate maps correspond to accurate renderings.

We considered using a perceptual metric to compare renderings in image space. This approach is obviously ideal for meeting the second goal, but computing perceptual-based error measures tends to be expensive [Ramasubramanian 1999]. This presents a difficulty if the error must be frequently recalculated, something that is necessary, for example, to rotate a 3D visualization of the error. An alternative is to use a few levels of a Laplacian or steerable pyramid to obtain a crude perceptual error measure. The rationale is that in the case of PRT, changes in sharpness or blurriness of shadows are paramount, and comparing the response of the images to derivative filters at multiple scales is a way to measure these changes. We implemented this simple approach, but early experiments showed that the differences at the various scales are very scene dependent and difficult to interpret, especially at coarser scales. It is not clear that they could be combined to form a meaningful perceptual error measure.

Due to the inadequacies of the perceptually-based metric, we chose to simply measure distance between vertex colors. These distances are simple to interpret and they may be visualized directly as colors in the scene, meeting the first and third goals. The mean, the standard deviation, the variance and the maximal color error provide additional information about the errors. The mean error is not a perceptual metric; renderings may have a relatively high mean error without being noticeable. Nevertheless, this value is important as algorithms must attempt to minimize it. The standard deviation measures, roughly, the noisiness of the error. If the error is noisy, then there will be portions of the rendering that are very inaccurate. It is desirable to have a very low standard deviation. It is also clearly desirable to minimize the maximum vertex error. The standard deviation and the maximal color error tend to provide a measure for the perceived error.

6 Results

In this section we discuss the results obtained with our triple product integral renderer, in particular, the influence of the input signal size and nonlinear approximations on the quality of the renderings. We also examine the correlation between the L^2 error of the wavelet coefficients after the nonlinear approximation and the vertex color error. For convenience, the resolution of the sides of the cubemaps is denoted as input signal resolution. For the nonlinear approximation ratio we use the symbol r. With a ratio of $r = 1$ all Haar basis coefficients are used, with $r = 0.01$ only 1%.
The graphs discussed in the following are shown in Appendix 8.

6.1 Image quality

The error measures, i.e. mean, standard deviation, variance and maximal color error, are computed as discussed in Section 5 and visualized in bar graphs. The computed color values are rescaled for the renderings so that the maximal color value is 0.7. The normalizaton is necessary because some light sources, in particular analytic ones with a small solid angle, do not have the energy to generate renderings which are bright enough to be meaningful. We also show rendering with the normalized vertex color error as vertex color. These plots provide a very intuitive way to understand the localization of the error. They are also useful to evaluate the appropriateness of the vertex color error measures.

6.1.1 Cylinder and Bump

The 'Cylinder and Bump Scene' contains a tall cylinder and a Gaussian-like bump. Both objects are arranged on a ground plane. The cylinder casts sharp shadows onto the ground plane. Soft shadows can be found on the sides of the bump.
Table 3 and Table 4 show the rendered scene with the computed vertex colors and vertex color errors, respectively. Table 5 to Table 8 and Table 9 to Table 12 contain plots of the vertex color error measures for fixed resolution of the input signal and fixed nonlinear approximation ratio, respectively.
For a fixed resolution of the input signals the error without nonlinear approximation, i.e. $r = 1$, is almost negligible. Only some small artifacts are introduced by low resolution input signals. For $64\text{x}64$, the error increases with decreasing nonlinear approximation ratio. For $128\text{x}128$ the differences in the error measures for approximation ratios of 0.1, 0.05 and 0.01 are significantly less pronounced than for $64\text{x}64$. For $256\text{x}256$ and $512\text{x}512$ such difference do not exist. The mean of the vertex color error is for $256\text{x}256$ and $512\text{x}512$ several orders of magnitude lower than for input signals with a lower resolution.
For $r = 1$ all error measures are decreasing with increasing input signal size. The only exception is $128\text{x}128$ which has a slightly increased error in the mean value. We believe that this results from the particular characteristics of the lighting environment. For a fixed approximation ration of $r = 0.05$ and an input resolution of $64\text{x}64$, the error measures, except the mean, are unexpected low compared to those of higher resolution input signals. We think that this is a result of the locally non energy preserving down sampling of the lighting environment. . This can cause that some lights are either that some lights are either dimmed or accentuated. This also explains the increased error in the mean for $r = 0.05$ and $64\text{x}64$.

6.1.2 Cubes Scenes

The 'Cubes Scene' consists of an array of cubes which is positioned in front of a wall. It is inspired by [Durand 2005]. We modeled different versions of the scene with two different cube sizes and three different distances of the cubes from the wall. The results for all variations are very similar. We therefore provide only those for large cubes which are far apart from the wall.
For this scene we performed the computations only for input signal resolution of $64\text{x}64$ and $128\text{x}128$. Renderings with the computed color values and the vertex color error are shown in Table 13 and Table 14, respectively. Because of the similarity with the results from 6.1.1 we omit error measure graphs for this scene.

6.1.3 Nursery Scene

The 'Nursery Scene' is a more realistic test case. It consists of a child seat and two corresponding tables. Renderings with the computed color values and the vertex color error are shown in Table 15 and Table 16, respectively. It has to be noted that the jaggy edges of the shadows on the ground plane result from a insufficient tessellation and not the rendering algorithm. The vertex color error measures are again very similar to those in Section 6.1.1 and Section 6.1.2. We therefore omit these here.

6.1.4 Cubes with Analytic Light Sources

Captured, realistic lighting environments are suitable for estimating the performance of the algorithm for real scenes. However, these scenes usually contain multiple light sources of different intensities. This makes it difficult to judge the correctness of the results. We therefore performed experiments with analytic light sources of varying solid angle. The generated renderings provide more insight into the frequency characteristics of the triple product integral rendering algorithm. We used a point light like light source to analyse the

resolution	no nonlinear approximation	nonlinear approximation
64	5.8 min	7.92 min
128	19.87 min	28.5 min
256	3.2 h	-
512	7.1 days	-

Table 2: Render times for 'Cubes Scene' with 85860 vertices (Intel Xeon CPU with 3.6 GHz, 4 GB RAM)

performance for high-frequency lighting effects and bigger light sources for low frequency effects. The different lights are shown in Table 19. We used the same scene as under 6.1.2 for these experiments.
Renderings with the computed color values and the vertex color errors are shown in Table 17 and Table 18, respectively. The plots show that the triple product integral rendering algorithm can capture low and high frequency effects well. For low frequency effects some ringing and blocking artifacts occur. We believe that these are introduced by the Haar basis. The vertex color error measures are very similar to those for the other scenes and therefore omitted.

6.2 L^2 Error in the Basis Coefficients

The L^2 error norm of the Haar basis coefficients for the 'Cylinder and Bump Scene' is shown in Table 20 and Table 21. The largest error occurs for the lighting environment, the HDR image of St. Peters Basilica. This results from the more regular structure of the BRDF and the visibility which make these signals more amenable for wavelet compression. When using an analytic light source such as those employed for the experiments in section 6.1.4, the L^2 error for the lighting is significantly lower. Analogously, for scenes with a complex visibility the L^2 error for the basis function representation is higher and increases with decreasing r; in the general behavior similar the L^2 error of the coefficients of the HDR image. A scene with such a more complex visibility is the 'Cubes Scene'. Table 20 shows results for a fixed resolution of the input signal. In this case the error increases quite rapidly with decreasing nonlinear approximation ratio. Analogously, for a fixed approximation ratio such as shown in Table 20, the error decreases quickly with increasing resolution of the input signal. Although the results are not fully consistent, the error roughly reduces by a factor of 4 when using the next higher resolution of the input signal. For a fixed resolution of the input signal, the error roughly increases by a factor of 5 when using 1% of the coefficients instead of 5%. This shows that the L^2 error in the coefficients is linear in the approximation ratio as well as in the resolution of the input signals.

6.3 Error estimation

One interesting question is if the error in the wavelet coefficients can be used to estimate the vertex color error. As shown in section 5, the L^2 error is increasing for decreasing input signal size or decreasing nonlinear approximation ratio. However, the results discussed in section 6.1 show that in both cases the image quality does not necessarily decrease. For example Table 5 to Table 7 show that the error remains constant for approximation ratios of 0.1, 0.05 and 0.01. This is in conflict with the second graph in Table 20. The renderings in Table 3 and Table 4 also show that the vertex color error does not increase in proportion to the L^2 error in the wavelet coefficients.

6.3.1 Discussion

The results obtained in our experiments show that both low and high frequency effects can be captured well by PRT using the Haar basis. In contrast to previous work we also performed experiments with high resolution input signals up to $512\text{x}512$. Our results show that a much more aggressive nonlinear approximation is possible in this case. Based on the results obtained for $64\text{x}64$ it is likely that also for higher input resolutions a similar increase in the error will be observed—but for far lower nonlinear approximation ratios.
The effects in the renderings show a variety of different effects in term of their frequency characteristics. This indicates that our mixture of different scenes and captured and analytic lighting environments is a good test bed which is suitable to spot characteristics of particular algorithms. We also believe that our error measures are suitable for future use because the information they provide lines up well with the renderings using the computed color values and the vertex color error. The results observed for the different scenes are very similar. This shows that the observed behavior is, to a large extent, scene independent and can be generalized.
Our results do not include a discussion of the runtime performance. First because we were mainly interested in the frequency characteristics of PRT using wavelets and designed the framework aiming at flexibility rather than optimized computation time. Second, we used Matlab to implement the renderings. This development environment has quite specific performance characteristics and any timings obtained are difficult to generalize to iterative programming languages such as C++ which is commonly used for real-time rendering [4]. The order of magnitude of the computation time can be found in Table 2.

7 Conclusions and Future Work

In this report we provided an introduction into Precomputed Radiance Transfer. We showed that the problem can be formulated as an n factor products integral and that it is efficient to compute this integral in a suitable basis. The derivations are detailed for the computation of the triple product integral using the Haar basis. A full rendering pipeline for this algorithm is discussed has been implemented. Results obtained with our system are presented and discussed. In particular the frequency characteristics of the algorithm and the influence of the resolution of the input signals, parametrized as cubemaps, and the nonlinear approximation on the quality of renderings are analysed. The applicability of an error measure in wavelet coefficient space to estimate the vertex color error has been examined.
We show show that the implemented algorithm can capture low and high frequency effects accurately even with rather low resolution input signals and a low number of wavelet coefficients. For input signal resolutions of $128\text{x}128$ and above, the vertex color error is mostly unaffected by the nonlinear approximation. This shows that a more aggressive nonlinear approximation than used in our experiments is reasonable.
One avenue of future work is therefore to experiment with smaller approximation ratios. Thereby, it would be interesting to use a fixed number of Haar basis coefficients rather than a fixed percentage value. This approach has also the advantage that the runtime performance remains constant. For a fixed percentage value, the computation time increases roughly by a factor of 4 when using a signal with 4 times as many values, e.g when using $64\text{x}64$ instead of $128\text{x}128$. The similarity of the results for the different scenes and lighting environments suggest that the results obtained are generalizable and our test bed is suitable for future use. However, adding some additional scenes to validate the results would be worthwhile. Recently, Wang et al. [Wang 2006] proposed an algorithm for efficiently rotating a signal in a wavelet basis. It would be interesting to incorporate this to avoid the

[4] For example, the computation of a dot product of a subset of indicies of two vectors is more expensive than the product of all elements of the vectors—even when the number of multiplications is only 1%.

recomputation of the BRDF for every vertex.

Some more general areas of future work can be identified as well. The nonlinear approximation of any of the factors in the triple product integral can be used to provide the sparsity pattern `index_approx_*` which is used in the computations. At the moment we use the patterns as proposed by Ng et al. [Ng 2004], e.g. the index list of the BRDF for case 2 of Theorem 7. However, different alternatives exist and it is not obvious which one provides the best runtime performance and quality. In particular, we believe that the best choice depends on the input signals. One indicator for this are the very different L^2 errors for the wavelet coefficients for captured and analytic lighting environments. In the future, we want to find a way to identify the most suitable sparsity pattern efficiently based on a input signals or its representation in the basis function.

At the moment the surface color is determined at each vertex of an input model and then linearly interpolated to obtain an image. This requires a high tessellation of the scene geometry even in flat areas. If the tessellation is not high enough artifacts such as the jaggy edges in Table 15 appear. A more sophisticated interpolation technique in basis function space, in spirit similar to [Green 2006], could provide an alternative.

Another interesting avenue for future work is the use of wavelet basis other than Haar. One direction is to employ existing standard wavelets such as Daubechies and Symlets. These wavelets are smoother than Haar wavelets and are likely to provide better compression performance, especially for complex signals. We performed some preliminary experiments by compressing visibility maps with different wavelet basis functions. With the same number of wavelet coefficients after compression, significant differences in the reconstructed images were visible. It is currently not clear how these differences will affect renderings and we want to investigate this in future work. An alternative to using existing wavelets is to construct custom wavelets. Previous work in image compression shows that adaptive wavelets can be used to improve the performance using either genetic algorithms [Takehisa 2000] [Grasemann 2005] or Machine Learning techniques [Ahmed 2005].

One problem when using wavelet bases other than Haar is that the tripling coefficients are not as sparse as for the Haar basis. An important factor for the sparsity of the integral coefficients is the support of the basis functions; it can be shown that the integral coefficients are 0 for the product of any two basis functions which do not overlap. The Haar basis has a support of 2 which is minimal for all wavelets. However, for new basis functions it might be possible to identify additional factors which can accelerate the computation. Using symmetries, e.g. in Symelts or biothorgonal waveletes, could be one approach to reduce the computational complexity; wavelets with a spherical domain might also provide some advantages. Another problem with wavelet bases other than Haar is that it is much more involved to derive the sparsity pattern in the integral coefficients. However, this derivation can be performed with a brute force approach by testing all possible combinations of basis functions. We performed some experiments and were able to show that this is in fact possible; in particular, we computed the nonzero tripling coefficients for the Haar basis.

When testing different wavelets for their applicability for PRT it would be valuable to investigate an alternative approach to estimate the vertex color and image space error based on the wavelet coefficients. This could significantly reduce the computation times to evaluate a new basis.

An examination of different basis functions was one of the original goals of this project. However, designing a flexible and reliable rendering pipeline took more time than expected. Therefore we focused on Haar wavelets.

References

[Ahmed 2005] R. Ahmed, *Wavelet-based Image Compression using Support Vector Machine Learning and Encoding Techniques*, Computer Graphics and Imaging 2005, August 2005

[Daubechies 1994] I. Daubechies, *Ten Lectures on Wavelets*, CBMS, SIAM, 61, 1994, 271-280, 1994

[Donoho 1999] D. Donoho, M. R. Duncan, X. Huo, O. Levi, J. Buckheit, M. Clerc, J. Kalifa, S. Mallat, and Yu Thomas, *Wavelab (a Matlab wavelet library)*, 1999, http://www-stat.stanford.edu/wavelab

[Debevec 1997] P. Debevec, J. Malik, *Recovering High Dynamic Range Radiance Maps from Photographs*, SIGGRAPH 1997, August 1997

[Dechevsky 2003] L. T. Dechevsky, N. Grip, E. Quak, *A comparative study of current Matlab and C++ wavelet software*, Three similar technical reports available from http://www.sm.luth.se/ grip/Research/Publications/Wavelets/SoftwareComparison/

[Durand 2005] F. Durand, N. Holzschuch, C. Soler, E. Chan F. Sillion, *A Frequency Analysis of Light Transport*, SIGGRAPH 2005, August 2005

[Cody 1993] M. A. Cody, *The fast wavelet transform: beyond Fourier transforms*, Dr. Dobb's Journal 1993; April pp:44-54.

[Faust 2005] M. Faust, *c2g: Color Image to Grayscale Conversion*, http://www.e56.de/c2g.php

[Gautron 2004] P. Gautron, J. Kivnek, S. Pattanaik, K.Bouatouch, *A Novel Hemispherical Basis for Accurate and Efficient Rendering*, Eurographics Symposium on Rendering 2004, June 2004

[Gooch 2005] Amy A. Gooch, Sven C. Olsen, Jack Tumblin, Bruce Gooch, *Color2Gray: salience-preserving color removal*, SIGGRAPH 2005, July 2005

[Gouze 2004] A. Gouze, M. Antonini, M. Barlaud, and B. Macq, *Design of Signal-Adapted Multidimensional Lifting Scheme for Lossy Coding*, IEEE Transactions on Image Processing, 13(12), December 2004

[Grasemann 2005] U. Grasemann, R. Miikkulainen, *Real World Applications: Effective Image Compression using Evolved Wavelets*, Proceedings of the 2005 conference on Genetic and evolutionary computation GECCO '05, June 2005

[Green 2006] P. Green, J. Kautz, W. Matusik, F. Durand, *View-Dependent Precomputed Light Transport Using Nonlinear Gaussian Function Approximations*, ACM 2006 Symposium in Interactive 3D Graphics and Games, March 2006

[Green 2003] R. Green, *Spherical Harmonic Lighting: The Gritty Details*, Game Developer Conference 2003, 2003

[James 2003] D. L. James, K. Fatahalian, *Precomputing Interactive Dynamic Deformable Scenes*, SIGGRAPH 2003, July 2003

[Jensen 2000] A. Jensen, A. la Cour-Harbo, *Ripples in Mathematics, The Discrete Wavelet Transform*, Springer-Verlag, 2000

[Jepson 2000] A. Jepson, *utvisToolbox*, University of Toronto, 2000

[Kajiya 1986] J. T. Kajiya, *The Rendering Equation*, SIGGRAPH 1986, August 1986

[Kautz 1999] J. Kautz, M. D. McCool, *Interactive Rendering with Arbitrary BRDFs using Separable Approximations*, Eurographics Workshop on Rendering 1999, June 1999

[Kautz 2004] J. Kautz, J. Lehtinen, T. Aila, *Hemispherical Rasterization for Self-Shadowing of Dynamic Objects*, Eurographics Symposium on Rendering 2004, June 2004

[Kautz 2005] J. Kautz, J. Lehtinen, P.-P. Sloan *Precomputed Radiance Transfer: Theory and Practice*, Course Notes SIGGRAPH 2005, August 2005

[Lehtinen 2003] J. Lehtinen, J. Kautz, *Matrix Radiance Transfer*, ACM 2003 Symposium on Interactive 3D Graphics, April 2003

[Liu 2004] X. Liu, P.-P. Sloan, H.-Y. Shum, John Snyder, *All-Frequency Precomputed Radiance Transfer for Glossy Objects*, Eurographics Symposium on Rendering 2004, June 2004

[Mei 2004] C. Mei, J. Shi, F. Wu, *Rendering with Spherical Radiance Transport Maps*, Eurographics 2004, 2004

[MacRobert 1948] T. MacRobert, *Spherical Harmonics; an Elementary Treatise on Harmonic Functions, with Applications*, Dover Publications, 1948

[Mathworks] Mathworks Inc., www.mathworks.com

[Misti 2003] Michel Misiti, Georges Oppenheim, Jean-Miche Poggi, and Yves Misiti. *The Mathworks wavelet toolbox (for Matlab)*, 2003, http://www.mathworks.com/products/wavelet/index.shtml.

[Ng 2003] R. Ng, R. Ramamoorthi, P. Hanrahan, *All-Frequency Shadows using Non-Linear Wavelet Lighting Approximation*, SIGGRAPH 2003, July 2003

[Ng 2004] R. Ng, R. Ramamoorthi, P. Hanrahan, *Triple Product Wavelet Integrals for All-Frequency Relighting*, SIGGRAPH 2004, August 2004

[Ojanen 1998] H. Ojanen, *WAVEKIT: a Wavelet Toolbox for Matlab*, http://www.math.rutgers.edu/ ojanen/wavekit/index.html

[PFStools] *PFSTools for High Dynamic Range Images and Video*, http://www.mpi-sb.mpg.de/resources/pfstools/

[Pharr 2004] M. Pharr, G. Humphreys, *Physically Based Rendering: From Theory to Implementation*, Morgan Kaufmann, 2004

[Ramasubramanian 1999] M. Ramasubramanian, S. N. Pattanaik, D. P. Greenberg, *A Perceptually Based Physical Error Metric for Realistic Image Synthesis*, SIGGRAPH 1999, July 1999

[Schröder 1995] P. Schröder, W. Sweldens, *Spherical Wavelets*, SIGGRAPH 1995, September 1995

[Simoncelli 2004] E. Simocelli, *matlabPyrTools: Matlab source code for multi-scale image processing*, http://www.cns.nyu.edu/ lcv/software.html

[Sloan 2002] P.-P. Sloan, J. Kautz, J. Snyder, *Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low- Frequency Lighting Environments*, SIGGRAPH 2002, July 2002

[Sloan 2003a] P.-P. Sloan, X. Liu, H.-Y. Shum, J. Snyder, *Bi-Scale Radiance Transfer*, SIGGRAPH 2003, July 2003

[Sloan 2003b] P.-P. Sloan, J. Hall, J. Hart, J. Snyder, *Clustered Principal Components for Precomputed Radiance Transfer*, SIGGRAPH 2003, July 2003

[Sloan 2005] P.-P. Sloan, B. Luna, and J. Snyder, *Local, Deformable Precomputed Radiance Transfer*, SIGGRAPH 2005, July 2005

[Stollnitz 1996] E. Stollnitz, T. D. DeRose, D. H. Salesin *Wavelets for Computer Graphics: Theory and Applications*, Morgan Kaufmann Publishers, 1996

[Sun 2006] W. Sun, A. Mukherjee, *Generalized Wavelet Product Integral for Rendering Dynamic Glossy Objects*, SIGGRAPH 2006, To appear July 2006

[Sweldens 1998] W. Sweldens, *The Lifting Scheme: a Construction of Second Generation of Wavelets*, SIAM J. Math. Anal., 29 (2), pp. 511-546, 1998

[Takehisa 2000] Y. Takehisa, H. Sakanashi, T. Higuchi, *Adaptive Wavelet Transform for Lossless Compression using Genetic Algorithm*, 2000 Genetic and Evolutionary Computation Conference, July 2000

[Tan 2005] P. Tan, S. Lin, L. Quan, B. Guo, and H.-Y. Shum, *Multiresolution Reflectance Filtering*, Eurographics Symposium on Rendering 2005, Juni 2005

[Wang 2006] , R. Wang, R. Ng, D. Luebke, G. Humphreys, *Efficient Wavelet Rotation for Environment Map Rendering*, Eurographics Symposium on Rendering 2006, To appear June 2006

[Wikipedia a] Daubechies Wavelets, http://en.wikipedia.org/wiki/Daubechies_wavelet

[Wikipedia b] Peak Signal To Noise Ration (PSNR), http://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio

[Zhou 2005] K. Zhou, Y. Hu, S. Lin, B. Guo, H.-Y. Shum, *Precomputed Shadow Fields for Dynamic Scenes*, SIGGRAPH 2005, August 2005

8 Graphs

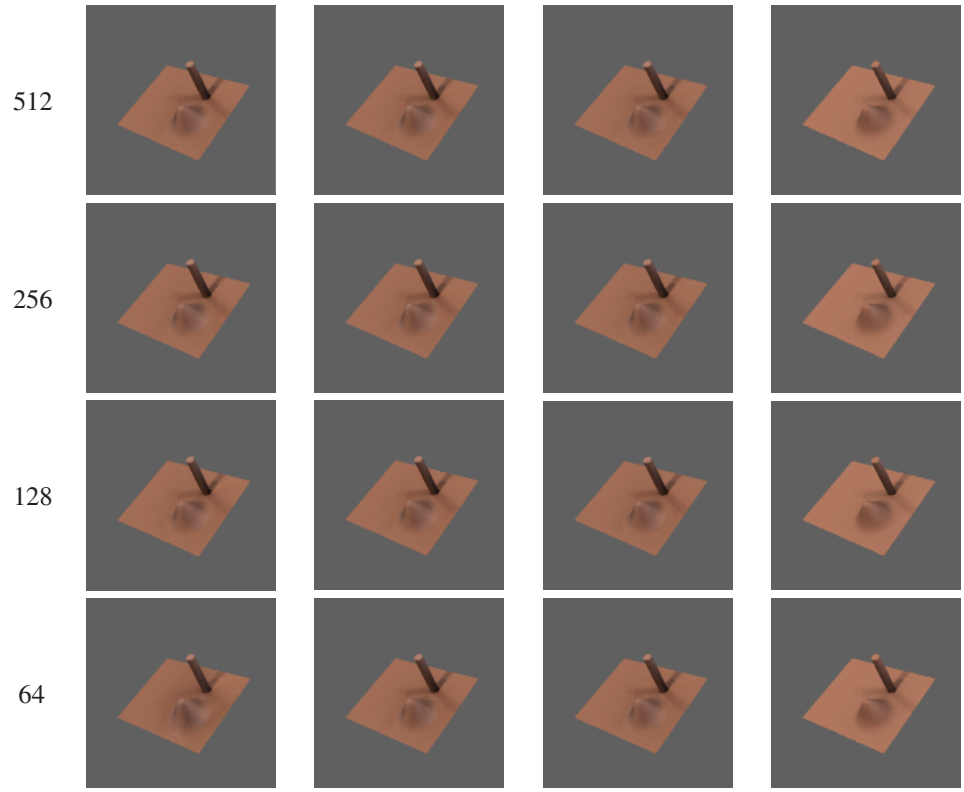

Table 3: Renderings for 'Cylinder and Bump Scene' for different input signal resolutions (y axis) and different nonlinear approximation ratios (x axis).

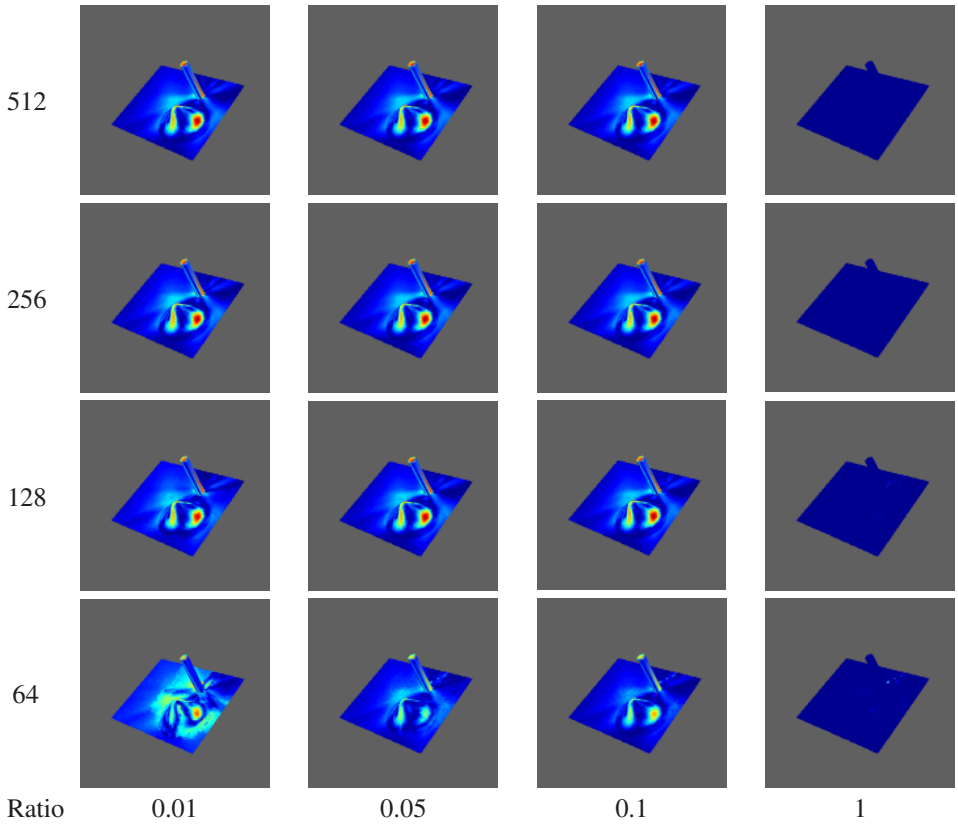

Table 4: Vertex color error for 'Cylinder and Bump' scene for different input signal resolutions (y axis) and different nonlinear approximation ratios (x axis). Dark blue corresponds to very low error, red areas to very high error. Note that error is only normalized over rows of the table.

64x64 128x128 256x256 512x512

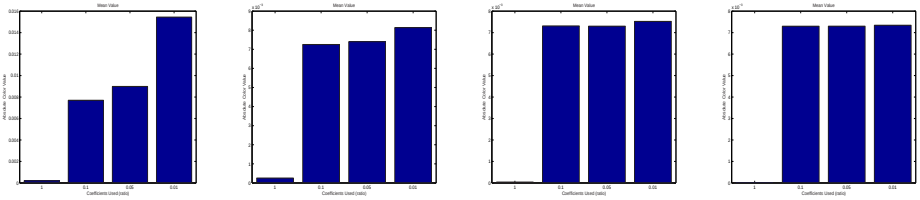

Table 5: Mean of the vertex color error for fixed resolutions and varying nonlinear approximation ratios. Note that the scale of the graphs for a resolution of 256x256 and 512x512 is orders of magnitude smaller than those for 64x64 and 128x128.

64x64 128x128 256x256 512x512

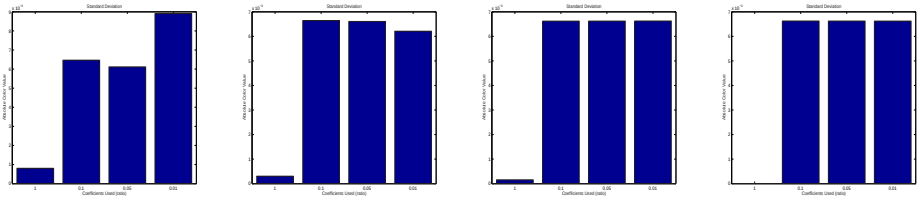

Table 6: Standard Deviation of the vertex color error for fixed resolutions and varying nonlinear approximation ratios.

64x64 128x128 256x256 512x512

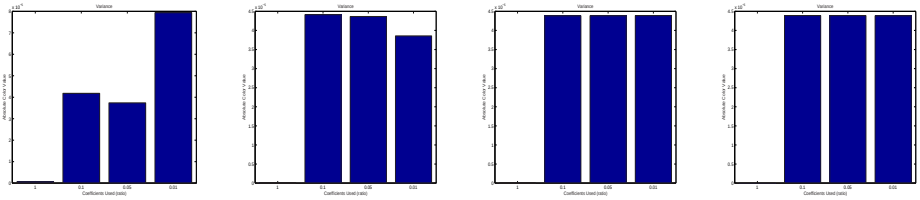

Table 7: Variance of the vertex color error for fixed resolutions and varying nonlinear approximation ratios.

64x64 128x128 256x256 512x512

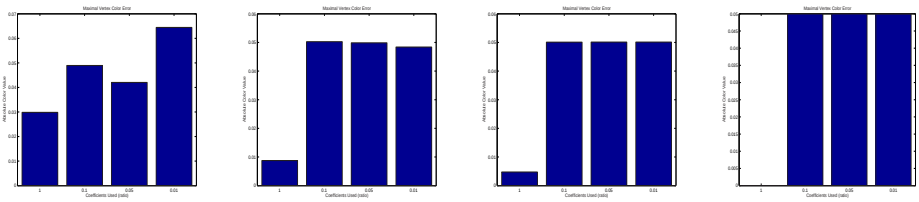

Table 8: Maximum vertex color error for fixed resolutions and varying nonlinear approximation ratios.

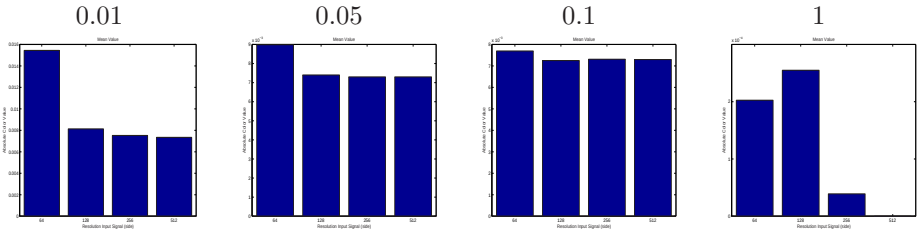

Table 9: Mean of the vertex color error for fixed nonlinear approximation ratios and varying input signal resolutions. Note that the scale of the graphs for approximation ratios of 0.05, 0.1 and 1 is orders of magnitude smaller than those for 0.01.

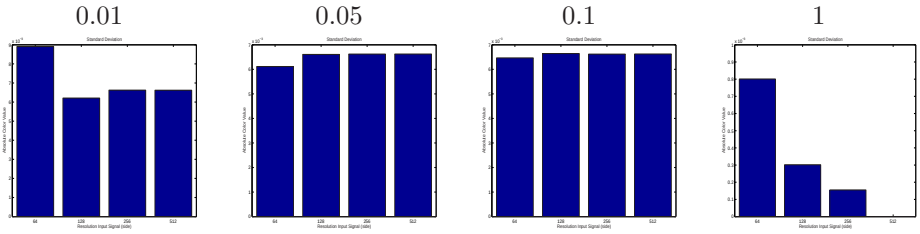

Table 10: Standard deviation of the vertex color error for fixed nonlinear approximation ratios and varying input signal resolutions.

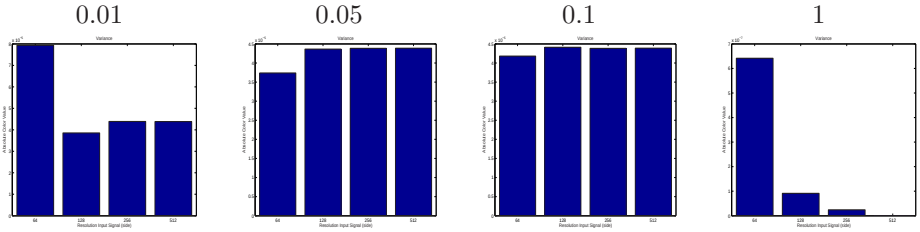

Table 11: Variance of the vertex color error for fixed nonlinear approximation ratios and varying input signal resolutions.

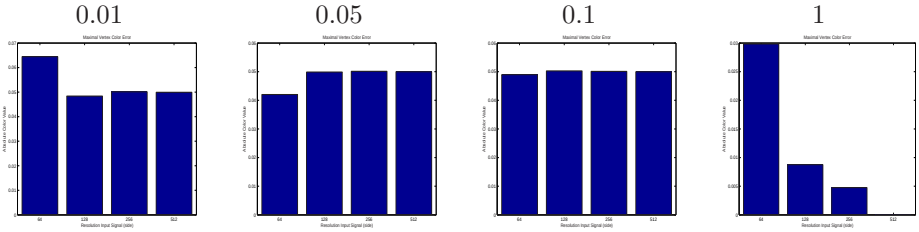

Table 12: Maximum vertex color error for fixed nonlinear approximation ratios and varying input signal resolutions.

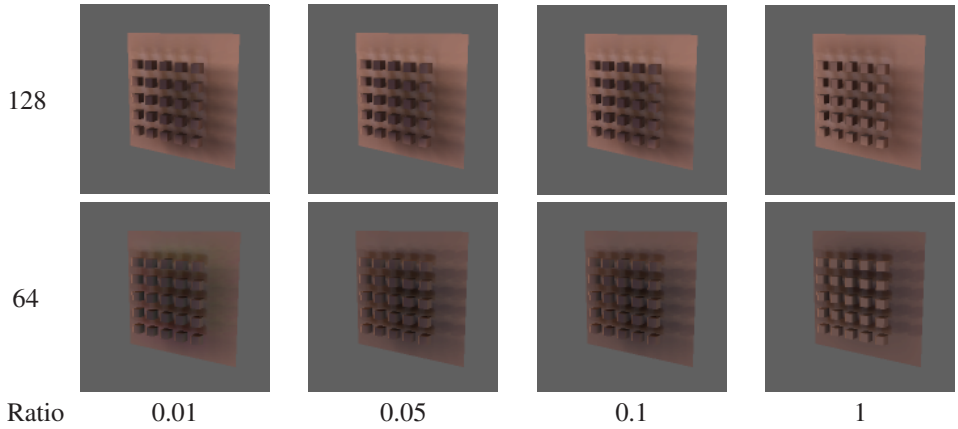

Table 13: Renderings for 'Cubes Scene' with large cubes which are positioned far apart from the wall. Different input signal resolutions (y axis) and different nonlinear approximation ratios (x axis) are shown.

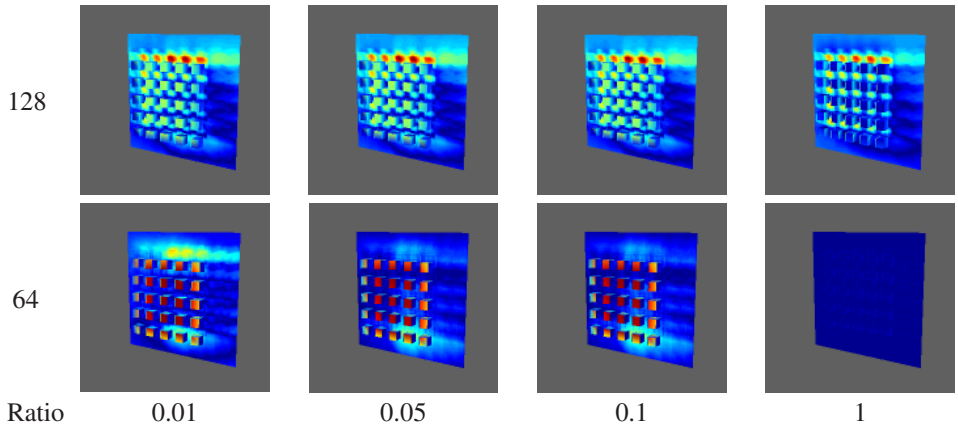

Table 14: Vertex color error for 'Cubes Scene' for different input signal resolutions (y axis) and different nonlinear approximation ratios (x axis). Dark blue corresponds to very low error, red areas to very high error. Note that error is only normalized over rows of the table.

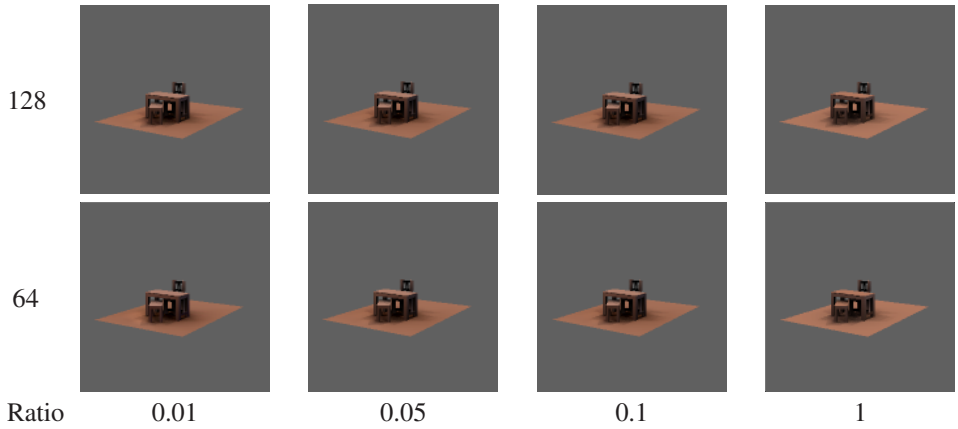

Table 15: Renderings for 'Cubes Scene' with large cubes which are positioned far apart from the wall. Different input signal resolutions (y axis) and different nonlinear approximation ratios (x axis) are shown.

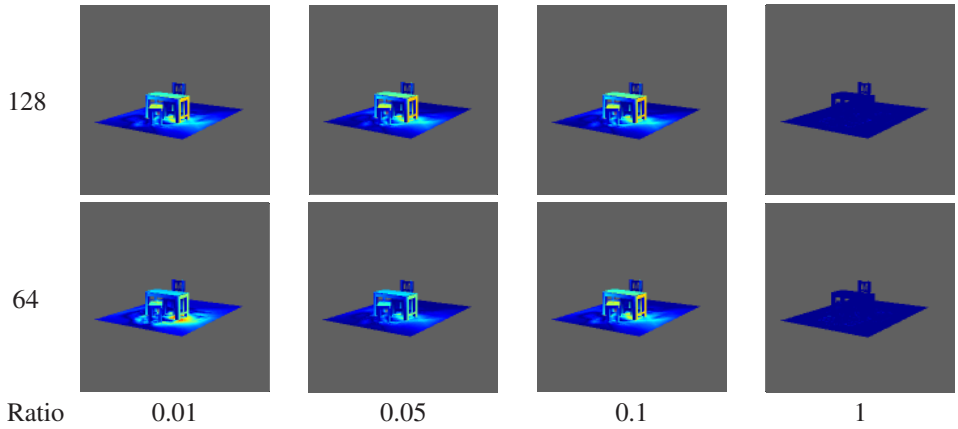

Table 16: Vertex color error for 'Nursery Scene' for different input signal resolutions (y axis) and different nonlinear approximation ratios (x axis). The shown error is normalized. Dark blue corresponds to very low error, red areas to very high error. Note that error is only normalized over rows of the table.

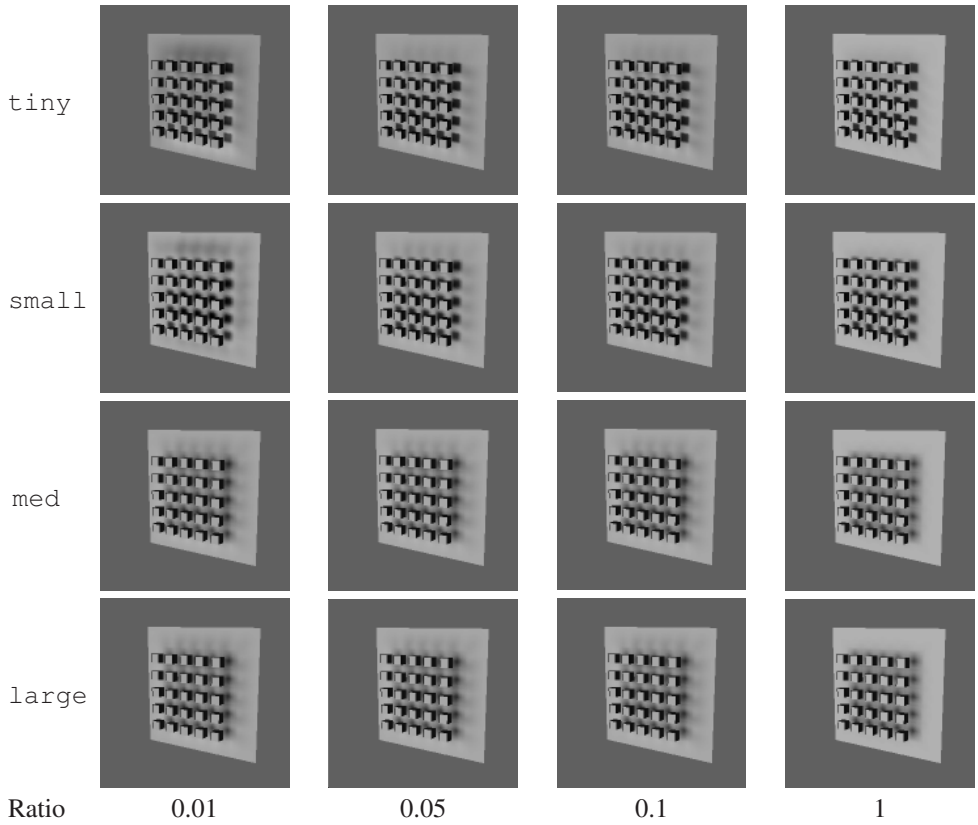

Table 17: Renderings for 'Cubes Scene' with different analytic light sources (y axis) for a fixed input signal resolutions of 64x64 and different nonlinear approximation ratios.

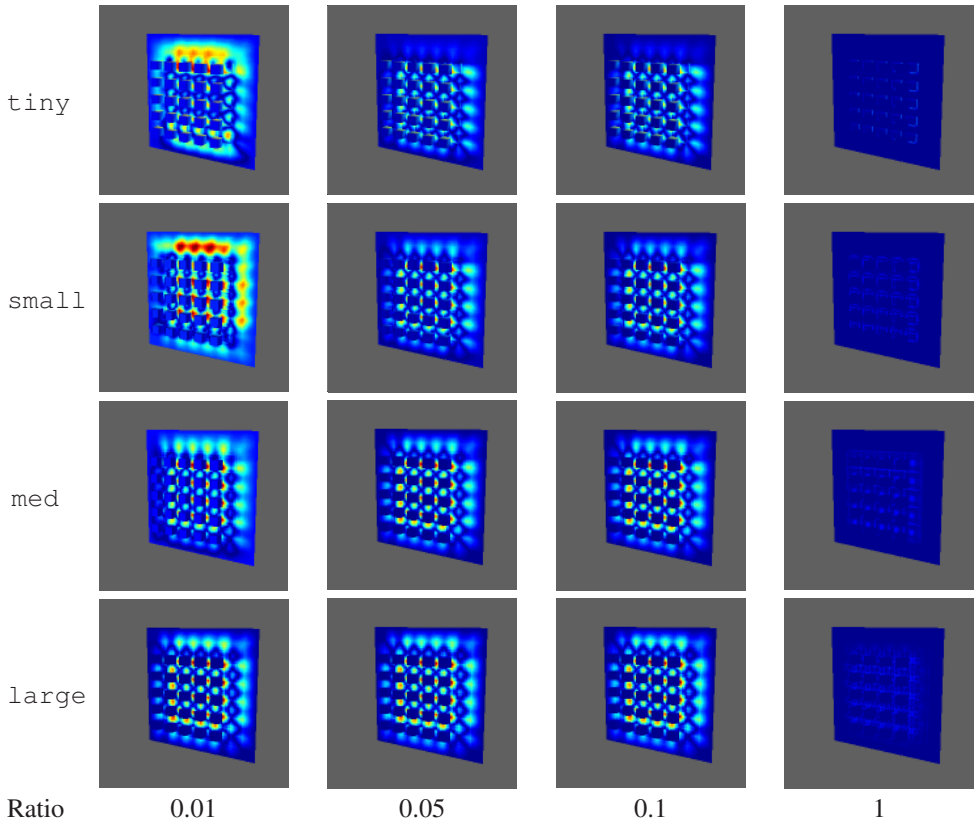

Table 18: Vertex color error for 'Cubes Scene' with different analytic light sources (y axis) for a fixed input signal resolutions of 64x64 and different nonlinear approximation ratios. The shown error is normalized. Dark blue corresponds to very low error, red areas to very high error. Note that error is only normalized over rowsof the table.

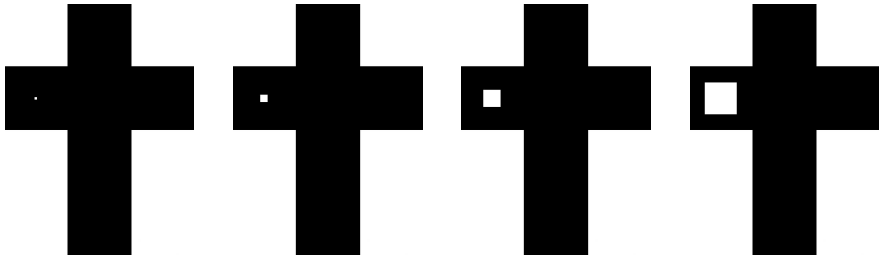

Table 19: Analytic light sources. The value in the brackets denotes the relative size of the light source.

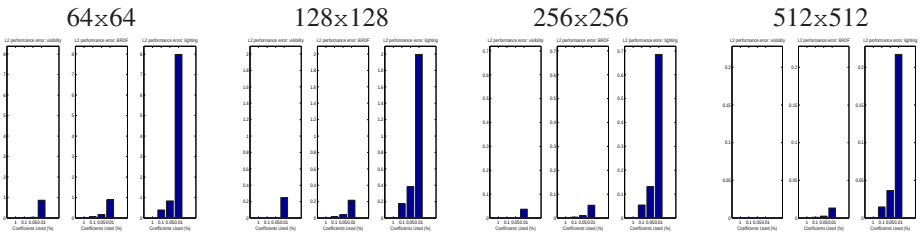

Table 20: L2 error of nonlinear approximations (in percent) for fixed resolutions and varying approximation ratios

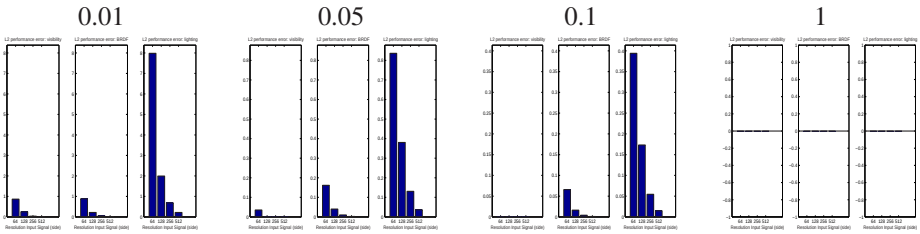

Table 21: L2 error of nonlinear approximations (in percent) for fixed approximation ratios and varying resolutions

9 Matlab functionality

This section contains an overview of the Matlab functions which have been implemented for this report. The functions are grouped according to the directories in the source tree.

9.1 analysis

This directory `analysis` contains functions for computing vertex color error measures between baseline and test renderings and for visualizing these errors. These functions permit the errors and the vertex colors to be visualized directly in the 3D scene, and allow these visualizations to be saved as `vrml` files.

The files `start_compare_*` are driver scripts which set up parameters for the functions `compare` and `compare_fa` which perform the comparisons and visualizations.

9.2 brdf

- `function fs = lambertianHC(res)`
  Returns a sampling of the lambertian BRDF along 5 faces of the hemicube. The BRDF is oriented assuming a surface normal pointing in the positive y direction. The bottom face is omitted because the lambertian BRDF is zero whenever the angle between the view direction and the normal is greater than $\frac{\pi}{2}$ degrees.
- `function fs = lambertianHC_fast(res, n)`
  Similar to `lambertianHC.m` but has been vectorized, and is therefore faster. Also allows the normal to be passed as a parameter.

9.3 filters

- `uc = conv1D( u, fil)`
  Performs 1D convolution. Both the signal and filter have to be one dimensional.
- `fc2D = wcoeffs2D( fc_h, fc_v)`
  Computes the matrix of 2D filter coefficients resulting from the tensor product of `fc_h` and `fc_v`. The tensor product is also called the outer product.
- `[a, d] = wdecomp( sig, w_type)`
  Perform a one level wavelet decomposition.

9.4 haar

- `[lighting, l2p_l] = cacheLighting( lighting_file, res, pcoeffs_keep)`
  Projects all color channels for all faces of the hemicube into the Haar wavelet basis, saves the result and also returns it. The name of the file into which these results are saved is returned by `getFilenameLighting(lighting_file, res, pcoeffs_keep)`.
- `C = ddecHaar(X, n)`
  Performs a full 1D Haar wavelet decomposition. The implementation is based on [Stollnitz 1996].
- `[C, a, dh, dv, dd, n] = ddecHaar2D( X, n)`
  Performs a multi-level 2D Haar wavelet decomposition. Note that coefficients are stored in vectors with row order precedence.
- `[a, dh, dv, dd, n] = ddecHaar2DFast( X, n)`
  Similar to `ddecHaar2D` but makes use of `dwtHaar2DFast` to perform a single-level 2D Haar wavelet decomposition.

- `[a, dh, dv, dd, n] = ddecHaar2DVF( X, n)`
  Identical to `ddecHaar2DFast` but makes use of `dwtHaar2DVF` to perform a single-level 2D Haar wavelet decomposition.
- `[approx, details, n] = ddecHaar2DVFD( img)`
  Performs the full Haar wavelet decomposition for a given image.
- `[a, dh, dv, dd] = dwtHaar2D( X, visualize, check_result)`
  Performs one level of the 2D discrete Haar wavelet decomposition on the given image. May also be used to visualize the result of the decomposition, and to verify the result against the native Matlab routines.
- `[a, dh, dv, dd ] = dwtHaar2DFast( X)`
  Identical to `dwtHaar2D`, but makes use of `dwtHaarFast` to perform one step of the 1D Haar wavelet decomposition.
- `[a, dh, dv, dd ] = dwtHaar2DVF( x)`
  Also identical to `dwtHaar2D` but makes use of Matlab's built-in `conv2` function for extra efficiency. This is very fast.
- `[approx, detail] = dwtHaar( X)`
  Performs one level of the Haar wavelet decomposition.
- `[approx, detail] = dwtHaarFast( X)`
  Itentical to `dwtHaar`, but has been vectorized and is therefore faster.
- `[lighting, l2p_l] = precompLighting( lighting_file, res, pcoeffs)`
  Performs a full 2D Haar wavelet decomposition for each color channel of each face of the lighting map and stores the coefficients in a `.mat` file.
  The triple product integral is computed in the global coordinate frame and so the lighting map is vertex independent. Therefore, precomputation only needs to be performed once and thereafter the lighting coefficients may be accessed cheaply from the `.mat` file when processing each vertex.
- `[tcoeffs_all_case2, tcoeffs_all_case3] = precompTCoeffs( root, res)` Get precomputed tripling coefficients for triple-integral product.
- ```
  precompVertRangeHaar( directory,
                        lighting_file, ...
                        basename_verts, ...
                        basename_output, ...
                        approx_rate, ...
                        verts_start_in,
                        verts_end_in, ...
                        remote_in, ...
                        res_in ...
                       )
  ```
  Precomputes the lighting coefficients if this has not already been done. Also precomputes the decomposition coefficients for the BRDF and visibility maps at each vertex in the scene with an index in the given range.

9.5 `util`

- `m = loadHemicube(filename)`, loads a run length encoded hemicube visibility map and returns a $(5, h, w)$ tensor containing the hemicube face data.
- `m = loadFace(filename)`, this is the same as above, but loads a single hemicube face.
  ```

- `showHemicube(m)`, displays the hemicube data in the usual cross format.
- `showFace(m)`, displays a hemicube face loaded with `loadFace`.
- `saveHemicubeCrossImage(m)`, saves the hemicube cross data as a `.tiff` file in the same cross format used in `showHemicube`.
- `diffs = lpdiff(A, B)`, given two image files, this script generates a laplacian pyramid for each and calculates the RMS error between the layers.
- `[data, displayable] = readPFM(filename)`, Read an image file in pfm format. Pfm is a simple format for HDR images. Examples are available from http://www.debevec.org/Probes/, and a description of the format can be found under http://netpbm.sourceforge.net/doc/pfm.html.
  - `filename` is the name of the pfm file data floating point data of the file.
  - `displayable` is a clamped version of data (to [0,1]) so that it can be displayed directly with `image(displayable)`.

9.6 External Libraries

- `matlabPyrToolbox` Library to compute hierarchical decompositions of a signal.
- `MCGL` Matlab Computer Graphics Library, we employed the `vrml` file writer from this library.