Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Master's Thesis

Global Illumination Computation for Augmented Reality

Bc. Tomáš Nikodým

Supervisor: doc. Ing. Vlastimil Havran, Ph.D.

Study Programme: Open Informatics

Field of Study: Computer Graphics and Interaction

May 10, 2012

Acknowledgements

I would like to thank to the supervisor of my thesis, doc. Ing. Vlastimil Havran, Ph.D. for his advices and for pointing me in the right direction. To my parents, my girlfriend, and to everyone who positively influenced my life.

Declaration

In Prague on April 15, 2012 ....

Abstract

In order to merge virtual objects into a real scene seamlessly, it is important to maintain consistent common illumination. In this thesis, we propose a framework that captures high dynamic range light probes and decomposes them into sets of directional light sources in real-time. The light sources, captured and processed on a dedicated device, are than made available to rendering engines via a server that provides wireless access. We implement three different importance sampling techniques and compare them in terms of the quality of sampling pattern, temporal coherence, and performance.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter gives an overview of this thesis and reviews related work. In Chapter 2, we take a closer look at the related work and provide background knowledge about the problems and techniques related to our project.

1.1 Overview

This thesis gives an overview of problems and techniques related to *augmented reality*. In particular, we focus on image-based lighting techniques and their use for real-time rendering with online light probe acquisition. We discuss existing importance sampling techniques and their suitability for dynamic environment sequences. We then implement three of these techniques and compare them in terms of the quality of sampling pattern, temporal coherence and performance. We also propose a framework based on a client-server architecture that attempts to simplify the integration of dynamic image-based lighting into existing rendering engines. We test our implementation on several testing applications and on the *Zora* augmented reality platform.

Our framework is of benefit in particular to augmented reality applications. The ability to lit virtual objects based on the lighting conditions of the surrounding environment adds realism to the augmented scene and enhances the user experience.

1.2 Related Work

An in-depth overview of augmented reality techniques is given in Bimber et al. [1]. Background information from optics and geometry, as well as an overview of stereoscopic displays and augmented reality platforms are presented in his book. A classification and survey of illumination methods for mixed reality is presented in Jacobs and Loscos [2]. A pioneer research on common illumination between real and computer generated images is presented in Fournier et al. [3]. A more modern approach to common illumination using global illumination computation and high dynamic range imaging was proposed by Debevec et al. [4]. Their method is based on Fournier's differential rendering algorithm. An algorithm to rapidly generate shadows in augmented reality settings is presented in Gibson et al. [5]. A real-time algorithm to detect shadows based on edge detection and to generate virtual shadows consistent with the real scene was proposed by Jacobs et al. [6]. A method based on Fournier's differential rendering algorithm and Debevec's extension of this algorithm was proposed by Knecht et al. [7]. Their method uses instant radiosity introduced by Keller et al. [8] with imperfect shadow maps proposed by Ritschel et al. [9] to approximate global illumination in real-time.

Image-based lighting techniques are overviewed in Reinhard's book [10]. High dynamic range acquisition and display techniques are explained in detail and basic principles of image-based lighting are presented. Many importance sampling techniques for static environment mapping have been proposed over past years. The most influential of these methods include structured importance sampling proposed by Agarwal et al. [11], an algorithm based on Lloyd's relaxation proposed by Kollig et al. [12], hierarchical importance sampling algorithm, based on Penrose tiling, proposed by Ostromoukhov [13], and median cut sampling algorithm proposed by Debevec [14]. Unfortunately, these methods does not work so well for dynamic environment sequences due to poor temporal coherence. An importance sampling method for dynamic environment maps was proposed by Wong et al. [15]. Their method exhibits strong temporal coherence, provides control over the number of samples taken for each frame and operates at real-time frame rates. A probability distribution function based importance sampling method for dynamic environment maps was proposed by Havran et al. [16]. In order to improve the temporal coherence of consecutive frames, they use two low pass filters to normalize the intensity of light sources and suppress high frequency movements. Their method achieves real-time performance and the number of samples can be adaptively changed.

Scalable parallel programming and optimization techniques for GPU using CUDA are discussed in the following articles: Nickolls et al. [17], Ryoo et al. [18] and Luebke et al. [19].

Description of the Spinnstube's hardware, an augmented reality platform similar to *Zora*, can be found in Wind et al. [20].

1.3 Structure of this thesis

In Chapter 2, we review the related work and give theoretical background of global illumination and augmented reality techniques. In Chapter 3 we analyse the problems and propose a solution. In Chapter 4 we take a look at the details of our implementation. Finally, in Chapter 5, we present the results of our work.

Chapter 2

Theoretical background

In this section, we review the related work and provide theoretical background about related problems and techniques. Firstly, we take a brief look at optics and ray shooting. We then describe some of the most common global illumination techniques. Then, we discuss image-based lighting for static and dynamic environments. Finally, we review illumination methods for augmented reality.

2.1 Optics

In this section, we give an introduction into optics. Firstly, we recall very briefly radiometry and photometry. We then discuss various surface reflectance models. In particular, we look at the bidirectional reflectance distribution function.

2.1.1 Radiometry and photometry

Radiometry is the science of measurement of radiant energy, including visible light. Radiometric techniques measure the radiant power in watts, as opposed to photometric techniques that measure the perceived brightness of light to the human visual system. In photometry, the radiant energy is weighted by a visual sensitivity function (depicted in Figure 2.1). The luminous power is measured in lumens. The SI base unit of photometry is candela, a unit of luminous intensity that measures the luminous flux per unit solid angle (lumen per steradian).

2.1.2 Surface reflectance

In computer graphics, the surface reflectance is usually described by a bidirectional reflectance distribution function (BRDF). It gives a ratio of the reflected radiance in direction ω_o to the incident radiance from direction ω_i. Because the directions ω_o and ω_i itself are

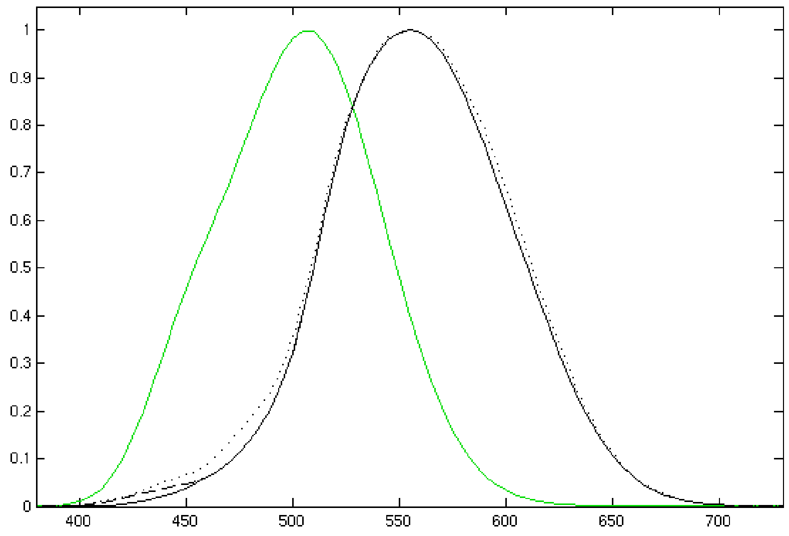


Figure 2.1: Photopic (daytime-adapted, black curve) and scotopic (darkness-adapted, green curve) luminosity functions. The solid black curve is the CIE 1931 standard [21]. The horizontal axis is wavelength in nm.

two-dimensional (azimuth angle ϕ and zenith angle θ), a BRDF is a four-dimensional function. It was first defined by Fred Nicodemus in 1965 [22]. A modern definition of BRDF is given in equation 2.1. An illustration is given in Figure 2.2.

$$f_r(\omega_i, \omega_o) = \frac{dL_r(\omega_o)}{dE_i(\omega_i)} = \frac{dL_r(\omega_o)}{L_i(\omega_i) cos\theta_i d\omega_i} \tag{2.1}$$

where:

ω_i is the incident angle,

ω_o is the outgoing angle,

L is the radiance,

E is the irradiance,

θ_i is the angle between ω_i and the surface normal.

Several more general surface reflectance models exist. For example, the bidirectional surface scattering reflectance distribution function (BSSRDF) takes into account internal scattering of light known as the subsurface scattering. For transparent surfaces, the light refraction is often describe by the bidirectional transmittance distribution function (BTDF). To account for effects such as luminescence, the reflectance functions need to be further extended by two dimensions - the wavelength of incident and outgoing light.

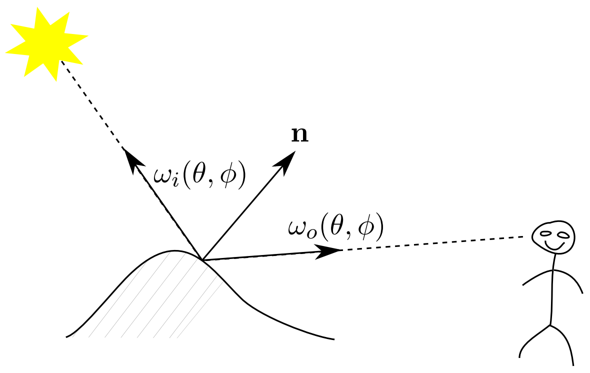


Figure 2.2: Diagram illustrating the bidirectional reflectance distribution function (BRDF).

2.2 Ray shooting algorithm and data structures

Ray shooting is a vital algorithm used in many computer graphics applications and beyond. For example, image synthesis algorithms use ray shooting to simulate the propagation of light. The ray shooting algorithm is also often used for visibility testing (e.g. in radiosity, refer to Section 2.3.3 for more details). It can also be used to find pixel-surface correlation in common illumination computation for augmented reality (refer to Section 2.6.2).

Given a ray and a set of objects, the task of a ray shooting algorithm is to determine the first ray-object intersection, or conclude that none exists. Naive implementation, processing all existing objects, would require $O(N)$ time. Given the number of objects is generally huge, linear time complexity is unacceptably high. A wide range of acceleration data structures exist, reducing the time complexity significantly. The average-case time complexity of a single query for most hierarchical data structures is $O(log_2N)$. The two most popular acceleration data structures for ray shooting are **kd-trees** (generally best performance for static scenes, although the object distribution can have a huge impact on performance) and **bounding volume hierarchies** (more suitable for dynamic scenes, as they are easier to update - often used for collision detection). Other acceleration data structures to consider are uniform and hierarchical grids (suitable for scenes with uniform distribution of objects, poor performance if the distribution is skewed), and octrees (special case of binary space partitioning structures, but performance usually worse than kd-trees).

A detailed discussion of ray shooting is given in Havran's thesis [23]. For further reading about ray shooting and the various acceleration data structures, refer to the following publications [24, 23, 25, 26].

2.3 Global illumination

Global illumination adds realism to computer generated images by considering indirect illumination. As opposed to direct illumination, that is, illumination coming directly from a light source, indirect illumination is any incident of light bouncing from other surfaces in the scene, not directly emitted by the surface. Although theoretically light reflection, refraction and shadows are effects of global illumination, in practice these are usually considered local effects. Throughout this paper, we will follow this practice and consider global illumination to refer only to the simulation of diffuse inter-reflections and caustics.

In this section, we describe the major global illumination algorithms. Radiosity is described in Section 2.3.3. In Section 2.3.4, photon mapping is described. Finally, path tracing is described in Section 2.3.2. All of these techniques, to some extent, use ray shooting and associated data structures discussed in Section 2.2. Many rendering systems based on global illumination also use image-based lighting, discussed in Section 2.4.

It should be pointed out that the algorithms discussed in this section generally do not achieve interactive frame rates and are thus of little use in real-time augmented reality applications. Illumination methods for augmented reality are discussed in Section 2.6. Nevertheless, the readers are encouraged to read this section to gain better understanding of the general concepts of global illumination.

2.3.1 Rendering equation

The rendering equation [27] is an integral equation that describes the equilibrium radiance in a scene. At each particular point the outgoing light is given as a sum of the emitted light and the reflected light. The reflected light is given as the sum of all incoming light multiplied by the surface reflection and the cosine of the incident angle. The rendering equation is given in equation 2.2.

$$L_o(\mathbf{x},\omega,\lambda) = L_e(\mathbf{x},\omega,\lambda) + \int_\Omega f_r(\mathbf{x},\omega',\omega,\lambda) L_i(\mathbf{x},\omega',\lambda)(-\omega'\cdot\mathbf{n})d\omega' \tag{2.2}$$

where:

$\mathbf{x}$ is a particular position,

ω is a particular outgoing direction,

λ is a particular wavelength,

$L_o(\mathbf{x},\omega,\lambda)$ is outgoing light,

$L_e(\mathbf{x},\omega,\lambda)$ is emitted light,

$\int_\Omega \ldots d\omega'$ is an integral over a hemisphere of incoming directions,

$f_r(\mathbf{x},\omega',\omega,\lambda)$ is the bidirectional reflectance distribution function (BRDF),

$L_i(\mathbf{x},\omega',\lambda)$ is incoming light,

$-\omega'\cdot\mathbf{n}$ is the attenuation of incoming light due to incident angle.

The rendering equation generalizes a variety of realistic rendering algorithms. For example, finite element methods such as the radiosity algorithm (described in Section 2.3.3), as well as Monte Carlo methods such as photon mapping (Section 2.3.4), path tracing (Section 2.3.2) and Metropolis light transport (Section 2.3.2.3) all attempt to solve the rendering equation. These methods are described in the following paragraphs of this section.

The rendering equation was first introduced to the computer graphics community in 1986 at a Siggraph conference simultaneously by Kajiya et al. [27], who coined the term *rendering equation*, and by Immel et al. [28]. Although the rendering equation is very general, there are some phenomena that it fails to capture. These are, for example, subsurface scattering, phosphorescence, and fluorescence [29].

2.3.2 Path tracing based algorithms

Path tracing algorithm simulates the physical behaviour of light very closely. Rays cast from camera or light source represent a patch of photons. At every interaction with a surface, only one secondary ray is emitted and recursively traced.

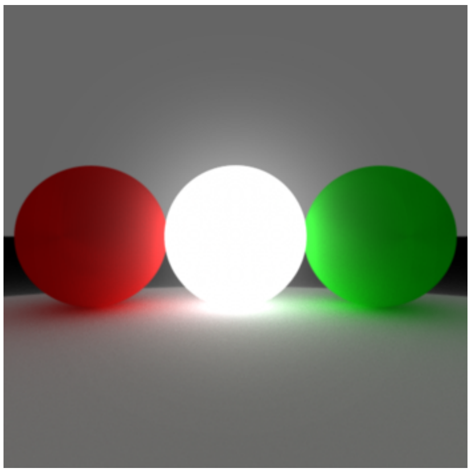


Figure 2.3: A simple scene rendered with path tracing (courtesy of John Carter).

The physical nature of path tracing produces many effects that would have to be specifically added into other rendering algorithms (such as scanline, or even conventional ray tracing). These effects include soft shadows, indirect lighting, color bleeding, caustics and depth of field. The implementation is relatively simple, compared to other methods that would produce the same results. A disadvantage of path tracing is that it is computationally very expensive. Unless a large number of samples per pixel is traced, the resulting image is noisy. The noise is especially disturbing in animated sequences, as it produces a random speckling.

Path tracing is an unbiased algorithm and the accuracy can be improved by increasing the number of samples. It is often used to generate reference images for testing the quality of other rendering algorithms.

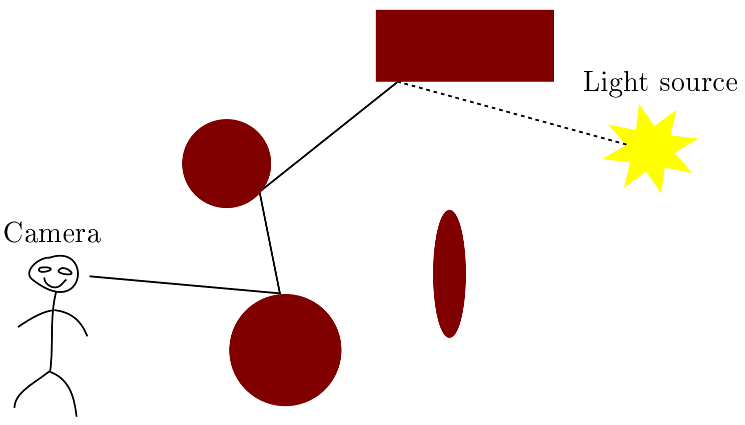


Figure 2.4: Illustration of the path tracing algorithm.

2.3.2.1 Path tracing

In real-world, light sources emit photons, which bounce off surfaces until they are absorbed. The direction of the bounces is random, with probability distribution given by surface material properties. Only a small fraction of photons emitted actually hit the photoreceptors at the retina of human eye, or the chip of a camera. But with the enormous number of photons being emitted, even this small fraction makes a huge number.

It would be computationally infeasible to trace such a large number of rays. To compensate for the lower number of rays, some modifications must be incorporated into the algorithm to minimize the noise. First of all, the rays are traced in the reverse direction, from camera to light sources. Given the relatively small area of light sources, the probability of a ray hitting a light source is still very small. To speed up the convergence, a shadow ray is cast from every interaction of a primary or secondary ray with a diffuse surface to a random point at a random light source (probabilities given by intensity and emission characteristics). This algorithm is outlined in algorithm 1. In practice, much quicker convergence is achieved with bidirectional path tracing algorithm, described in Section 2.3.2.2.

In contrast to conventional ray tracing, only one ray is traced recursively for every ray-surface interaction. The secondary ray is traced regardless of the material properties, unlike in conventional ray tracing which traces secondary rays only if the surface is reflective or refractive.

2.3.2.2 Bidirectional path tracing

Tracing the paths in only one direction would require computing a very large number of bounces to get any useful information. If the ray bounces randomly across the scene, the probability of it hitting a light source is very low or zero for point light sources.

This problem is compensated for by bidirectional path tracing. Rays are traced in both

Algorithm 1 Path tracing

```
TracePath(Ray, depth)
{
  if(depth >= MaxDepth)
    return Black

  X = FindNearestIntersection(Ray, Scene)
  if(NoIntersectionExists)
    return EnvironmentMap(Ray)

  Emission = IntersectedObject.Material.Emission

  RandomDirection = ImportanceSampling(X)
  SecondaryRay = (X, RandomDirection)

  RandomLightSource = ImportanceSampling(LightSources)
  RandomSample = ImportanceSampling(RandomLightSource)
  ShadowRay = (X, RandomSample)

  if(NoIntersectionExists(ShadowRay))
    Emission += EmittedLight(RandomLightSource, RandomSample)

  return Emission + TracePath(SecondaryRay, depth+1)
}
```

directions, from camera and from light sources. At every interaction with a surface, the paths are combined, joining the prefix of every path from camera with the suffix of every path from light sources.

This process is illustrated in Figure 2.5. In this example, only one bounce was computed in both directions. By joining these two paths, four distinct paths were created, each starting at the camera and ending at a light source.

2.3.2.3 Metropolis light transport

Metropolis light transport algorithm is an extension to bidirectional path tracing, which in some cases converges to the solution of the rendering equation quicker than if using the bidirectional path tracing method on its own.

The algorithm constructs paths from camera to light sources in the same way as bidirectional path tracing. The nodes of the path are stored in a list. The algorithm then uses some statistical calculations to modify the path by adding extra nodes to create new paths. Each

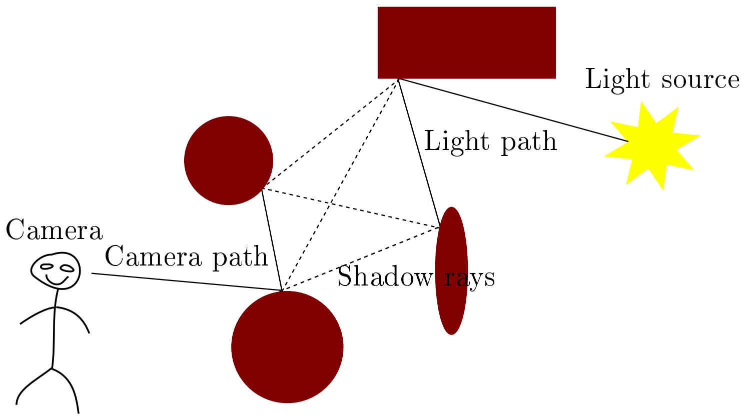


Figure 2.5: Illustration of bidirectional path tracing.

such mutation of the path is either accepted or rejected with a certain probability, ensuring that the paths are sampled according to the contribution they make to the ideal image [30].

By exploring the path space locally with modifications to the already found paths, the cost per sample is reduced. It also helps to reduce noise. On the other hand, even though the modifications are chosen in a way to improve the convergence of the solution, in some cases other unbiased methods provide quicker convergence.

2.3.3 Radiosity

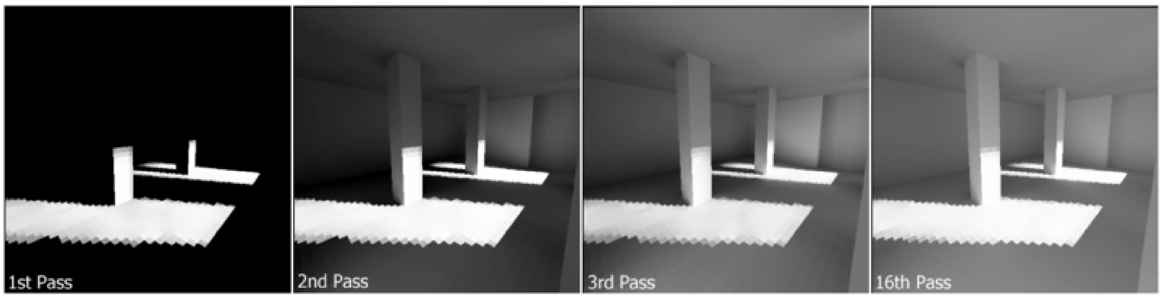


Figure 2.6: Progress of the radiosity algorithm (courtesy of Hugo Elias).

Radiosity algorithm attempts to solve the rendering equation with the finite element method [31]. The scene is divided into a number of patches (small geometric primitives). Then, form factors are computed for each pair of patches. A form factor represents the fraction of light leaving one patch that hits the other. It consists of two parts multiplied together: visibility factor and geometry factor. The visibility factor can have value of either zero (if there is an obstruction between the two surfaces) or one (if the two surfaces can see each other). Usually, a ray shooting algorithm is used to determine the visibility (see Section 2.2). The geometry factor (ranging between zero and one) is a coefficient describing how well the two patches can see each other. If they are oriented in different directions or far away from each other, then the geometry factor, and thus the form factor becomes small.

The form factors are used as coefficients in a set of linear equations. The solution of this $N \times N$ set of linear equations gives the radiosity of each patch.

$$B_i = B_{e,i} + \rho_i \sum_{i=1}^{N} B_j F_{ij} \tag{2.3}$$

where:

B_i is radiosity of surface i,
$B_{e,i}$ is emissivity of surface i,
ρ_i is reflectivity of surface i, and
F_{ij} is form factor of surface j relative to surface i.

Solving this set of n equations with a direct method (e.g. Gauss elimination) would take $O(n^3)$ time, which is unacceptable considering the number of elements can be large. Therefore in practice, iteration methods are used, reducing the time complexity to $O(n^2)$. There are two types of iteration methods: energy gathering (Jacob iteration, Gauss-Siedel iteration) and energy shooting (Southwell iteration, known also as progressive radiosity). Due to the fact that most of the energy is usually distributed over a small fraction of patches, progressive radiosity has significantly faster convergence. After each iteration we have immediate radiosity values for every patch for the corresponding bounce level. The pseudocode for progressive radiosity method is given in algorithm 2. The progress of the algorithm is illustrated in Figure 2.6.

An advantage of the radiosity method is that the solution is independent of the viewpoint. For static scenes, the radiosity can be precomputed and then used for each frame of an animated sequence. A disadvantage is that only diffuse surfaces are considered. Thus, phenomena such as caustics cannot be simulated with this method.

Further details about radiosity can be found in Cohen [31]. Stochastic radiosity is explained in Dutre et al. [32].

2.3.4 Photon mapping

Photon mapping is a global illumination algorithm solving the rendering equation in two steps. In the first step, rays are cast from light sources to create a photon map. The second step is where the actual rendering takes place, using the photon map to compute the radiance values.

The photon mapping algorithm is capable of producing multiple global illumination effects, such as refraction of light travelling through transparent objects, interreflection between diffuse surfaces, subsurface scattering, and some effects caused by rays of light travelling through participating media (e.g. smoke). These properties of the algorithm enable simulation of phenomena such as caustics and color bleeding [34].

Algorithm 2 Progressive Radiosity

```
for(i=0; i<N; i++)
{
  B[i] = dB[i] = Be[i];
  while(!converged())
  {
    // pick i with maximal energy dB[i]
    i := argmax(dB);
    // shoot accumulated energy from i
    for(j=0; j<n; j++)
    {
      db = rho[j]*F[j][i]*dB[i];
      dB[j] += db;
      B[j] += db;
    }
    // accumulated energy of i is now zero
    dB[i]=0;
  }
}
```

Photon mapping is a consistent method so convergence to the correct solution of rendering equation can be achieved by increasing the number of rays. Unlike some other global illumination methods (most notably path tracing - see Section 2.3.2) it is a consistent rendering algorithm, so average of many rendering iterations does not converge to correct solution.

2.3.4.1 Construction of photon map

The first step of the photon mapping algorithm is to create a photon map. Rays are cast from light sources and traced in the same way as in the path tracing algorithm (due to reciprocity of the BRDF function) [35]. Light source and direction is selected randomly (based on a probability function). On every ray-surface interaction (unless the surface is a perfect mirror) an entry is added to the photon map, storing intersection point and incoming direction. Then, it is randomly decided (probability based on surface properties) whether the ray is absorbed, reflected or refracted.

Usually, two distinct photon maps are constructed during this phase. The global map ($L[S|D]*D$) contains both direct and indirect lighting. The map of caustics ($LS+D$) contains only indirect lighting [36]. During the process of photon map construction, the photon map is stored as a linear list. When the list is finished, a data structure more efficient for k-nearest neighbour (kNN) search is constructed. The most popular data structure for this purpose is a kd-tree (see Section 2.2 for more details).

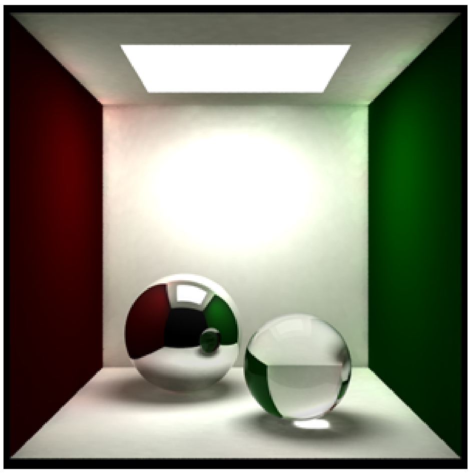


Figure 2.7: Photon mapping produces global illumination effects such as caustics and color bleeding, as illustrated in this image (courtesy of Zack Waters [33]).

2.3.4.2 Rendering

In the second pass of the algorithm, the scene is rendered using the photon map created in the first pass. A distributed ray tracing algorithm is used. For efficiency reasons, the rendering equation is decomposed into four terms: direct illumination, ideal reflection/refraction, caustics, and indirect illumination. The first two do not require the use of a photon map. Direct illumination is computed by sampling of light sources and casting shadow rays. Reflection and refraction is handled by recursive tracing of deterministic secondary rays. For estimation of caustics, a special photon map is used both for primary rays and for secondary rays produced by perfect reflection or refraction. To efficiently compute indirect illumination, a method known as final gathering is used. One level of recursion is executed in the distributed ray tracing. For each secondary ray, the illumination is estimated from global photon map [36].

2.3.4.3 Irradiance caching

Irradiance caching is an optimization technique used to compute the indirect illumination more efficiently. Sampling the hemisphere in final gathering is a costly operation. The idea behind irradiance caching is that there is a strong spatial coherence in the lighting at diffuse surfaces. Thus, some of the computation can be avoided by reusing the data already computed and interpolating between them. When the final gathering is evaluated, the result is stored in a cache. If the point to evaluate can be interpolated from the cached samples with an error below a threshold, the costly process of sampling the hemisphere is avoided. Algorithm 3 illustrates this idea.

Algorithm 3 Irradiance caching

```
GetIrradiance(X)
{
  if(irradiance can be interpolated from cache)
  {
    IRRADIANCE = InterpolateFromCache(X)
  }
  else
  {
    IRRADIANCE = SampleHemisphere(X)
    InsertIntoCache(X, IRRADIANCE)
  }
  return IRRADIANCE
}
```

2.3.4.4 Effects

This section describes some of the phenomena that can be simulated with photon mapping. Figure 2.7 illustrates the capabilities of the photon mapping algorithm.

Caustics. Caustics are patterns caused by reflected or refracted light. Curved highly reflective or refractive surfaces focus incoming light into specific spots, increasing the intensity of light hitting diffuse surfaces at these regions. Figure 2.8 illustrates the caustics created by light refracted at the wavy surface of water (image on the right). The shape of the waves causes the light to concentrate at some places, while reducing the amount of light hitting other places. The other two images in the figure show caustics produced by a glass of water and an ice cube.

Photon mapping can simulate this artifact of real world by tracing photons and recording the locations where photons hit diffuse surfaces after interaction with reflective and refractive surfaces.

Diffuse interreflection. Diffuse interreflection refers to the process of light reflecting off diffuse surfaces and hitting other diffuse surfaces. This becomes especially noticeable when the two surfaces involved have different color. For example, if the light bounces of a red surface, only a specific part of the original spectrum of the light inciding the surface is reflected, the rest is absorbed. The other surface then receives only this part of the color spectrum. This effect is known as *color bleeding* [36]. Suppose we have a red and a white surface next to each other. The part of the white surface nearest the red surface receives light reflected off the red surface, changing it apparent color to red. The color of one surface seems to bleed onto the other surface. This effect is well illustrated in Figure 2.3 (rendered

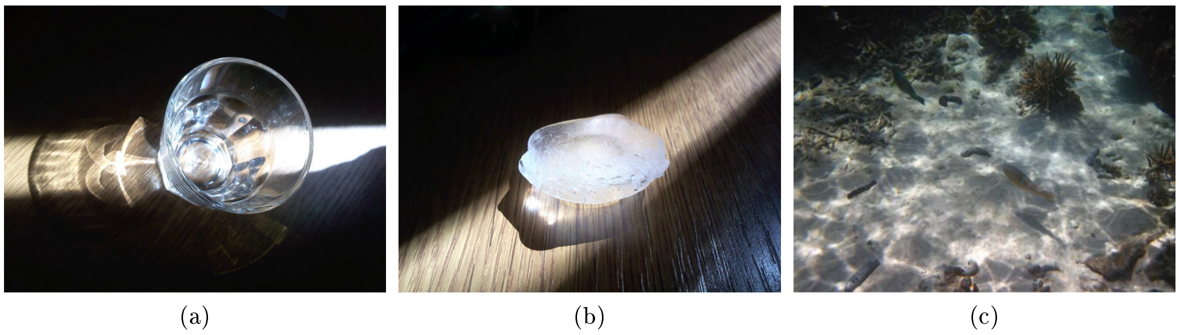


Figure 2.8: Photograph (a) shows caustics produced by a glass of water, photograph (b) shows caustics produced by an ice cube and photograph (c) shows caustics projected onto see floor, produced by waves at the surface of the water.

with path tracing). Another example showing the effect of diffuse interreflection is given in Figure 2.7 (rendered with photon mapping).

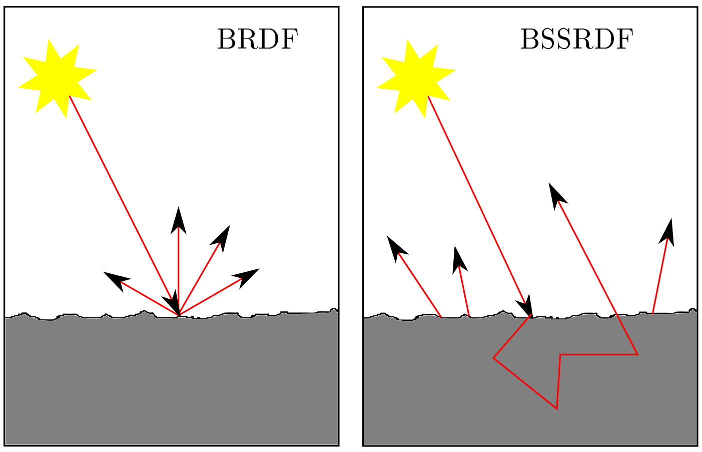


Figure 2.9: Schematic depiction of BRDF vs. BSSRDF.

Subsurface scattering. Subsurface scattering is the effect of light scattering inside the surface of a material before it is reflected [34]. Due to this property of a material, the light leaves the surface at a different position than where it enters. The effect is evident in materials such as human skin, milk, etc. It is especially important in the rendering of human faces, as human visual system is very critical and sensitive to details when evaluating the features of faces of other human beings.

Although photon mapping algorithm is capable of modelling this phenomenon, it becomes computationally expensive for highly scattering materials. Generally, it is preferable to use bidirectional surface scattering reflectance distribution functions (BSSRDFs) as their use is more efficient and produces results visually not too far from reality. The principle of subsurface scattering is depicted in Figure 2.9.

2.4 Image-based lighting

Image-based lighting is a process of using images of real-world as light sources [10]. These are usually high dynamic range images (HDR), attempting to capture the full range of illumination of a real scene. Rendering of computer generated objects with the use of IBL produces realistic appearance of the virtual objects, as if they were placed in the environment where the environment map was captured.

This is particularly useful in augmented reality applications, where we attempt to merge real and virtual objects seamlessly. Image-based lighting also adds realism to virtual scenes and is often used in commercial renderers, including game engines. Recent advances in hardware allow computation of IBL in real-time.

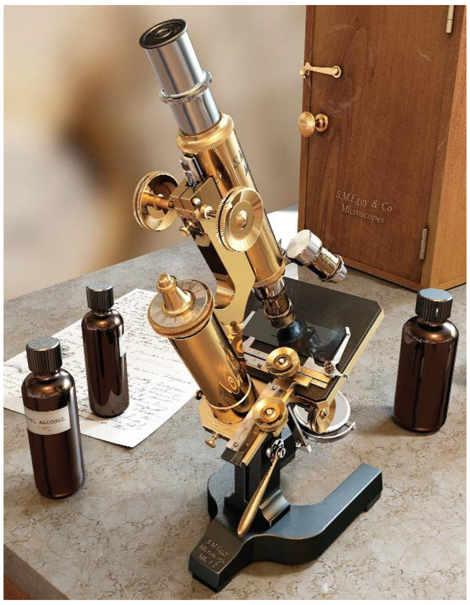
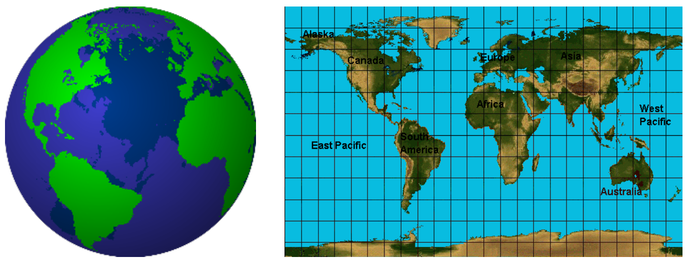


Figure 2.10: Model of a microscope illuminated by light captured in a kitchen, rendered using IBL techniques (courtesy of Paul Debevec [37]).

Figure 2.10 shows a computer generated scene rendered with image-based lighting techniques. The smooth shiny surface of the bottles reflects light coming from the environment map, revealing the appearance of the real environment where the map was captured. Interreflections of the virtual objects are also noticeable, due to the ray-tracing based global illumination techniques that were used to render the image (refer to Section 2.3 for more details on global illumination).

2.4.1 Environment map representation

An environment map stores information about the illumination coming from the surrounding environment. Several techniques to encode this information are used in computer graphics. A very common way of storing this information is the latitude-longitude map [10]. Using this representation, the amount of light coming from a direction $\omega(\theta, \phi)$ can be trivially retrieved by a texture look-up at index (θ, ϕ).



(a) The globe. (b) A latitude-longitude map of the globe.

Figure 2.11: The globe and its latitude-longitude projection.

The latitude-longitude mapping is also commonly used in cartography. In order to project the entire globe onto a 2D plane, latitude-longitude maps are commonly used. Figure 2.11 shows the globe and its projection into a latitude-longitude map. Note the straight lines in the picture on the right; each line corresponds to a particular altitude θ or azimuth ϕ angle, respectively.

2.4.2 Environment map capture

The light probe images can be captured in several different ways. Popular techniques use mirrored spheres, tiled photographs or fish-eye lenses. In this section, we discuss each one of these three techniques.

2.4.2.1 Mirrored spheres

Photographs of a sphere with highly reflective surface capture a field of view of 360 degrees. Part of the visual field is obstructed by the camera and the parts near the edges are poorly sampled. To properly capture incident illumination in all directions, two images need to be taken, usually rotated by 90 degrees. Where one of the images has artifacts, the other yields good quality and combining the two images produces a relatively good light probe of the environment [10]. Figure 2.12 illustrates the process of light probe acquisition with a mirrored sphere.

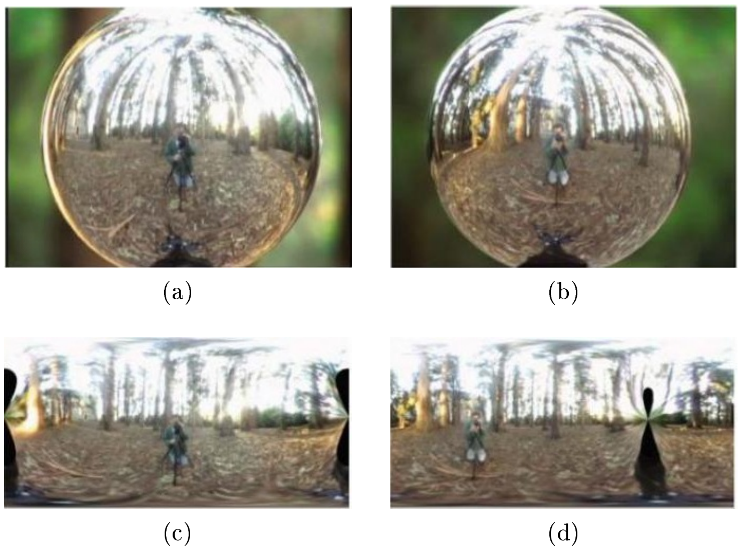


Figure 2.12: Capturing environment map with a light probe. Images (a) and (b) show the photographs of the sphere. Images (c) and (d) show a latitude-longitude map obtained by projection of the respective images (courtesy of Erik Reinhard [10]).

2.4.2.2 Tiled photographs

The light probe of the environment can also be reconstructed from a set of images taken from the same position, facing different directions [10]. This technique, known as stitching, is often used in the production of panoramatic pictures. An advantage of this technique is that it is possible to produce high-resolution images with a standard camera. To prevent the problems of stitching together poorly aligned images, it is important to maintain the same camera position for each frame.

2.4.2.3 Fish-eye lenses

Fish-eye lens cameras provide field of view of 180 degrees or more. It is thus possible to capture omnidirectional light probes by taking two images with such cameras. As most of the light usually comes from the sky, for some applications it is sufficient to capture one 180 degree image facing upwards. Fish-eye lenses are particularly useful for acquisition of video environment maps. Video environment maps and dynamic image-based lighting are discussed in Section 2.5.

2.4.3 Use of environment maps in rendering

In this section we discuss techniques used for sampling of light probe images. Firstly, we explain the terms light source identification and light source constellation. Secondly, we describe major importance sampling methods and discuss their advantages and drawback in context of environment map sampling.

2.4.3.1 Light source identification

The global illumination algorithms described earlier in Section 2.3 compute direct illumination by casting shadow rays directly to light sources. Random sampling of hemisphere is used only for indirect illumination. It is thus important to know the position of light sources. Sampling the environment map as a source of indirect illumination requires a large number of ray samples and is thus inefficient. The environment often contains spots of concentrated illumination such as the sun in the sky and if the sampling algorithm does not know position of these spots, the visual quality of the results is very low unless a huge number of random samples is taken to adequately sample these areas.

The idea of light source identification is that if the spots where illumination is concentrated are identified, they can be sampled more thoroughly or used directly as light sources. Corresponding areas of the environment map are then either removed and the modified environment map is sampled in the indirect lighting computation [10], or the entire environment map is transformed into a sequence of directional light sources, as described in the following section (2.4.3.2).

2.4.3.2 Light source constellation

The idea of light source identification, discussed in previous section (2.4.3.1), can be taken further so that the entire environment map is transformed into a constellation of light sources. Care needs to be taken to select a representative sequence of light sources, so that the aliasing and artifacts caused by these approximations are minimized. If a suitable importance sampling algorithm is used, a good approximation can be achieved with a manageable number of light sources. These light sources can be used in either traditional scanline algorithm or in global illumination computation. Some of the importance sampling techniques are discussed in the following section (2.4.3.3). The environment map is still sampled directly in the computation of mirror-like reflection and refraction.

2.4.3.3 Importance sampling

In this section, we discuss importance sampling techniques in the context of environment map sampling. Most of these techniques can be used in general to sample any two-dimensional domain and are useful in many different applications.

The process of converting an environment map into a set of light sources often involves dividing the environment map into a number of regions. Each of these regions is represented by a light source with intensity and color corresponding to the region of the EM. Either directional or area light sources can be used. Alternatively, the environment map can be sampled based on a probability distribution function.

A naive method, using a uniform division of the environment map, would oversample dark regions and undersample bright regions, leading to inefficient and aliased computation. Generally, it is desirable to represent regions of high intensity of illumination with more light sources than low intensity regions. A simple yet efficient solution uses a hierarchical subdivision based on the median cut algorithm. An algorithm proposed by Debevec [14] and inspired by Heckbert's median cut color quantization algorithm [38] constructs 2^n light sources from a light probe image in latitude-longitude format in the following way:

1. Add the entire light probe image to the region list as a single region.
2. For each region in the list, subdivide along the longest dimension such that its light energy is divided evenly.
3. If the number of iterations is less than n, return to step 2.
4. Place a light source at the center or centroid of each region, and set the light source color to the sum of pixel values within the region.

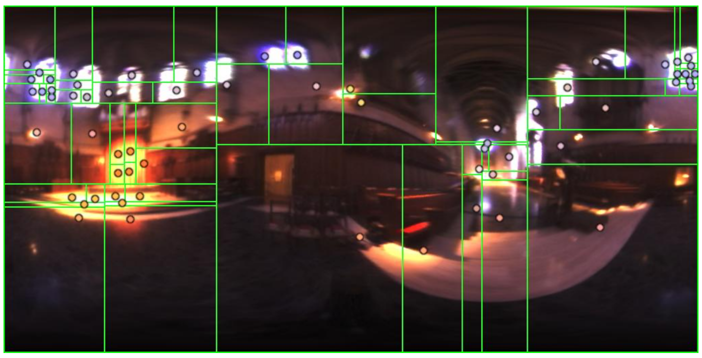


Figure 2.13: The Grace Cathedral light probe subdivided into 64 regions of equal light energy using the median cut algorithm. The small circles represent the positions of the light sources placed at the centroids of each region (courtesy of Paul Debevec [14]).

This method is very fast compared to most importance sampling techniques. A drawback of this approach is that angular extent is not taken into account. Small regions of concentrated illumination that could be well approximated with a small number of light sources are oversampled. Dark areas are not sampled sufficiently, which leads to visible artifacts in rendering, particularly when bright light sources are occluded. A light probe image subdivided into 64 regions using this algorithm is given in Figure 2.13. Figure 2.14 presents the results of using various numbers of samples compared to Monte Carlo integration.

Ostromoukhov et al. [13] proposed a fast hierarchical importance sampling algorithm that takes into account area of regions as well as intensity of illumination. Their method, based

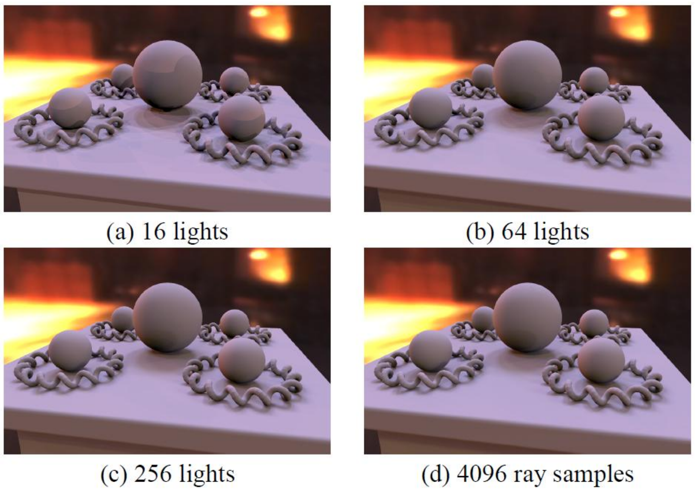


Figure 2.14: Comparison of results for a scene lighted with the above Grace Cathedral environment map. Images (a,b,c) were rendered with 16, 64, and 256 direct light sources. Image (d) was rendered with Monte Carlo integration using 4096 random rays per pixel (courtesy of Paul Debevec [14]).

on Penrose tiling, exhibits blue noise properties. The sampling pattern produced with this method have more appropriate spatial distribution than the median cut algorithm described earlier in this section. This technique is also very fast and the sampling time grows linearly with the number of samples. A light probe image can be sampled with hundreds of samples (light sources) in just a few milliseconds. This is very attractive for dynamic scene lighting, where light probe images need to be sampled at interactive frame rates. The technique is also deterministic and for consecutive frames exhibit quite good temporal coherence. A more thorough discussion of importance sampling methods and their suitability for static and dynamic environment mapping is given in sections 2.5.2.2 and 2.5.2.3.

2.5 Image-based lighting using dynamic environment sequences

Existing importance sampling techniques mainly focus on static environment maps. Little effort has been put to design sampling algorithms exhibiting strong temporal coherence. Even local changes often lead to global differences in positions of samples. This results in unwanted flickering artifacts in animated sequences. Many existing methods also assume offline processing and do not attempt to achieve interactive frame rates.

The rationale behind decomposing environment maps into a set of directional light sources have been discussed in Section 2.4. In this section, we focus on problems related to video environment maps (VEM). We start by introducing techniques of HDR image acquisition (Section 2.5.1). Then, in Section 2.5.2, we discuss the requirements on importance sampling techniques that need to be met in order for them to be useful in dynamic settings. We also explore algorithms proposed for static scene lighting and evaluate their suitability for dynamic scenes. Finally, we take a close look at some importance sampling algorithms for dynamic scenes.

2.5.1 High dynamic range light probe acquisition

The human eye is capable of capturing a dynamic range of luminance of more than 9 orders of magnitude. The luminance of a typical outdoor scene ranges from less than 10^{-4} lux (moonless overcast night sky) to more than 10^5 lux (direct sunlight) [10]. Standard cameras are incapable of capturing this range of luminances in a single shot.

Traditionally HDR images are captured with standard LDR cameras by taking multiple images from the same viewpoint with varying exposure times. If the exposure time of every successive image taken doubles, the full range of illumination in the scene can be captured with about 6 to 9 images, depending on the scene [10]. This set of images is then aligned and merged together. Pixels that are saturated in slow shutter images are exposed properly in images taken with lower exposure time and vice versa. If the scene contains moving objects, than the process of fusing is more challenging, although some techniques, known as ghost removal, exist [10].

Modern cameras have sufficient computational resources to perform exposure fusion on-chip. A problem is that most of the manufacturers do not provide a programmable interface to fully exploit the functionality of the device. Recently, an API have been developed for the Nokia N900 smartphone [39] that allows the programmer to fully control the camera processing pipeline. It has been shown that the device can be programmed to capture multiple exposure image and fuse them into a high dynamic range image in real-time [40].

Recently, HDR video sensors have been developed that are capable of capturing HDR images directly. These sensors are very attractive for video environment map acquisition. Their rapidly dropping cost suggests that they will soon be used in many image-based lighting applications. Examples of HDR video sensors are Digital Pixel System (Pixim), Autobrite and HDRC [41].

2.5.2 Importance sampling of video environment maps

Firstly, in Section 2.5.2.1, we highlight the features that are important for efficient VEM sampling, as opposed to sampling of static EMs. Secondly, in Section 2.5.2.2, we review existing methods for static environment mapping and evaluate them based on these requirements. Methods designed to work well for dynamic environment mapping are reviewed in Section 2.5.2.3. The importance sampling algorithms for static environment mapping are described in more details in Section 2.4.3.3. Finally, in Section 2.5.2.4, we describe the PDF-based importance sampling methods in detail and in Section 2.5.2.5 we look at the Q2-Tree importance sampling method in detail.

2.5.2.1 Requirements

Apart from features desirable in static image-based lighting, such as a sampling pattern that well represents the environment, a good importance sampling algorithm for dynamic scene lighting should have the following properties [16]:

- *Temporal coherence.* The algorithm should produce similar sampling patterns for consecutive frames. If local changes in the environment map lead to global changes in the entire sampling pattern, it will cause popping artifacts and jumping of shadows in the rendered sequence.
- *Fast computation.* Acquisition of HDR environment map as well as decomposition of the map into directional light sources need to be computed in time of order of milliseconds. Clearly, if the EM processing algorithm is to be used in real-time rendering, it needs to run in real-time as well.
- *Control over number of samples.* To maintain constant frame rate of the animation while maximizing visual quality, it is useful to adaptively change the number of light sources. For simple scenes, it is desirable to add more light sources to enhance the quality while for complex scenes it might be necessary to reduce the number of light sources to maintain real-time performance.

2.5.2.2 Existing methods for sampling of static environment maps

Structured importance sampling proposed by Agarwal et al. [11] combines stratified and importance sampling. They proposed an importance metric that takes into account both surface area and integrated illumination of a region (refer to equation 2.7). The number of samples can be easily controlled. For dynamic sequences, temporal coherence of sampling pattern is poor. The algorithm places samples based on a threshold which may change from frame to frame due to local changes, leading to global changes of sampling pattern in the entire map. The computation time, dozens of seconds for a single environment map, is far from interactive.

Kollig et al. [12] proposed an algorithm based on Lloyd's relaxation. Their method yields good results for static environment maps. The number of samples can be easily controlled. The time to process single EM is dozens of seconds; too high for real-time applications. Also, the algorithm exhibits very poor temporal coherence as even local changes can cause global changes in the sampling pattern of the entire EM.

The hierarchical importance sampling algorithm proposed by Ostromoukhov [13], based on Penrose tiling, can operate at interactive frame rates. Furthermore, the sampling pattern produced by this algorithm exhibits relatively good temporal coherence. A problem with this method lies in the difficulty to control the number of samples, which is dependent on the environment map being sampled.

The median cut sampling algorithm proposed by Debevec [14] is fast enough to be used in interactive systems. A problem with this method is that it exhibits poor temporal coherence between consecutive frames of animation. Localized changes in illumination effect the entire sampling pattern. Another drawback of this algorithm is that the control over the number of samples is limited. At each level of subdivision the region is further subdivided into two regions of the same total illumination so the number of samples taken is 2^n, where n is the chosen depth of subdivision.

2.5.2.3 Existing methods for sampling of dynamic environment maps

Havran et al. [16] proposed a probability distribution function (PDF) based importance sampling method for dynamic environments. They use an inverse transform method for hemispheres proposed by Havran et al. [42]. It is similar to the standard inversion procedure, used for importance sampling of static environment maps by Pharr [43] and Burke [44]. However, the inverse transform method proposed by Havran et al. exhibits better continuity and uniformity, which leads to better stratification of the resulting sample positions. To handle dynamic environments, Havran et al. use two low pass filters to improve temporal coherence. The first filter normalizes the intensity of light sources to preserve total energy of the system. The second filter suppresses high frequency movements of light sources. Invisible light sources elimination and light source clustering methods are used to improve rendering performance. Their method exhibits good temporal coherence, real-time performance and the number of light sources can be adaptively changed. The method is described in detail in Section 2.5.2.4.

Wong et al. [15] proposed an importance sampling method for dynamic environments. Their method is based on quadrilateral subdivision of a sphere. Firstly, the sphere is mapped into 2D space using the HEALPix mapping [45, 46]. A quad tree is adaptively constructed over the quadrilaterals (referred to as *quads*). At every step, the region with highest importance is further subdivided into four quads. This process is repeated until required number of samples is reached. Adaptive changes of the number of samples are made very easy. The method is fast and the results for static scenes are comparable with methods such as structured sampling [11] and Penrose-based sampling [13]. It exhibits very good temporal coherence, which makes this method well suited for dynamic scene lighting. The algorithm is explained in details in Section 2.5.2.5.

Spatio-temporal sampling proposed by Wong et al. [47], based on Q2-tree sampling proposed by Wong et al. [15] six years earlier, further exploits temporal and spatial coherence of environment sequences. Their method treats an environment sequence as a volume constructed by stacking up all the frames in the chronological order. At each iteration of the algorithm, it is decided whether to perform a binary split in the time domain or a quad split in the spatial domain. The cost of such split is determined based on a novel importance metric. The proposed method produces a temporally coherent sampling pattern with slightly better characteristics than the original Q2-tree sampling algorithm. A limitation of this method is that the entire environment sequence needs to be known in advance. For this reason, the method cannot be adopted for augmented reality applications requiring online processing of video frames in real-time.

2.5.2.4 Probability distribution function based importance sampling methods

In this section, we discuss in detail the importance sampling methods based on probability distribution functions (PDF).

A common approach to PDF sampling is via the standard inversion procedure. Below, we illustrate the procedure on a 1D function. Firstly, a cumulative distribution function (CDF) is constructed. Given a discrete PDF, the value of the corresponding CDF for any element i is given by the Equation 2.4.

$$CDF_i = \sum_{j=0}^{i} PDF_j \tag{2.4}$$

The construction of a CDF can be performed in linear time using an iterative formulation, as shown in Equation 2.5.

$$CDF_i = CDF_{i-1} + PDF_i \tag{2.5}$$

Then, a serious of random numbers is drawn with uniform probability. Each of these numbers is projected via the CDF. The projection is often implemented as a binary search, so the projection of a single sample takes logarithmic time (note that no extra steps need to be taken to sort the CDF, because the ascending order of elements is ensured by the definition of the CDF). Equation 2.6 illustrates the projection, where y is a random number drawn with uniform distribution and x is the position of the sample.

$$x = CDF^{-1}(y) \tag{2.6}$$

Note that because the CDF function has a lower first derivative for elements that correspond to the PDF elements of lower values, the samples x are drawn with a probability distribution proportional to the PDF. The procedure is illustrated in Figure 2.15.

Sampling of an environment map is in principle sampling of a 2D discrete function. The standard inversion procedure described above can be used for importance sampling of environment maps. A 2D PDF is constructed from the brightness of the pixels of the environment map. Supposing the environment map is in a latitude-longitude format, the intensity of each pixel needs to be multiplied by $sin(\theta)$ to account for smaller angular extent near the poles, where θ is the altitude angle. In practice, quasi-random number generators with uniform distribution, such as Halton, are often used. The 2D random vectors are then mapped via the CDFs as follows. Firstly, an altitude angle θ is selected. Then, azimuth angle ϕ is selected for the particular scanline corresponding to the altitude angle selected in the previous step. This method has been successfully applied for importance sampling of static environment maps by Pharr [43] and Burke [44].

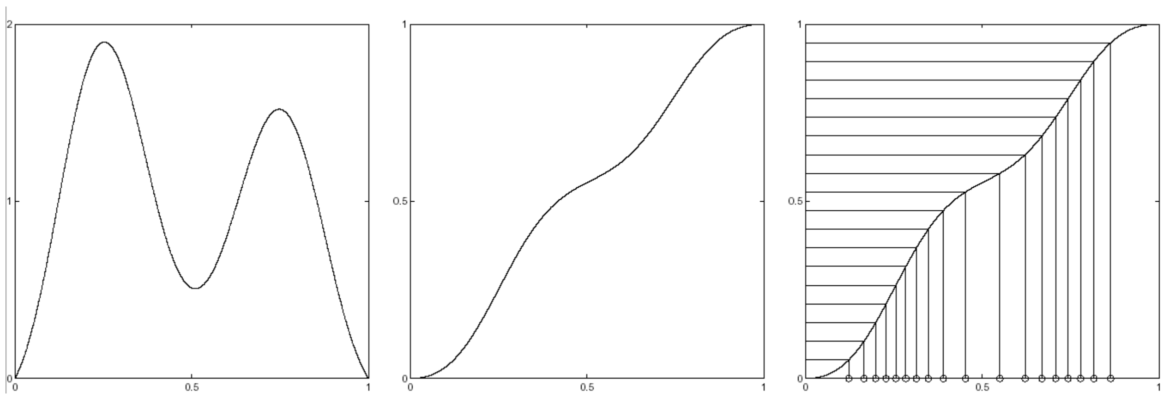


Figure 2.15: Sampling via standard inversion procedure. From left to right: 1D PDF, corresponding CDF, transformed samples (courtesy of Secord et al. [48]).

Havran et al. [16] proposed an extension of this method to improve the temporal coherence for dynamic environment mapping. They uses an inverse transform method proposed by Havran et al. [42]. It exhibits better continuity and uniformity properties than the standard inversion procedure describe earlier in this section. Most importantly, it removes the discontinuity for $\phi = 0$. During the sampling of a dynamic environment sequence, a history of the sampling patterns for a past few frames is kept. After mapping of the samples drawn with uniform probability via the CDFs, two low pass filter are applied.

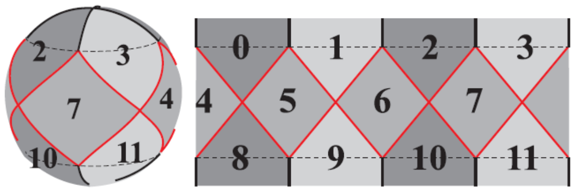
The first low pass filter operates on the total energy of the light sources. Some types of light sources, such as fluorescent tubes, emit light of intensity proportional to the phase of the power supply. In the European power supply network, the voltage changes with a frequency of 50 Hz. Although the frequency is high enough to be ignored by the human visual system, it might cause changes in the total brightness of an image captured by a camera from frame to frame, if the shutter exposure time is in order of milliseconds or less. This source of flickering is prevented by the first filter.

The second low pass filter operates on the trajectories of the samples. Local changes in the illumination result in global changes of the CDFs. Thus, the samples are mapped into different positions than in the previous frame, even though they are placed in regions where the illumination is stable. In order to prevent flickering artifacts caused by this property of the CDFs, the second filter suppresses high frequency movements of the samples.

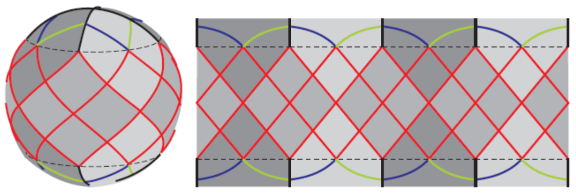
2.5.2.5 Spherical Q2-tree for sampling dynamic environment sequences

In this section, we are going to explore in more detail the method proposed by Wong et al. [15] and briefed in the previous section (2.5.2.2). The method is based on mapping a sphere to rectangles and further subdividing these rectangles.

Firstly, the sphere is subdivided into 12 quadrilaterals (or quads for short) using the HEALPix mapping [45, 46] (depicted in Figure 2.16). Each of these quads has equal surface area, and thus equal solid angle. Each of the quads is then adaptively subdivided to four quads, with each sub-quad maintaining the equal solid angle property - each sub-quad has equal solid angle to all other quads at the same level in the hierarchy and four times less than quads one level up. By repeating the subdivision step until a termination criteria is met, a tree is constructed. Strictly speaking, it is a forest of 12 quad trees.



(a) HEALPix, 12 quads (level 0).



(b) HEALPix, 48 quads (level 1).

Figure 2.16: The HEALPix projection. Sphere (depicted on the left) unrolled into a latitude-longitude map (depicted on the right). The number of quads multiplies by four at every level of subdivision (courtesy of Tien-Tsin Wong [15]).

The algorithm for Q2-tree construction, given in pseudo-code in algorithm 4, maintains a sorted list of interior and leaf nodes. The nodes are sorted in decreasing order according to their importance. The importance metric is described in the next paragraph; basically it is based on the radiance and solid angle of the quad. At each iteration, the left most leaf node (highest importance) is subdivided into four quads. It is then moved to interior nodes and the new for leaf nodes are added to the sorted list. This process stops when required number of nodes is reached. The progress of the algorithm is illustrated in Figure 2.17.

During the sampling process, it is desirable to sample more densely bright regions and less densely dark regions. On the other hand, we want to avoid oversampling small bright regions as they can be well approximated with fewer samples due to their small solid angle. The Q2-tree sampling method uses importance metric proposed by Agarwal et al [11]. It combines both of these requirements, good stratification and higher density in bright areas. The importance is given by the following equation (2.7).

$$p = (L)^a \cdot (\Delta\omega)^b \tag{2.7}$$

L is the total illumination of the region and $\Delta\omega$ is the solid angle. Parameters a and b are non-negative constants that are used to favor illumination (large value of a) or solid

Algorithm 4 Sampling dynamic environment map with a Q2-tree

```
Input:
  EnvironmentMap: Spherical light probe of the environment
  RequiredNumberOfSamples: Integer

Output:
  LightSources: List of directional light sources

Algorithm:
  Q2TREE = HEALPixMappingTo12Quads( EnvironmentMap )
  EvaluateImportance( Q2TREE.Nodes )
  while(Q2TREE.NumberOfNodes < RequiredNumberOfSamples)
  {
    NewNodes = Subdivide( Q2TREE.LeafNodes[0] )
    EvaluateImportance( NewNodes )
    Q2TREE.MoveToInterior( Q2TREE.LeafNodes[0] )
    Q2TREE.AddLeafNodes( NewNodes )
  }
  LightSources = CreateLightSources ( Q2TREE.LeafNodes )
  return LightSources
```

angle (large value of b) component. The result, p in the above equation, is the importance of the region.

In the paper on Q2-tree sampling [15], Wong et al. use constants of $a = 1$ and $b = \frac{1}{4}$. Their importance metric is given by equation 2.8.

$$p = L \cdot \Delta\omega^{\frac{1}{4}} \tag{2.8}$$

A nice property of the equal-solid-angle subdivision proposed by Wong et al. is that both terms of the importance metric can be computed in constant time. The solid angle of quad at level i can be directly computed as

$$\Delta\omega = \frac{\pi}{3 \cdot 4^i}. \tag{2.9}$$

The illumination term is computed using a technique commonly used for texture prefiltering, known as summed area tables [49].

Once we obtain the desirable number of subdivided regions, we can construct a directional light source for each quad. The color and intensity of the light source is given by summed illumination of each corresponding pixel. The light is positioned inside the quad. Wong et al. propose that the light source should be jittered deterministically around the centroid of the region to avoid regularity and maintain determinism at the same time.

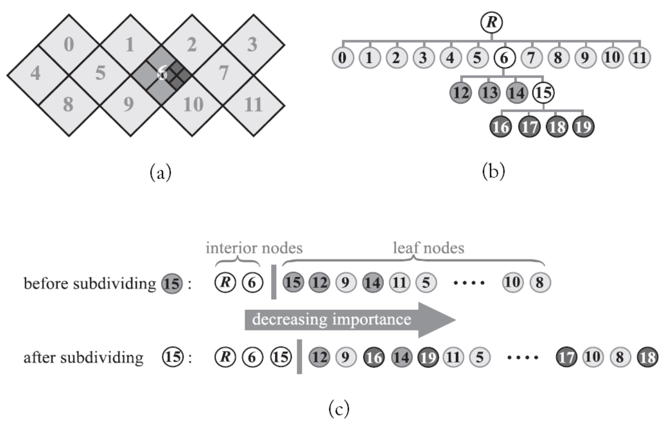


Figure 2.17: Diagram illustrating the progress of Q^2 tree construction. (a) unrolled environment map, (b) Q^2 tree, (c) importance-sorted list (courtesy of Tien-Tsin Wong [15]).

For dynamic sequences of environment maps, the sampling pattern could be computed independently for each frame. But due to small differences between consecutive frames, it is more efficient to keep the Q2-tree from the previous frame and apply some update operations. It also gives us more control over temporal coherency.

The Q2-tree update is based on merge-and-split operations. Firstly, the importance of each region is recomputed. After this operation, there is some inconsistency in the importance-sorted list due to changes in the environment (it is no longer sorted). To adjust the Q2-tree, a series of merge and split operations is performed. A merge operation picks the right most interior node (lowest importance) and combines it and its children into a single leaf node. A split operation picks the left most leaf node (highest importance) and subdivides it into four regions (creating four leaf nodes and converting the node into interior node). Performing one merge operation reduces the number of samples by three. Performing a split operation increases the number of samples by tree. Thus, a combined merge-and-split operation does not change the number of samples.

Merge-and-split operations are performed either until the sorted order of the list is recovered (producing the same Q2-tree as building from scratch would), or until the difference between the left most leaf node and the right most interior node is less than a threshold. Setting this threshold to a value above zero creates a temporal blur, improving temporal coherence of the sampling pattern but reducing the quality of sampling.

Advantages of this algorithm are good temporal coherence and control over the number of samples. The quality of sampling is similar to that of other algorithms [11, 13]. A drawback of this method is that the number of samples needs to be relatively high. The Q2-tree construction starts with uniform subdivision into 12 regions. Using a small number of samples does not allow the algorithm to properly sample bright regions. The results of image-based lighting with dynamic environment map using this algorithm are given in Figure 2.18.

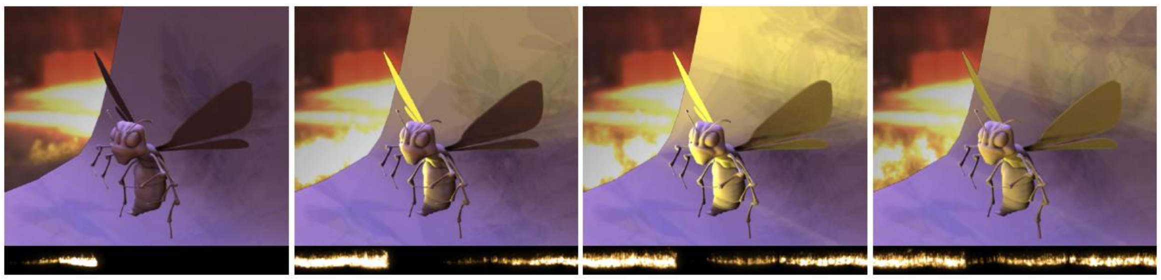


Figure 2.18: Snapshots from a rendering sequence obtained by the Q2-tree sampling algorithm. The shadow underneath the honey bee remains relatively unchanged due to a good temporal coherence of this algorithm (courtesy of Tien-Tsin Wong [15]).

2.6 Illumination methods for augmented reality

In order to blend real and computer generated images seamlessly, it is important to establish common viewing parameters, common visibility and common illumination. In this section, we review several illumination methods for augmented reality.

Firstly, a classification of illumination methods, proposed by Jacobs and Loscos [2], is presented in Section 2.6.1. In Section 2.6.2, we start the discussion of common illumination methods with a pioneer work of Fournier et al. [3], who were the first to introduce differential rendering. In Section 2.6.3, we look at a different approach to common illumination. A method proposed by Jacobs et al. [6] precisely detects real shadows using edge detection and approximate shadow contours estimated from the scene geometry and the light source position. They use shadow mapping or shadow volumes to render virtual shadows, while the regions identified as real shadows are protected from further modification. In Section 2.6.4, we examine the method proposed by Knecht et al. [7]. Their method, known as Differential Instant Radiosity, is based on differential rendering proposed by Fournier et al. [3] and described earlier in Section 2.6.2. The global illumination computation in this method is based on Instant Radiosity introduced by Keller et al. [8] with Imperfect Shadow Maps proposed by Ritschel et al. [9].

2.6.1 Classification of illumination methods

Mixed reality refers to environments consisting of both real and virtual objects, as illustrated in Figure 2.19. To make the environment look realistic, illumination and shadows need to be consistent. Jacobs and Loscos [2] propose a classification of illumination methods for mixed reality. Three different illumination methods are identified: common illumination, relighting and inverse illumination. The methods can also be classified based on the amount of geometry and radiance information that needs to be known about the real environment. Generally, the more input information is available, the more realistic results can be obtain. On the other hand, the need to gather this extra information about the real-scene decreases the usability of the particular method.

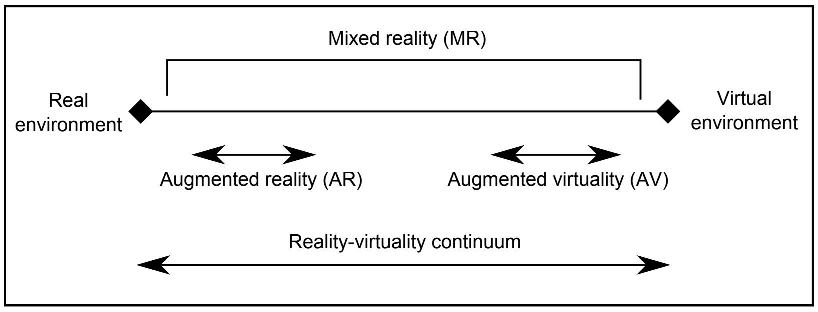


Figure 2.19: Reality-virtuality continuum.

Common illumination refers to those methods providing certain level of illumination blending [2]. This includes shadows projected from virtual objects onto real objects and vice versa, color bleeding, caustics and more. These techniques are limited to preserving the illumination of the real scene; no modifications such as adding or removing light sources or changing light source intensity are allowed. For global illumination computation, it is often important to know approximate BRDF of the materials of real objects. Precision of the BRDF estimate is reflected in the realism of the resulting images. An example of global common illumination is given in Figure 2.20. Techniques used for generation of these images are described in Sato et al. [50].

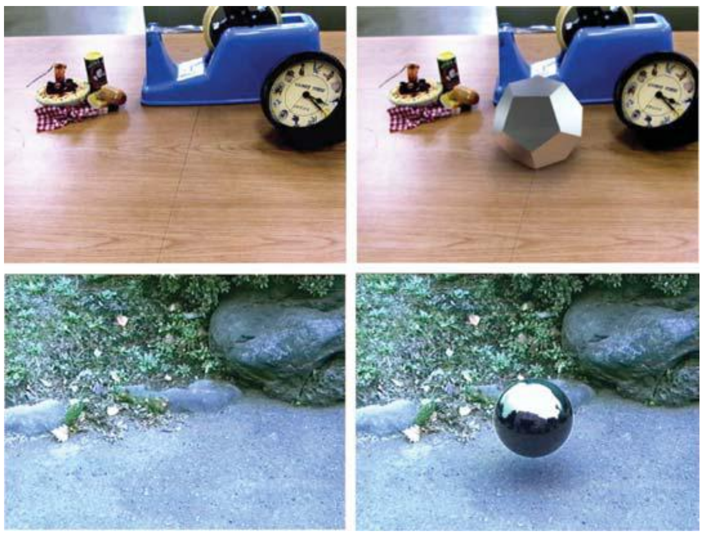


Figure 2.20: Results of global common illumination computation. Left: input images. Right: images augmented with virtual objects (courtesy of Sato et al. [50]).

Relighting techniques allow modification of the lighting pattern of the environment [2]. Usually, this is done in two steps. Firstly, the illumination of the real scene is analysed. Secondly, new illumination is computed. Changes are introduced into the illumination pattern (addition/removal of light sources, intensity modification, etc.) and the scene is rendered with the new illumination. In order to obtain these changes in illumination pattern while keeping the realism of the real scene, it is important to have a geometric model of the real scene. Also, an estimate of the BRDF of the real objects in the scene is required, although the exact knowledge is not necessary if we are only concerned with producing a result that looks realistic to human visual perception rather than being physically accurate. The results of relighting computation are illustrated in Figure 2.21.

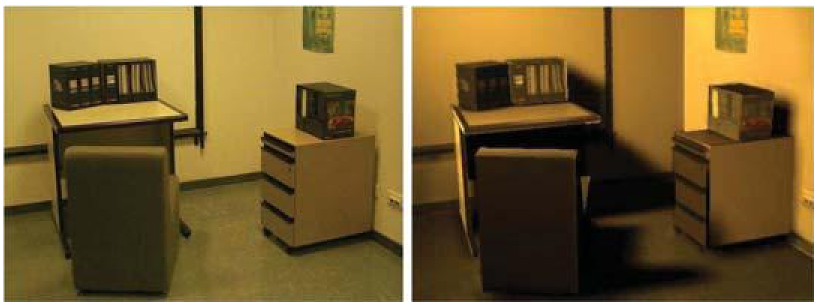


Figure 2.21: Results of relighting. Left: real-scene. Right: synthesised image after light removal and insertion of a new light source. Global common illumination computed at interactive frame-rates (courtesy of Loscos et al. [51]).

Inverse illumination, also known as physically-based illumination, is a set of methods attempting to compute the BRDF functions of all materials in the scene, as well as positions and intensities of light sources [2]. Once computed, this information can be used for both common illumination and relighting. An in-depth overview of inverse illumination techniques can be found in Patow and Pueyo [52].

2.6.2 Common illumination based on differential rendering

Fournier et al. [3] proposed a common illumination method based on differential rendering. They use a simplified model of the real-scene. It is used for global illumination computation as well as to determine visibility and viewing parameters. Only diffuse surfaces are considered. Thus, this algorithm is not capable of computing phenomena such as specular highlights, caustics, etc. Also, if this phenomena are present in the input image, the precision of the estimated radiosity decreases.

The estimation of total power of light sources present in the scene is based on the radiosity equation (see equation 2.10).

$$B_i = B_{e,i} + \rho_i \sum_{i=1}^{N} B_j F_{ij} \tag{2.10}$$

where B_i refers to the radiosity of surface i, E_i is the emissivity of surface i, ρ_i is the reflectivity of surface i, and F_{ij} is the form factor of surface j relative to surface i.

The relationship between surface elements and pixels is determined using ray-tracing. Once the mapping is complete, the radiosity of each surface element is approximated with the average intensity of the pixels corresponding to it. The reflexivity is estimated using a heuristic based on the ratio of radiosity of the element being estimated to the average radiosity of neighbouring pixels. The idea behind this heuristic is that if an object in a dark part of the scene appears bright, it is likely to be a result of its relatively high reflexivity. Obviously, there are many other factors affecting the brightness of an object and this heuristic might not always give good results, but in most cases, it gives a reasonable estimate.

It is assumed that the position and intensity of light sources is known. After modelling of the light sources, we can compute the global radiosity. The solution gives us an estimate of radiosity value for each element in real-scene.

A differential rendering method is used to compute corrections for shadows and inter-reflections coming from computer generated objects. A global illumination computation is performed to get approximate radiosity for the real-scene and the augmented scene. The ratio between these two radiosity values (with and without CG objects) represents the attenuation or gain that needs to be applied to the real image. If, for example, the pixel is darker after insertion of CG objects, it means that a shadow is cast on the corresponding region of the real-scene and its value needs to be decreased. This way, the impact of inaccurate estimates made in previous steps on the output image quality is minimized, as long as the error is systematic in both GI computations.

The global illumination is computed using a method known as progressive radiosity (see Section 2.3.3) and the visibility is tested by ray-casting (Section 2.2). Re-rendering of the scene is done by the ray-casting algorithm. If the pixel corresponds to a computer generated object, then pixel intensity is determined based on radiosity of the surface and material properties (color, texture, etc.). Otherwise, the intensity of a corresponding pixel from the real image is used, multiplied by the attenuation coefficient computed in the previous step (ratio between estimated radiosity with and without CG objects).

2.6.3 Shadow detection and generation

Jacobs et al. [6] proposed a real-time solution to common illumination based on shadow detection and shadow generation. Their method works in three steps. Firstly, real shadows in the real scene are detected. Then, a shadow protection mask is generated based on the shadow regions identified in the previous step. Finally, shadows cast by virtual objects are generated. Only the regions of real scene not protected by the shadow protection mask computed in the previous step are modified to maintain consistency. The algorithm is depicted in Figure 2.22 and explained in detail in the next paragraphs.

Although the results are promising, the method works well only on limited type of scenes. The method can handle only a single light source. Also, it is restricted to hard shadows (although pseudo-soft shadows are approximated reasonably well for scenes with a small bright light source). Approximate geometric model of the real scene and the position of the light source need to be known.

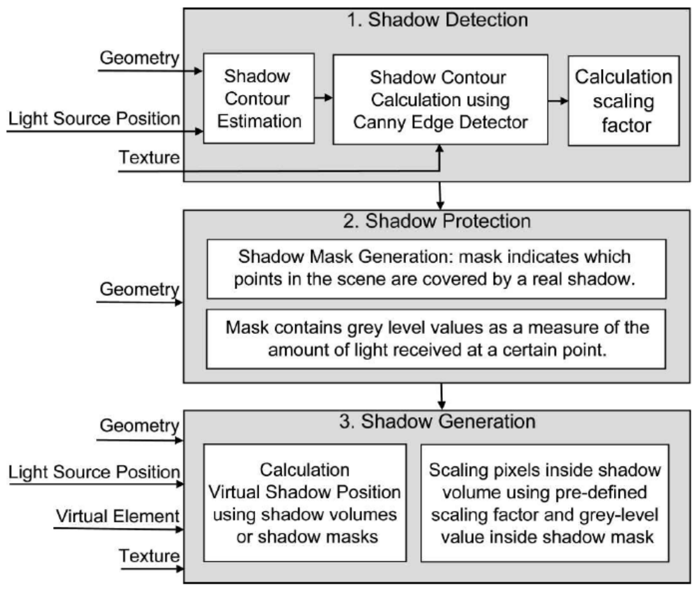

1. Shadow Detection
Geometry
Light Source Position
Texture
Shadow Contour Estimation
Shadow Contour Calculation using Canny Edge Detector
Calculation scaling factor
2. Shadow Protection
Geometry
Shadow Mask Generation: mask indicates which points in the scene are covered by a real shadow.
Mask contains grey level values as a measure of the amount of light received at a certain point.
3. Shadow Generation
Geometry
Light Source Position
Virtual Element
Texture
Calculation Virtual Shadow Position using shadow volumes or shadow masks
Scaling pixels inside shadow volume using pre-defined scaling factor and grey-level value inside shadow mask

Figure 2.22: A schematic overview of the three steps involved in the generation of consistent shadows as proposed by Jacobs et al. (courtesy of Jacobs et al. [6]).

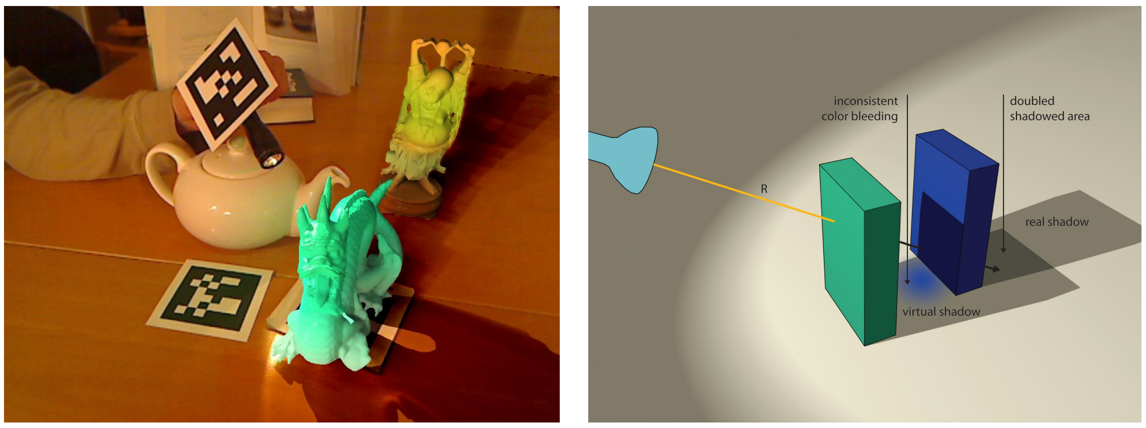
Shadow detection. The shadow detection steps aims at identification of the real shadows in the scene and computation of the scaling factor for each material. Firstly, the shadow contour is estimated from the scene geometry and light position. Because we only have approximations of the scene geometry and light position, the result does not precisely conform to the real shadows. The estimate of shadow contours is then used together with the texture of the real scene to compute the exact contour of the shadows. The exact shadow contour is extracted using an edge detector (Jacobs et al. use the Canny edge detector [53] in their paper [6]). The scaling factor is defined as a ratio of colour between shadow regions and non shadow regions. An average pixel value is computed for both regions for each material separately to get the scaling factors.

Shadow protection. In the shadow protection step, a shadow mask is constructed from the shadow regions identified in the previous step. To ensure consistency between real and virtual shadow, it is important to align the shadow mask with the real shadows as precisely as possible. A gray scale shadow mask with gradients at the edges instead of binary shadow mask can be used to account for soft shadows. The shadow mask is used in the next step to prevent modification of the regions that are already in shadow.

Shadow generation. In the shadow generation step, virtual shadows are computed. A real-time method such as shadow volumes or shadow maps is used. Only the regions of virtual shadows that do not overlap with real shadows are modified. The modified pixel intensity is obtained by multiplying the pixel value by the scaling factor computed in the first steps.

2.6.4 Differential instant radiosity

Knecht et al. [7] proposed a global illumination rendering system for mixed reality based on instant radiosity [8] and differential rendering [3]. Their method calculates common illumination between real and virtual objects at real-time frame rates.



(a) (b)

Figure 2.23: Figure (a): a mixed reality scenario rendered by the Differential Instant Radiosity method proposed by Knecht et al. Figure (b): illustration of limitations of the proposed method (courtesy of Knecht et al. [7]).

They proposed several improvements of the techniques. Firstly, they proposed a modified Instant Radiosity approach that can be used to perform Differential Rendering in a single pass. As opposed to two-pass differential rendering, this improves the efficiency as well as reduces the error caused by inconsistent sampling in the two passes. Furthermore, they use a new approach to imperfect shadow maps, in which the splats are aligned to the surface normal. They also proposed a novel method to assign virtual point lights (VPLs) to multiple primary light sources. Finally, the reduced temporal flickering artifacts by exploiting temporal coherence.

There are some limitations and imprecisions resulting from the way indirect illumination is handled. Most noticeable is the effect of double shadowing. This is because the reflective shadow map (RSM) generated from the viewpoint of a light provides information only about the front most objects. Another limitation of their approach is the inability to cancel out color bleeding artifacts in virtual shadows. These problems are illustrated in Figure 2.23.

Algorithm outline

It is assumed that a geometric model of the real scene is known and correctly registered to the virtual scene. BRDFs of the real materials are estimated. Since the estimate is usually not very accurate, direct rendering produces strong visual artifacts. As explained in Section

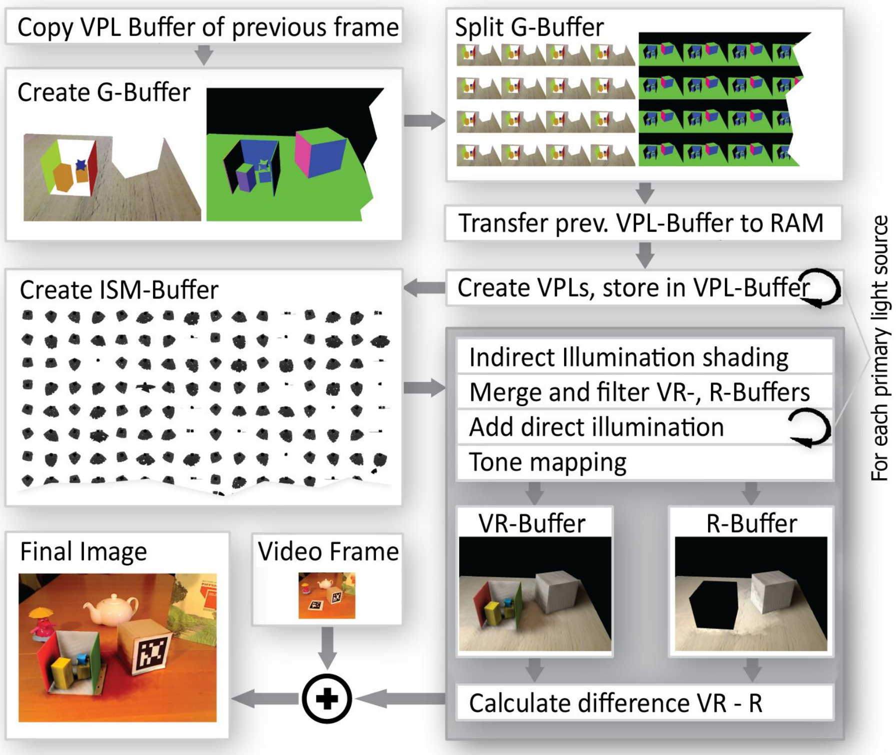

Copy VPL Buffer of previous frame
Create G-Buffer
Split G-Buffer
Transfer prev. VPL-Buffer to RAM
Create VPLs, store in VPL-Buffer
Create ISM-Buffer
Indirect Illumination shading
Merge and filter VR-, R-Buffers
Add direct illumination
Tone mapping
For each primary light source
VR-Buffer
R-Buffer
Final Image
Video Frame
Calculate difference VR - R

Figure 2.24: Algorithm outline of the rendering system proposed by Knecht et al. (courtesy of Knecht et al. [7]).

2.6.2, these artifacts can be minimized with Differential Rendering [3]. The idea is to render the scene twice; in one pass, both real and virtual objects are considered, while in the other pass, only real objects are considered. Only the difference is added to the original value of each pixel.

Knecht et al. modified this method to work in a single pass. The same paths are generated for both rendering targets, but the pixel value is added only to the appropriate buffer. This is achieved by associating a flag to each element (light source, point to be shaded, ...) to distinguish between real and virtual path. If a path contains any virtual elements, it only contributes to the combined (real and virtual) buffer but not the real buffer.

To approximate global illumination in real-time, Knecht et al. use Instant Radiosity. For each primary light source, a number of virtual point lights (VPLs) is created. The scene is rendered from the viewpoint of the light source to create a reflective shadow map (RSM) and the RSM is then sampled using importance sampling.

An environment map (in the system proposed by Knecht et al. captured with a fish-eye lens camera) can be transformed into a set of light sources using importance sampling the same way as if sampling the RSMs.

The Instant Radiosity method they use is based on Imperfect Shadow Maps (ISM) proposed by Ritschel et al. [9]. A low resolution shadow map is created for each virtual point light, using a subsampled version of the scene, represented as a point cloud.

Chapter 3

Analysis and design

In this chapter, we analyse the requirements on our framework and propose a solution. This chapter deals quite abstractly with our design goals and decisions, while in Chapter 4, we present our implementation in more details.

3.1 Problem statement and analysis of solution

In this section, we state the desired features of our framework. In the rest of this chapter, we will discuss possible solutions and propose a framework that suits our needs.

We plan to use the proposed framework in an augmented reality platform, known as *Zora* [54]. It is a portable platform similar to the *Spinnstube* [20]. It uses a 3D Ready TV mounted above a half-silvered mirror. The half-silvered mirror serves as an optical combiner. By attaching the platform to a table the user creates a workspace where ideally, real and virtual objects are merged seamlessly.

It is known that establishing common illumination between the real and virtual objects greatly improves the level of realism. In order to record and reproduce the illumination of the real environment, we need to capture a high dynamic range light probe of the upper hemisphere. Capturing as well as processing of the light probe should happen at real-time framerates. To use the recorded illumination in the renderer, we need to implement an importance sampling algorithm with strong temporal coherence to avoid frame to frame flickering.

We decided that it would be more cost efficient and flexible to use a universal device that is already available on the market than developing our own platform to capture and process the light probes. We used the Nokia N900 smartphone with a linux-based operating system. The device has a built-in camera of sufficient resolution, framerate and quality to capture the light probe. It can be programmed to capture a burst of varying exposure images that are fused into a single high dynamic range image [55]. The device provides wireless connectivity such as WLAN IEEE 802.11, Bluetooth and GPRS. It makes a suitable light probe acquisition platform as it can be easily placed anywhere in the scene to capture the light probe of the environment and send the results to the computer wirelessly. The device comes with sufficient computational power to perform the importance sampling on-chip and distribute over network the positions of the light sources, which significantly reduces the network bandwidth. More details on the particular device we used can be found in Section 4.1.

For efficient illumination of the scene using the light probe of the environment, it is necessary to decompose the light probe into a set of directional light sources. Several criteria should be considered when choosing a proper sampling algorithm. These include the quality of sampling pattern, temporal coherence and performance. Section 2.5.2 gives a discussion of existing importance sampling algorithm.

The following section gives an overview of our framework design and the technologies and algorithms to be used.

3.2 Framework design

In this section, we state our design goals. We then propose a framework that makes it easy to integrate our algorithm into existing projects. We also discuss the technologies and algorithms to be used in our implementation.

3.2.1 Concepts

The design goals of our project were to develop a framework that is easy to integrate into existing platforms and that can be easily maintained and extended. We used a client-server architecture to separate the image capture and processing logic from the user code. Both the server side and client side use a layered architecture. We designed a communication protocol that is compliant with standard norms and protocols. Apart from that, we attempted developing a communication protocol that provides both fast machine readability as well as human readability. The next sections present and explain our design choices.

3.2.2 Software architecture

We designed a client-server architecture where all processing, including image capture and importance sampling, is done on a server. In our testing scenario this is a portable device equipped with wireless connectivity. Results of the processing are then sent on request to clients via standard protocols. The server also provides interface for the clients to change the settings of the server.

The design of the server itself is based on the layered architecture. A network daemon provides interface for the clients to connect to and to send requests to and receive replies.

Change requests are passed to the processing core via a message queue. Results of processing are passed to the server via shared memory using double buffering, mutexes and signals.

The design of the client framework is in principle similar to the server. A network client encapsulates all the communication with the server. This client provides a signal that is emitted every time new data are received from the server. A double buffering mechanism with mutexes then provides the means to transfer the data into a processing thread. The user program should hook to this signal and read the updated data when the signal is emitted.

3.2.3 Technologies

Our communication protocol complies with the *Hypertext Transfer Protocol* (HTTP). Replies are encoded in *Hypertext Markup Language* (HTML), *Extensible Markup Language* (XML), or plain text, depending on the nature of the data. This compatibility with standard technologies gives a great flexibility to the framework and assures interoperability with existing applications. For example, any web browser can be used to configure the server because it acts as a standard web server. In fact, any request to the server can be sent via a web browser and the reply can be displayed in a web browser. Apart from fulfilling our main design goals, this concept also proved to be useful during the implementation, as it provides means of debugging the framework using existing third party client programs.

Our application is based on the Qt framework [56]. It solves the portability issues as the same code can be compiled for both embedded systems as well as desktop computers. Another advantage is that there is a good support for developing in Qt on our targeted device that runs the server side, the Nokia N900 smartphone, as both the device patent and the framework are owned by the same company [56].

The core parts of our application, on both client and server side are written in C++. Demo applications, such as the renderer, use OpenGL graphics library and the GLSL shading language.

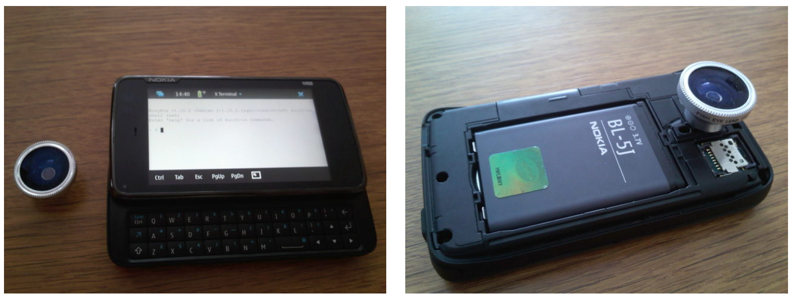
Chapter 4

Implementation

In this chapter, we present our implementation. Firstly, we briefly describe the hardware that we worked with and discuss the calibration of a camera. Then, we describe the architecture of our framework. Finally, we give an overview of the importance sampling algorithms that we have implemented.

4.1 Hardware description

In this section, we briefly describe the hardware that we use to capture and process the light probes.



(a) Nokia N900; front view. (b) Nokia N900; back view.

Figure 4.1: Photographs of the Nokia N900 smartphone and the fish-eye lens.

The server program runs on Nokia N900 smartphone equipped with a fisheye lens camera. Its operating system is Maemo 5, a Linux-based OS. It has a Texas Instruments OMAP3 microprocessor with the ARM Cortex-A8 core, clock rate 600 MHz. The processor features a 128-bit single instruction multiple data (SIMD) instruction set. The phone has a PowerVR SGX 530 GPU, that supports OpenGL ES 2.0. It provides a wide range of connectivity, such as WLAN IEEE 802.11 b/g, Bluetooth, and GPRS [39].

The phone has a built-in camera of 5 MPx resolution (2584×1938). We attached a fisheye lens to the camera that provides a field of view of about 160 degrees horizontally and 140 degrees vertically. A photograph of the device is shown in Figure 4.1.

4.2 Camera calibration

In order to provide mapping of the captured image into polar coordinate system properly, it is necessary to know a geometric model of the camera. This model can be computed by geometric calibration, described in Section 4.2.1. In some applications, it is also important to know the intensity of light sources in physical units. Photometric calibration is explained in Section 4.2.2.

4.2.1 Geometric camera calibration

A common method of geometric camera calibration uses a set of images of a checkerboard, taken from different perspectives [57]. Cameras having a field of view less than 180 degree can be modeled by an intrinsics matrix and several radial distortion coefficients. We used two distortion coefficients (k_0, k_1) and an intrinsics matrix containing the principal point (u_0, v_0), skew (γ) and focal length in pixels (α, β). Equation 4.1 projects 3D points (x, y, z) to 2D camera space (u_{final}, v_{final}). *Extrinsics* is a 3×4 matrix that defines the perspective of the camera.

$$\begin{aligned}
\begin{bmatrix} u & v & w \end{bmatrix}^T &= \begin{bmatrix} \alpha & \gamma & u_0 \\ & \beta & v_0 \\ & & 1 \end{bmatrix} \times \begin{bmatrix} extrinsics \end{bmatrix} \times \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T \\
u' &= \frac{u}{w} \\
v' &= \frac{v}{w} \\
u_{norm} &= \frac{(u' - u_0)}{\alpha} \\
v_{norm} &= \frac{(v' - v_0)}{\beta} \\
d_1 &= u_{norm}^2 + v_{norm}^2 \\
d_2 &= d_1^2 \\
u_{final} &= u' + (u' - u_0)(k_0 \cdot d_1 + k_1 \cdot d_2) \\
v_{final} &= v' + (v' - v_0)(k_0 \cdot d_1 + k_1 \cdot d_2)
\end{aligned} \tag{4.1}$$

The intrinsics matrix and distortion coefficients are computed from a set of images of a checkerboard (shown in Figure 4.2) by detecting the corners of the black and white squares and finding a correlation among the pairs of images. There are several open source programs to perform the geometric calibration automatically. We experimented with CamChecker [58] and OCamCalib [59]. CamChecker is implemented in C++ language and computes the camera model described above. A problem of this tool that we found is that it fails to detect corners in the border regions of the image, which leads to imprecisions of the calibration. OCamCalib implements a more robust corner detection algorithm in the C language. Other parts of the tool are implemented in Matlab. It is a semi-automatic calibration tool, requiring some user interaction. Another possible way to calibrate the camera is to use the OpenCV library [60], which makes it quite easy to implement an automatic calibration tool in a compiled language from scratch. It is possible to run the calibration tool directly on the camera device.

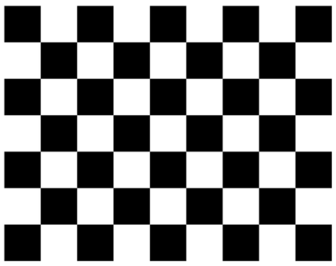
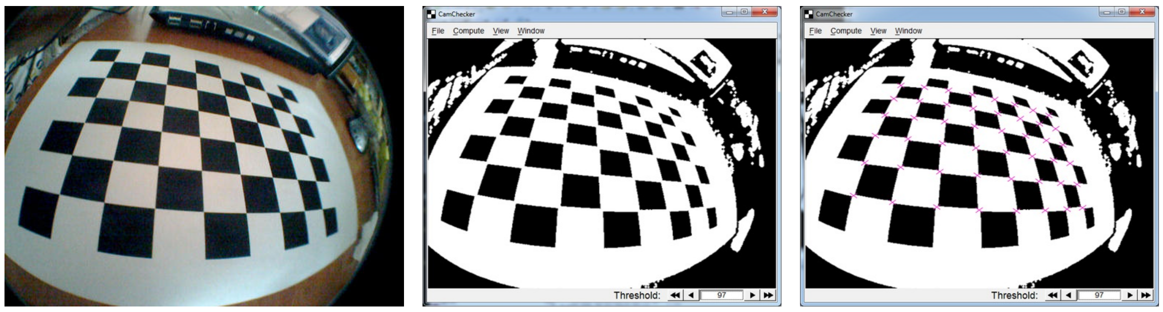


Figure 4.2: A checkerboard pattern that can be used in geometric camera calibration.

We used the CamChecker tool to perform automatic camera calibration. Due to the corner detection problem described in the previous paragraph, the camera model it produced has a significant imprecision in the outer regions of the image, up to 10 degrees. We plan to integrate the corner detection algorithm of OCamCalib into CamChecker, which should solve this issue. Some of the steps involved in automatic camera calibration are illustrated in Figure 4.3.



(a) Photograph of a checkerboard (b) Thresholded photograph (c) Detected corners

Figure 4.3: Geometric camera calibration using a checkerboard. Multiple views are required to compute the camera model. In this particular case, all corners have been successfully detected.

4.2.2 Photometric camera calibration

We experimented with two photometric calibration techniques: direct and indirect. In the direct method, the illuminance of a surface is first measured by an illuminance meter. Then, the illuminance meter is replaced by the camera to be calibrated, positioned in a way that the amount of light entering the camera lens is the same as the amount of light measured by the illuminance meter. The lighting conditions of the environment are kept unchanged. If the camera uses a physically based HDR image representation, independent of the exposure and gain, then the ratio of the measured illuminance to the sum of luminances of all pixels should be the same for every environment. Multiplying pixel intensities by this ratio gives pixel luminances in physical units (lux), as measured by the illuminance meter.

A problem of this method is when the solid angle of the camera differs from that of the illuminance meter, as in our case. The visible angle of the fisheye lens we use is about 160 degrees in horizontal direction and 140 degrees in vertical. Because the illuminance meter records light coming from the entire hemisphere, the measured ratio is imprecise. The error can be decreased if we perform the calibration in a controlled environment, minimizing the amount of light coming from the directions that are not visible by the camera.

An indirect approach solves this issue. A photograph is taken of a surface of known reflectance. The illumination of a small area on this surface is measured by an illuminance meter. Knowing the area of a pixel projected onto this surface, and the illuminance of the surface, one can compute the outgoing luminous flux. The area covered by one pixel can be computed from the solid angle of the pixel and the distance between the camera and the surface. The solid angle of a particular pixel can be computed from geometric calibration, or estimated from the total solid angle of the lens and camera resolution. Knowing the luminous flux of a single pixel and the value of that pixel in a physically based HDR image representation, the ratio of pixel values to the physical energy can be computed.

A disadvantage of the indirect method is that the noise of the camera sensor leads to significant errors in the computation of a single pixel value. This error can be minimized by averaging the values of multiple pixels, weighted by a gauss curve.

In our implementation, we use the results obtained by the direct calibration method. Due to camera sensor noise, imprecise geometric calibration and estimation of the surface reflectance, values obtained for various scenarios differed more significantly for the indirect method than for the direct method. The results of our photometric calibration of the Nokia N900 smartphones's camera are discussed in Section 5.2.2.

4.3 Software architecture

In this section, we describe the architecture of the implemented system. Firstly, in Section 4.3.1, we discuss the general concept of the proposed framework. Then, in Section 4.3.2, we go into more detail about the architecture of the server. In Section 4.3.3, we discuss the architecture of the clients and how it can be integrated into existing systems. Finally, in Section 4.3.4, we briefly overview the communication protocol.

4.3.1 Concept

As mentioned in the previous sections, the light probes are being captured on a dedicated device that is capable of wireless communication. Because the processing power of the device is sufficient to perform the importance sampling in real-time, we decided to process the light probes on the device. This saves a significant amount of bandwidth. Instead of sending a full light probe image, we send only a list of light sources. It also conserves computational resources of client devices.

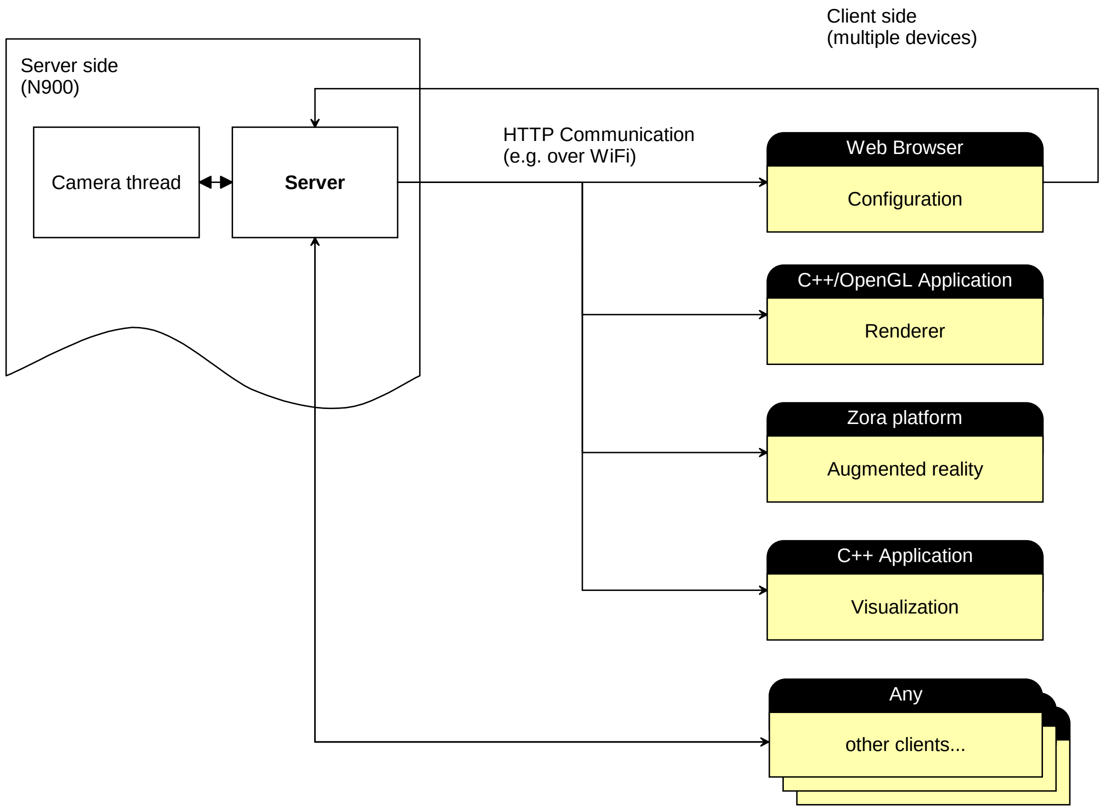


Figure 4.4: Conceptual diagram of our framework. The device running the sampling algorithm communicates with multiple clients over network.

Diagram 4.4 illustrates this concept. The server provides interface compliant with the HTTP protocol. Multiple clients can connect to the server at a time and request sampling results, or change the configuration of the importance sampling algorithm. The server also provides configuration interface based on HTML, so any web browser can be used to configure the device. The device also runs a processing thread that communicates with the HTTP daemon thread via message passing and shared memory with mutexes, and performs the importance sampling.

4.3.2 Server side

The server runs on a mobile device. We used the Nokia N900 smartphone, as described in Section 4.1. The device runs a linux operating system and supports the Qt framework, including network module and multithreading.

We designed a layered architecture that separates the network interface from the core processing and makes maintenance and future development of the software simpler. The architecture is illustrated in Figure 4.5. The following paragraphs describe our design.

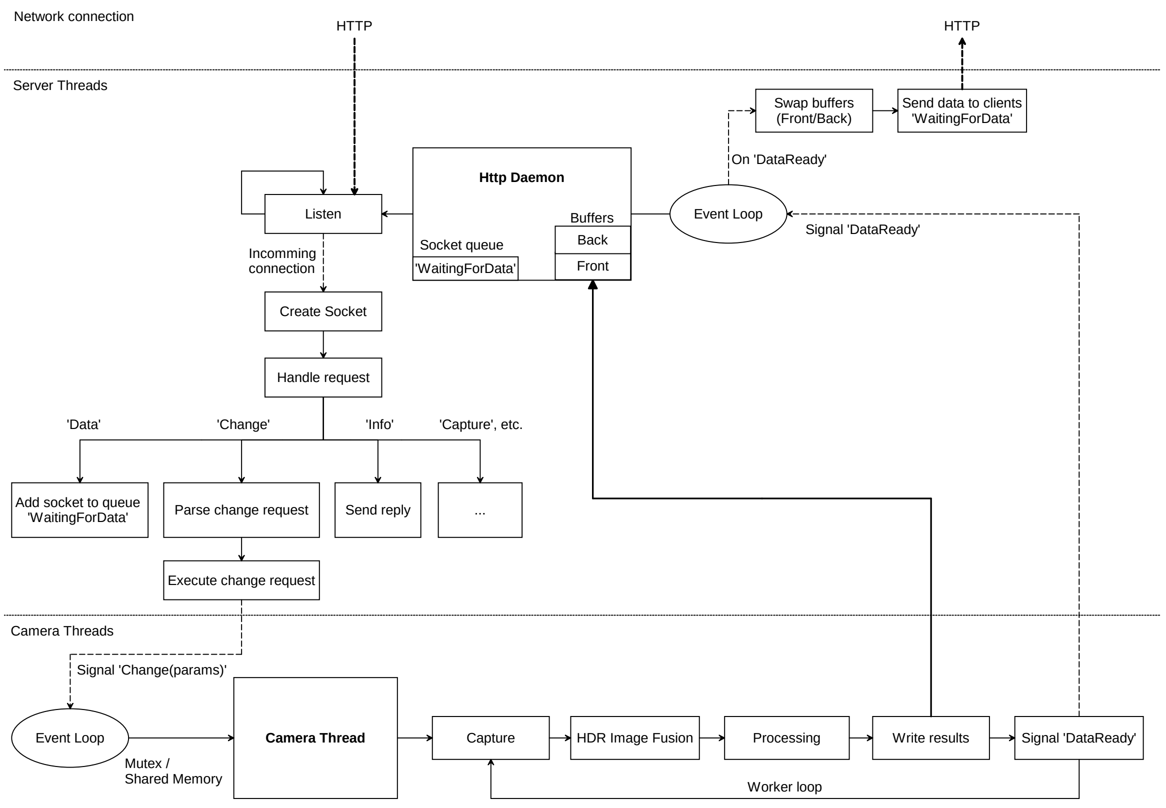


Figure 4.5: A flowchart diagram of the server side. The HTTP Daemon receives requests from the clients. When the processing of a frame is finished, the results are sent to the clients requesting them.

The *HTTP Daemon* listens for new incoming connections and either handles them immediately, passing information to the *Camera Thread* if necessary, or adds the client socket to a queue if the requested resources are not yet available. Change requests, altering the configuration of the importance sampling algorithm, are passed to the *Camera Thread* through an event loop. The *Camera Thread* itself is executing an infinite loop of light probe acquisition and sampling. When a new frame is processed and the results are ready to be sent, the *Camera Thread* moves them into the front buffer of the *HTTP Daemon* and fires a signal *'DataReady'*. When the event loop of the *HTTP Daemon* receives this signal, it moves the data to a back buffer and starts sending them to the clients that are *waiting for data*. In the meantime, the *Camera Thread* continues processing the next frame.

The execution loop of the *Camera Thread* consists of five steps. Firstly, a set of captured images is read from the sensor and the sensor is configured to asynchronously start capturing a new burst of images. The captured burst of images with varying exposures is then fused into a single high dynamic range image. In the processing step, the image is mapped into polar coordinates and a selected importance sampling algorithm is executed, as described in Section 4.4. Finally, results are copied to the front buffer of the *HTTP Daemon* via shared memory protected with a mutex and a *'DataReady'* signal is emitted. Following that, the loop starts again from the beginning to process the next frame.

4.3.3 Client side

We implemented a client module that handles the communication with the server and parses the results. It provides a simple interface that can be used to plug it into existing systems. A signal is emitted every time the client module receives new sampling data from the server. It provides a method to get the processed sampling data (positions and colours of light sources) that should be called from the event handler that is executed on emission of this signal.

4.3.4 Communication protocol

The protocol we designed for the communication between the server and clients is based on HTTP. The client sends a GET requests and receives a reply. The server uses regular expressions to parse the request and performs an appropriate action. A complete list of supported requests and their syntax is provided in Appendix A.1. The clients can, for example, request the sampling data in XML or plain text format, request change of the importance sampling settings, such as the number of samples or the algorithm being used, or request information about current settings.

As the protocol is fully compliant with standard web technologies and protocols, any web browser can be used to send requests and read the replies in a human readable format. The format of results is also designed to be easily processed by machines.

4.4 Implemented importance sampling methods

We have implemented three different importance sampling algorithms. While the first of them targets static environment map sampling, the later two have been specifically designed for dynamic environment sequences.

The simplest of them, which we will refer to as *Pharr* in the rest of this paper, is based on the standard inverse procedure [61]. It was first used for static environment map sampling by Pharr et al. [43] and Burke et al. [44] in 2004.

The second method we have implemented was proposed by Havran et al. [16] in 2005. In principle, it is similar to the method described earlier. The environment map sampling is also based on a probability distribution function (PDF). However, instead of the standard inverse procedure, the inverse transform method proposed by Havran et al. [42] is used. Apart from that, an attempt has been made to improve the temporal coherence of the sampling pattern. This is done by temporal filtering of the light source energy and position. Captured light source power may change from frame to frame due to fluorescent lamps, for example. To suppress these changes, a filter operates on the total luminance of the environment map. Another source of flickering and jumpy shadow changes is caused by the global nature of the PDF-based algorithm. An abrupt change of the illumination in one part of the scene causes a change in the entire sampling pattern. To limit the light source movements, another filter operates on the trajectories of the samples. In the rest of this paper, we will refer to this method as *Hemigon*.

The third method, known as the *Q2-Tree*, comes from a completely different approach. Proposed by Wong et al. [15] in 2005, this method builds an adaptive quad tree over the sphere projected onto the Euclidean plane. It uses the HEALPix projection [45], widely used for all sky mapping by astrophysicists. Details about the HEALPix projection, including implementation in several languages, can be found on the NASA web pages [46]. In the process of quad tree construction, the algorithm maintains a sorted list of quadrilateral in order of their importance. As Wong et al. [15] proposes, we evaluated the importance of quadrilaterals as a product of their angular extend and brightness. These two values are raised to a predefined exponent, which regulates their influence of the resulting importance value. It can be used to alternate the sampling pattern, giving more favor to either the angular extend or brightness of the quadrilaterals. The algorithm is explained in detail in Section 2.5.2.5.

For a detailed discussion of importance sampling algorithms for dynamic sequences, see Section 2.5. In Section 5.1, we compare the implemented methods in terms of the quality of the sampling pattern, temporal coherence, artifacts and performance.

Chapter 5

Results

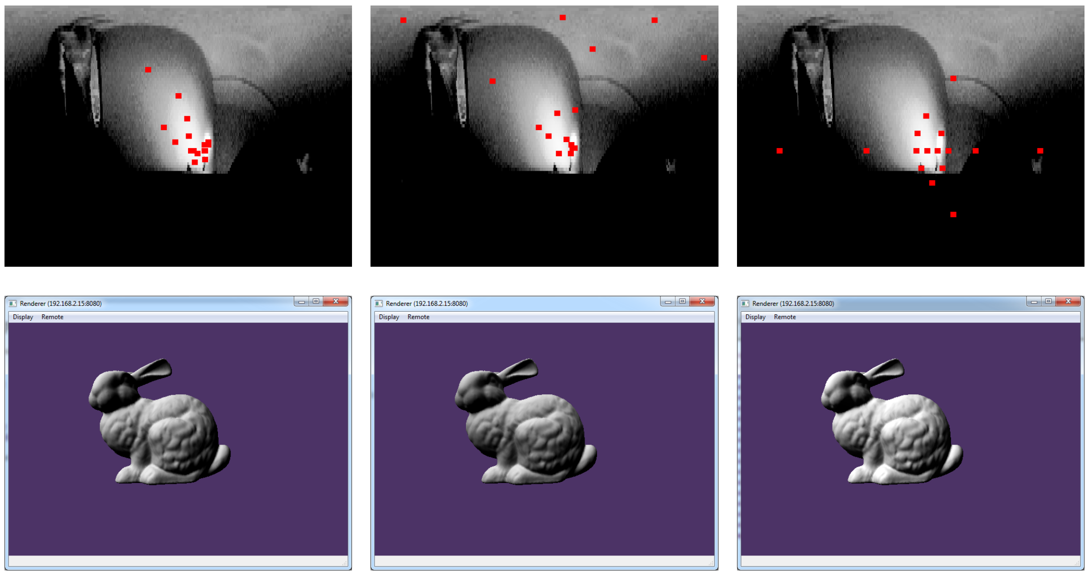
In this chapter, we present the results of our work. Firstly, we compare the importance sampling methods in terms of sampling pattern quality, temporal coherence and performance. We then discuss the correctness and verification of our implementation. Additional information, including screenshots and photographs are given in Appendix C.

5.1 Comparison of implemented methods

in Section 4.4, we described the three importance sampling methods that we have implemented. In this section, we compare these methods based on several criteria. The first criterion is how well the sampling pattern approximates the light distribution in the captured environment. Secondly, we discuss the temporal coherence of the pattern. This includes suppression of any unwanted frame to frame flickering as well as other artifacts produced by abrupt lighting changes. Apart from sampling pattern and its temporal coherence, an important aspect for real-time use of the algorithms is the performance. A comparison of the three implementations in terms of processing time will be given.

5.1.1 Quality of sampling pattern

Both of the two PDF-based sampling algorithms produce equal energy samples. That means that the distribution of the samples is proportional solely to the distribution of the energy in the captured environment. While in general such a sampling pattern makes a good approximation of the environment, in some cases undersampling of relatively dark regions might be a problem. Suppose we have an environment that contains one very bright source of light and several much dimmer light sources. This setting is quite typical for both day outdoor scenes, where the brightest light source is the sun, as well as for night scenes with artificial lighting. A sampling pattern that puts almost all of the samples in the small bright region would be reasonable for most views of the rendered scene. But when the main light source gets obstructed and the view being rendered is in a shadow, then the dim light

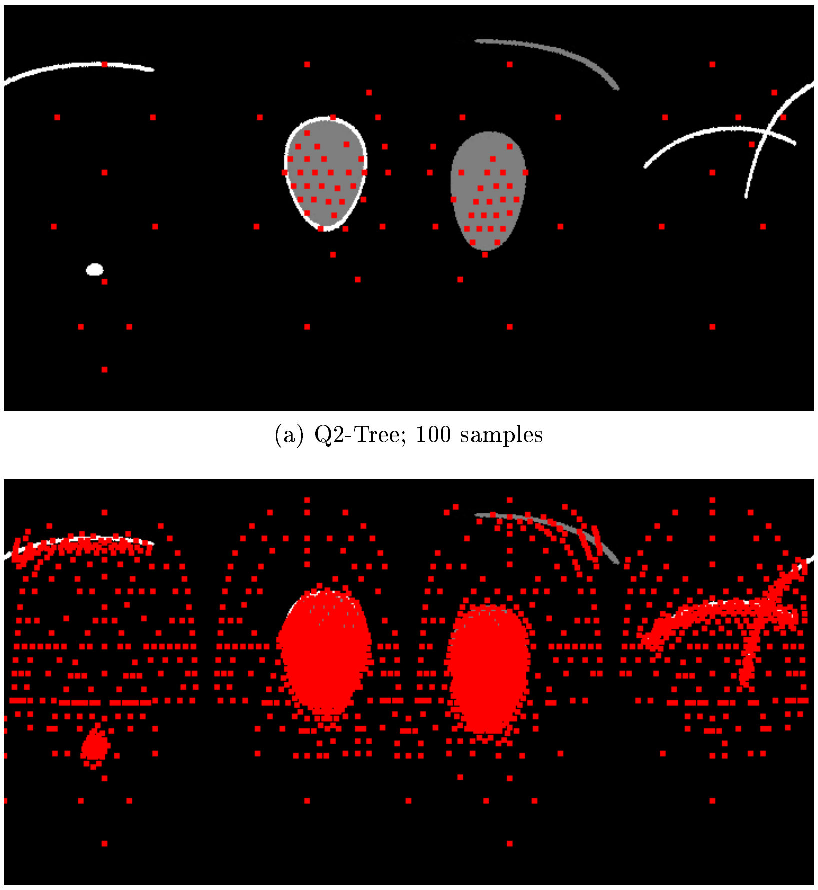


(d) Pharr (e) Hemigon (f) Q2-Tree

Figure 5.1: The upper row shows the sampling patterns produced by the three algorithms for an environment map of resolution 360×90, 16 samples; a render of a bunny lit by each respective set of light sources is given below. From left to right: Pharr, Hemigon, Q2-Tree.

sources come in play. Sampling a small bright region very thoroughly and undersampling the rest produces poor results in such cases. This is where the strength of the Q2-Tree sampling algorithm lies. Spreading the samples according to an importance metric based not only on the brightness, but also influenced by the angular extent produces better results in such situations. Each sample can carry a different amount of energy, so the overall energy distribution represented by the sampling pattern approximates the environment just as well as the sampling patterns produced by PDF-based methods. But the idea is that it is sufficient to approximate the small bright regions with fewer samples than vast areas of the same total brightness.

On the other hand, the Q2-Tree algorithm suffers a limitation to its usability in real-time systems. It requires relatively many samples to approximate an environment well. The first 12 samples is distributed uniformly and a few dozens more is placed in spots that do not match well the light source positions. If we take only a few dozens of samples, the positions of light sources and thus the shadows cast in the renderer do not match the real environment and cause strong artifacts, especially in augmented reality. Furthermore, for subtle movements of light sources, the sampling pattern does not change, only the energy of each sample changes. Suppose a scene with a light bulb swinging on a cord. If the number of samples is not sufficiently high, we obtain the same sampling pattern for each frame. When rendered, the shadows do not move as the light bulb swings, only the intensity changes. Such scenes look very odd and diminish the overall impression of the virtual environment, as the human visual system is very sensitive to shadow positions, providing an important cue about the environment. This problem is overcome if the number of samples is high enough, or if advanced rendering techniques are used to smooth out the boundaries of individual shadows. Unfortunately, both solutions are computationally very costly and with the hardware available at the moment, hard to achieve in real-time.


(a) Q2-Tree; 100 samples

(b) Q2-Tree; 1000 samples

Figure 5.2: An artificial environment map sampled using the Q2-Tree sampling algorithm with (a) 100 and (b) 1000 samples.

Figures 5.3 and 5.6 show the sampling patterns produced by the three algorithms. Note that while Pharr and Hemigon algorithms produced similar sampling pattern, the Q2-Tree algorithm spreads the samples across a broader area because the importance is weighted by angular extent as well as brightness.

Figure 5.1 shows the sampling pattern of each of the three algorithms together with a scene renderer using the respective set of light sources. The burst of varying exposure LDR images that produced the HDR environment map in this figure is given in Appendix C.4 and a photograph of the testing setup in Appendix C.3.

Figure 5.2 shows the sampling pattern of the Q2-Tree algorithm for a varying number of samples. Note that for a relatively low number of samples (i.e. less than 100), the positions of samples differ significantly from the positions of the real light sources. As the number of samples increases, the sampling is more accurate. This is caused by the adaptive nature of the Q2-Tree sampling algorithm. The first 12 samples are placed uniformly and it takes a few more dozens of samples to sample the light sources adequately. In contrast, the PDF-based methods (such as Pharr and Hemigon) place every sample independently of the other samples. Thus, these methods provide better approximation than the Q2-Tree if the number of samples is low.

5.1.2 Temporal coherence

An important aspect of importance sampling algorithm for dynamic sequences is the temporal coherence of the sampling pattern. Frame to frame changes of the light source positions and their intensity cause flickering.

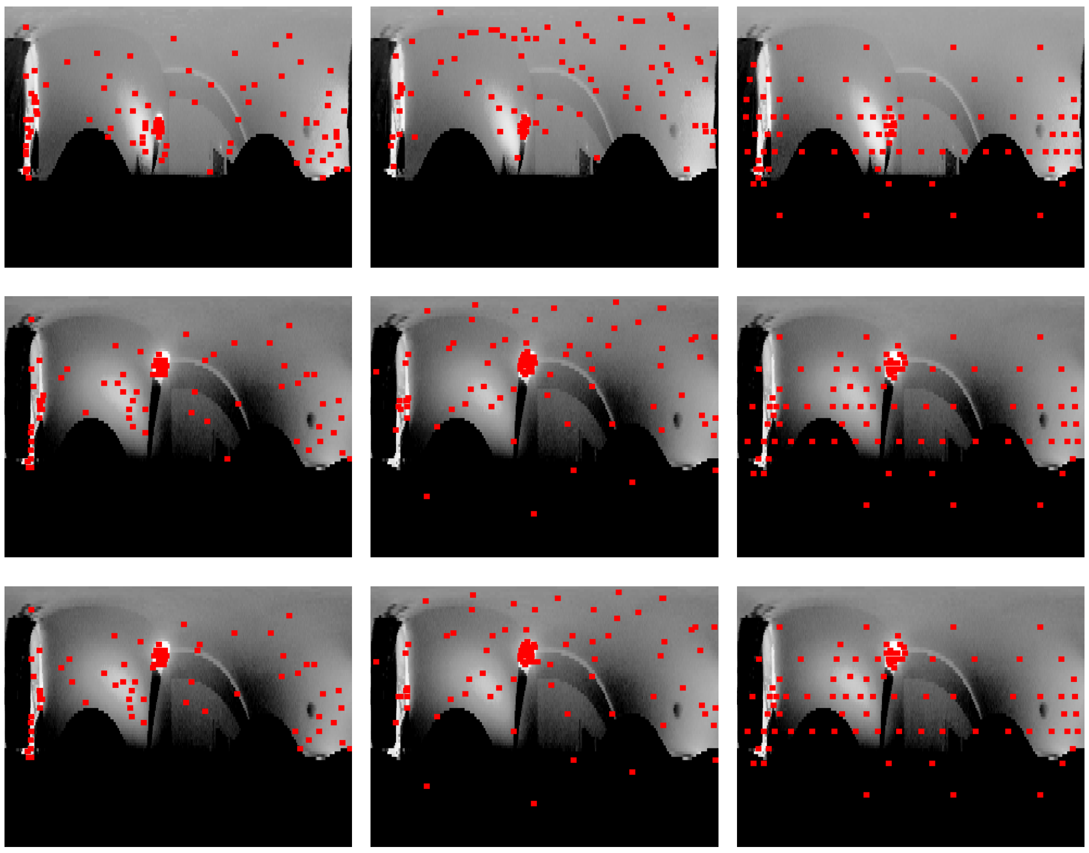


Figure 5.3: Snapshots of the sampling patterns produced by the three implemented sampling algorithms. From left to right: Pharr, Hemigon, Q2-Tree. Vertically: environment maps of varying lighting conditions.

In the implementation of the first algorithm, *Pharr*, we used a Halton generator to generate a sequence of quasi-random numbers in two-dimensional space. These numbers are then mapped via a cumulative probability distribution function (CDF) to obtain the final sample positions. Each sample is assigned the same quasi-random vector for every frame, which means that given two equal input images, the algorithm provides the same results for both. Position of samples changes when the CDF changes. The problem of this algorithm is that it does not handle local changes well. A local change in the input image produces different CDFs and thus the entire sampling pattern changes, even in the regions of the image that stayed unchanged.

The second implemented algorithm, *Hemigon*, attempts to solve these issues by several improvements. Firstly, a different mapping have been used, removing the discontinuity for $\phi = 0$. Secondly, two filters operating on the intensity and trajectories of samples improve the temporal coherence. These processing steps do increase the temporal coherence, but introduce several other problems. The most serious disadvantage of this approach is that abrupt illumination changes are not handled well. As explained in the previous paragraph, switching a light on or off results in a continuous movement of samples from their original position to the new position, even though in the real environment, the change was abrupt. Despite these issues, for a typical scene with subtle frame to frame illumination changes, this method produces best results of the three methods that we have tested.

The *Q2-Tree* sampling method exhibits different temporal coherence issues than the previously discussed two methods. An advantage of this method is the local nature of the quad tree subdivision. Local changes in the illumination do not affect the sampling pattern in the unchanged regions. But using this kind of subdivision introduces some unwanted artifacts as well. Firstly, as described in Section 5.1.1, subtle movements of light sources in the real environment are not reflected by changing the position of samples. Secondly, if the total number of samples is kept constant, local changes of illumination lead to adding or removing samples in other regions. Although the energy distribution represented by the samples is not changed, as the sum of sample energy in a particular region is kept constant (removing samples is implemented by merging four samples into one and summing their energy), abruptly changing the number of light sources in a particular region leads to visual artifacts in the renderer scene.

5.1.3 Performance

Our goal was to implement a system that achieves real-time performance. In this section, we present the performance measurements of our implementation and compare the three importance sampling methods in terms of processing time.

Please note that the results may vary greatly depending on the implementation. Also note that while the implementation of the *Hemigon* algorithm was provided by the author of the algorithm [16], the other two algorithms, *Pharr* and *Q2-Tree*, were implemented by the author of this thesis.

The measurements were performed on a Nokia N900 smartphone, as described in Section

4.1. Three images of varying exposure were fused into one high dynamic range image. The images were captured at resolution 640×480 pixels and mapped to a polar image of resolution one pixel per degree (i.e. 360×90 for a hemisphere). The Q2-Tree algorithm used a HEALPix mapping of resolution $12 \times 100 \times 100$ pixels. The source code was compiled in GCC compiler, version 3.4.4, with the -O3 option enabled.

The asymptotic time complexity is $O(w \times h + n)$, where $w \times h$ is the resolution of the environment map and n is the number of samples. The first term, $O(w \times h)$, accounts for the processing of the environment map prior to placement of samples. This includes, for example, construction of the CDFs in PDF-based sampling algorithms and construction of the summed area table in the Q2-Tree sampling algorithm. The second term, $O(n)$, accounts for the post-processing of samples. This includes computation of the colour of a particular sample. In the Q2-Tree sampling algorithm, the construction and update of a Q2-Tree is also dependent on the number of samples.

The table 5.1 lists the processing times for each algorithm and their overall sampling performance. The first column lists the processing times of importance sampling, assuming a HDR polar image is available. The second column lists processing times including the tasks that are common for all the three algorithms, such as image capture, HDR exposure fusion, mapping to polar coordinates, etc. These tasks took in total 120 ms in this particular case. The measurements were performed with the debugging mode disabled. In debug mode, this stage takes 30 ms more to complete tasks such as tone mapping of the HDR image to be displayed on the Nokia N900 display. All times shown here are in milliseconds. The number of samples was set to 200.

Algorithm	**Sampling time**	**Overall time**
Pharr	10	130
Hemigon	340	460
Q2-Tree	30	150

Table 5.1: Comparison of performance of the three implemented methods. HDR image captured at resolution of 640×480 pixels and mapped into polar coordinates at 360×90. 200 samples. All times are in milliseconds.

Figure 5.4 shows a plot of the execution time versus the number of sample for the three sampling algorithms. Note that for a reasonable number of samples (i.e. up to several thousands) the execution time is almost constant, because the environment map processing dominates the sample processing. As the number of samples increases, the time to process the samples, linear in the number of samples, becomes significant and the overall processing time increases linearly. For the Q2-Tree sampling algorithm, a huge number of samples makes the size of the Q2-Tree become an issue and due to a limited size of the cache, the time complexity even exceeds linear growth. For the Hemigon sampling algorithm, we were provided a set 5000 precomputed quasi-random vectors by the author, so our implementation of this algorithm is limited to 5000 samples. Details, including the data that were used to construct this plot, are given in Appendix C.

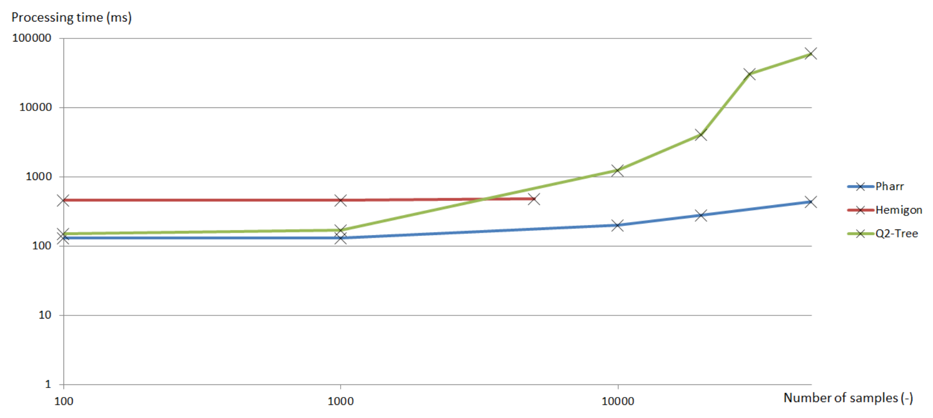


Figure 5.4: Execution time versus number of samples.

5.1.4 Conclusion of the comparison

No single method can be voted to be the best. Each of the tested methods produces better results then the other methods for a particular environment, but has problems to handle situations that are handled successfully by other algorithms. This section should point out the problems of each of the tested methods and find a space for improvements, rather than guide the reader to pick a particular method.

To sum up the discussion of comparing the three importance sampling algorithms, there are some patterns that we have observed. The PDF-based methods produce good results for environment sequences where the frame to frame changes of illumination are subtle. Filtering of sample trajectories and intensities improves the temporal coherence but causes unwanted effects if the changes are abrupt. On the other hand, the sampling method based on spherical Q2-tree handles successfully abrupt changes, especially if the changes are local, but has problems handling minor light source movements.

5.2 Verification and testing

In this section, we describe some of the tests that we performed to ensure correctness of results and to debug potential problems. We also present photographs of our system in use and screenshots illustrating sampling patterns produced by the three implemented algorithms.

5.2.1 Stability testing

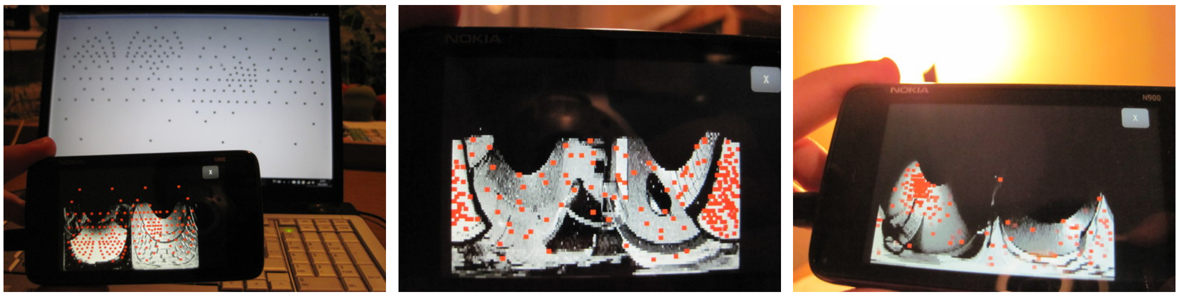
In order to test the stability of the implemented software, we kept the program running for 24h and logged all exceptional events. During the testing, we changed the configuration of the server and the importance sampling algorithm occasionally. The lighting conditions of the environment were changing from darkness to bright light throughout the testing period.

Two clients were connected to the server for the entire period (*Renderer* and *Visualization*) while other instances were connected and disconnected occasionally. The changes of the configuration were made via web browsers Mozilla Firefox and Microsoft IE.

During the testing period, no major errors occurred and the system kept operating normally. The only unexpected event that we recorded was a client not receiving a reply to its request. This situation happened approximately once in 20 000 requests. It might be due to some network problems that are not caused by our implementation. Anyway, the occurrence of this error does not cause any major problems as the client automatically resends the request after not receiving a reply in a specified timeout period (2 seconds by default). Resending a request if the original request was delayed but not lost does not cause any problems neither, as such situations are handled by the server.

5.2.2 Correctness of sampling pattern and photometric calibration

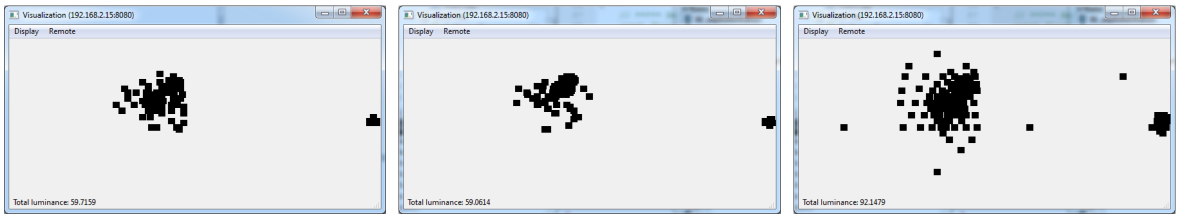
In order to test the correctness of the sampling pattern, a debug mode can be turned on via the configuration tool. If it is enabled, the captured light probe mapped into polar coordinates is displayed on the display of the Nokia N900 smartphone, tone mapped to a range displayable by the device. The Sigmoid tone mapping operator is used [62, 63]. Positions of samples are visualized by red dots, as shown in photographs in Figure 5.5. It can be observed that the density of samples corresponds to the distribution of brightness in the image.



(a) (b) (c)

Figure 5.5: Photographs of the Nokia N900 display while the device is running in debug mode. Photograph (a): display of the phone and a visualization running on a laptop, displaying the same results in a different coordinate system; (b) and (c): display of the phone in different settings. Note that the samples are more concentrated in brighter regions, as expected.

The regions of the hemisphere that are not covered by our fisheye lens are also visualized on the display. This information gives an idea about the correctness of the geometric calibration. We found that the blind area is slightly broader than expected. This error can be attributed to an imprecision of the calibration matrix and distortion coefficients. This imprecision is due to the inability of the corner detection software that we used to detect



(a) Pharr (b) Hemigon (c) Q2-Tree

Figure 5.6: Sampling pattern produced by the three algorithms. The screenshots were captured during a live operation of the device, so the lighting conditions might not be exactly the same in all three images.

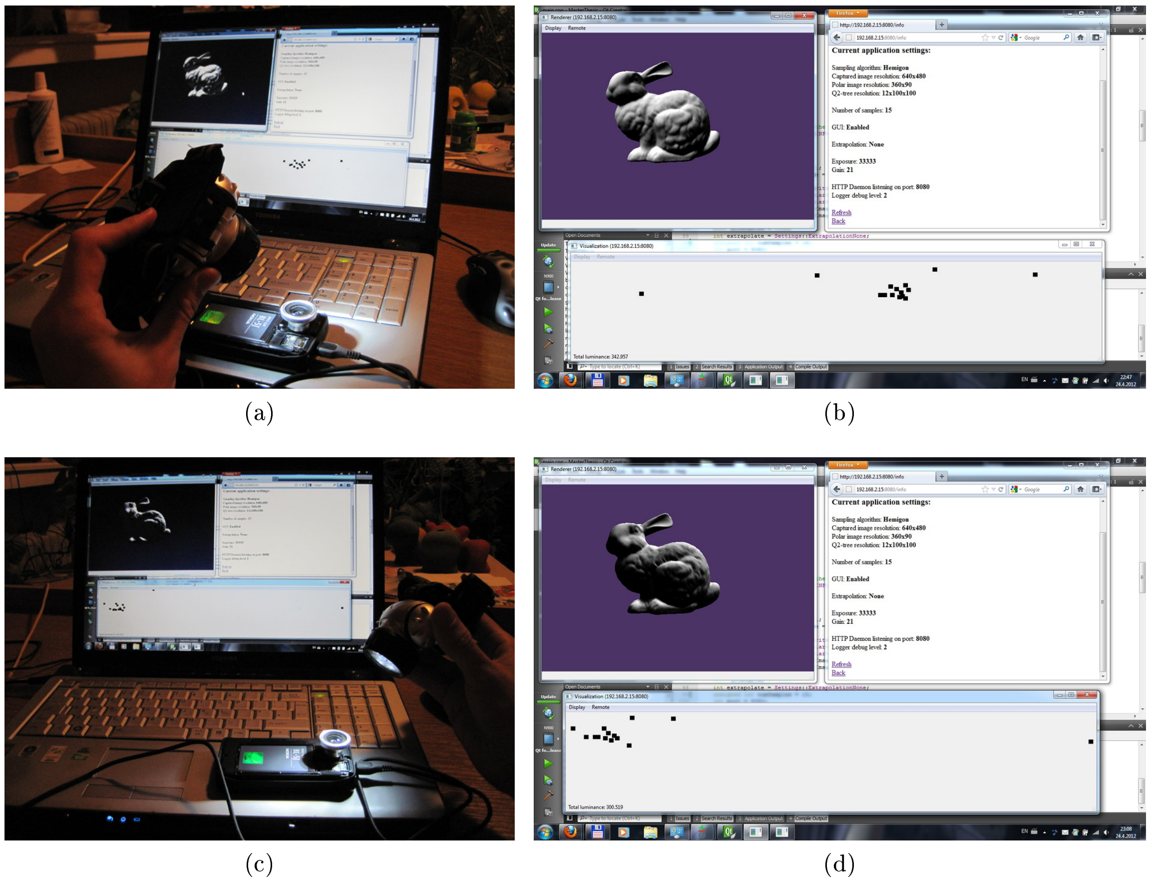
corners in the border regions of the image, as explained in Section 4.2.1. Section 4.2.1 suggests a solution to this problem that is planned as future work. Further examination of the geometric calibration, as described in Section 5.2.3, found that the central regions of the view are calibrated correctly.

In order to debug and test the client and the transmission and serialization / deserialization of the sampling data, we implemented a visualization tool that connects to the server and displays the samples. It also displays the total energy of the samples, which corresponds to the illuminance of the place where the camera is positioned and can be used to test the photometric calibration. The measured illuminance is close to the expected values (we measured approximately 40 lux for indoor environments with dim lighting and approximately 15 000 lux for outdoor environments on a bright day if the camera is not placed directly in sunlight).

5.2.3 Renderer testing

In order to illustrate the use of our framework and to test the its usability, we implemented a simple renderer in OpenGL with GLSL shaders. It renderers a scene lit by directional light sources of positions and intensities corresponding to the samples taken by our importance sampling algorithm. The renderer runs in real-time. Changes in the lighting of the environment where the capture device is placed immediately effect the lighting of the virtual environment. Figure 5.7 shows photographs of the testing setup. In this particular case, the user uses a flashlight pointed at the camera. The illumination of the virtual environment corresponds to the movements of the real flashlight.

We also implemented a debug mode that displays the positions of directional light sources as lines in 3D space. By moving a flashlight around the camera it can be observed that the virtual light sources correctly correspond to the position of the real light sources.


(a)
(b)
(c)
(d)
Firefox
Renderer (192.168.2.15:8080)
Visualization (192.168.2.15:8080)
Display
Remote

http://192.168.2.15:8080/info

Current application settings:

Sampling algorithm: **Hemigon**
Captured image resolution: **640x480**
Polar image resolution: **360x90**
Q2-tree resolution: **12x100x100**

Number of samples: **15**

GUI: **Enabled**

Extrapolation: **None**

Exposure: **33333**
Gain: **21**

HTTP Daemon listening on port: **8080**
Logger debug level: **2**

Refresh
Back

http://192.168.2.15:8080/info

Current application settings:

Sampling algorithm: **Hemigon**
Captured image resolution: **640x480**
Polar image resolution: **360x90**
Q2-tree resolution: **12x100x100**

Number of samples: **15**

GUI: **Enabled**

Extrapolation: **None**

Exposure: **33333**
Gain: **21**

HTTP Daemon listening on port: **8080**
Logger debug level: **2**

Refresh
Back

Figure 5.7: Photographs of a testing setup. The user moves a flashlight around the camera. Changes of the light source positions are immediately observed in the renderer.

5.2.4 Software integration

In order to prove the concept of our design, we integrated our software into existing rendering engines. By integrating our module into Tomas Barak's thesis [64] and the *Zora* platform [54], it has been shown that our framework can be integrated into existing platforms in just a few hours of work. The client module connects to the server and requests sampling results. Every time the processing of a light probe is completed, the client receives the light source positions and notifies the rendering engine via a signal. The renderer then reads the light source positions from the client module and uses them in the rendering.

Figure 5.8 shows screenshots from a scene illuminated by an interactive video environment map, rendered using instant radiosity. In this particular case, the light probe was sampled with 200 samples using the Q2-Tree sampling algorithm. The system operated at real-time frame rates (rendering 47 frames per second; environment map capture and sampling ran asynchronously at 6.7 frames per second).

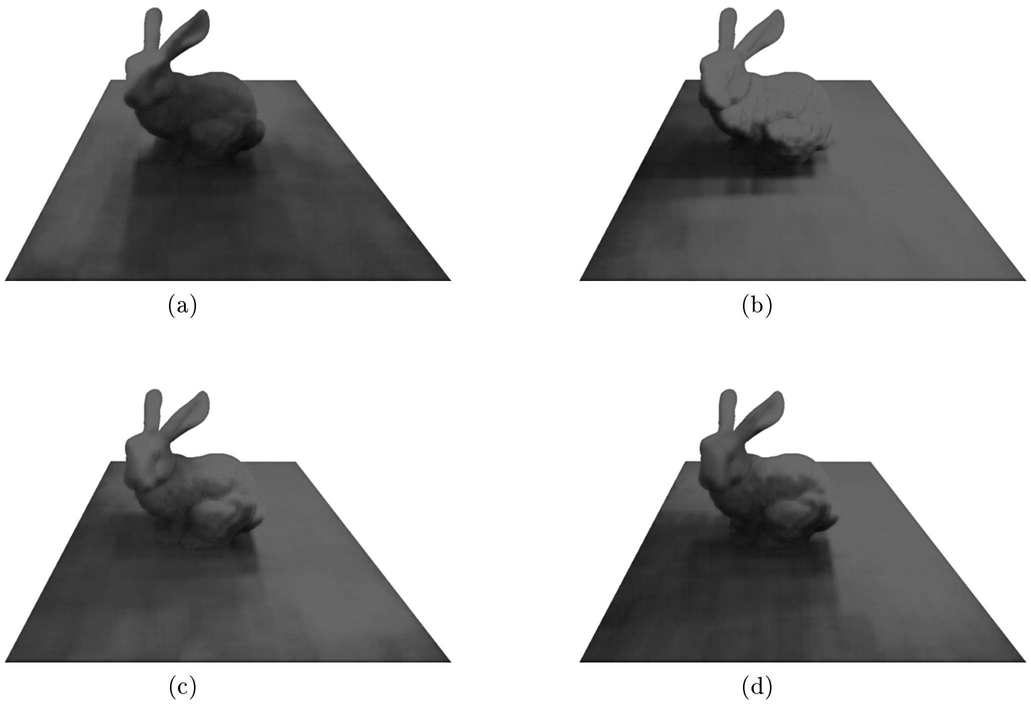


Figure 5.8: Virtual scene lit by light sources decomposed from a light probe of a real environment. Our framework has been integrated into the rendering engine developed by Tomas Barak [64] to produce these screenshots (courtesy of Tomas Barak [64]).

Chapter 6

Conclusion

In this thesis, we reviewed illumination methods for augmented reality. In particular, we focused on existing methods of importance sampling for image-based lighting, and their suitability for dynamic environments. We also discussed the importance of establishing common illumination between real and virtual objects in augmented reality settings. We looked at several different algorithms for the computation of common illumination in mixed reality environments where the scene geometry is approximately known.

We have implemented three different importance sampling methods and compared them in terms of the quality of sampling pattern, temporal coherence, and performance. We proposed a software framework based on a client-server architecture that simplifies the integration of our implementation into existing platforms. We have verified our design on several testing applications. We have also integrated our framework into two existing rendering engines.

We have identified the strengths and weaknesses of the three sampling algorithms. We have verified that the algorithm referred to as *Pharr* [43] provides a good approximation of the environment map and has relatively low usage of computational resources, but suffers poor temporal coherence. We have found that although the low pass filtering proposed by Havran et al. [16] improves the temporal coherence, it causes artifacts if the illumination changes are abrupt. We have also found that although the *Q2-Tree* sampling algorithm [15] has a strong temporal coherence, it does not handle well subtle movements of light sources unless the number of samples is very high.

As a future work, we suggest designing a better low pass filter for the trajectories of samples in the *Hemigon* algorithm [16]. If the filter handled well abrupt illumination changes, the quality of the sampling pattern for dynamic environment sequences of this algorithm would be superior to the other two implemented algorithms.

Apart from that, a fully automated tool for the geometric calibration of the camera should be implemented. The Nokia N900 smartphone provides sufficient computational resources to perform the calibration on-chip. Computing the camera model directly on the phone would simplify the calibration process.

Bibliography

[1] O. Bimber and R. Raskar. *Spatial augmented reality.* AK Peters, 2005.

[2] K. Jacobs and C. Loscos. Classification of illumination methods for mixed reality. In *Computer Graphics Forum*, volume 25, pages 29–52. Amsterdam: North Holland, 1982, 2006.

[3] A. Fournier, A.S. Gunawan, and C. Romanzin. Common illumination between real and computer generated scenes. In *Graphics Interface*, pages 254–254. CANADIAN INFORMATION PROCESSING SOCIETY, 1993.

[4] P. Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *ACM SIGGRAPH 2008 classes*, pages 1–10. ACM, 2008.

[5] S. Gibson, J. Cook, T. Howard, and R. Hubbold. Rapid shadow generation in real-world lighting environments. In *Proceedings of the 14th Eurographics workshop on Rendering*, pages 219–229. Citeseer, 2003.

[6] K. Jacobs, J.D. Nahmias, C. Angus, A. Reche, C. Loscos, and A. Steed. Automatic generation of consistent shadows for augmented reality. In *Proceedings of Graphics Interface 2005*, pages 113–120. Canadian Human-Computer Communications Society, 2005.

[7] M. Knecht, C. Traxler, O. Mattausch, W. Purgathofer, and M. Wimmer. Differential instant radiosity for mixed reality. In *Mixed and Augmented Reality (ISMAR), 2010 9th IEEE International Symposium on*, pages 99–107. IEEE, 2010.

[8] A. Keller. Instant radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56. ACM Press/Addison-Wesley Publishing Co., 1997.

[9] T. Ritschel, T. Grosch, M.H. Kim, H.P. Seidel, C. Dachsbacher, and J. Kautz. Imperfect shadow maps for efficient computation of indirect illumination. In *ACM Transactions on Graphics (TOG)*, volume 27, page 129. ACM, 2008.

[10] E. Reinhard. *High dynamic range imaging: acquisition, display, and image-based lighting.* Morgan Kaufmann, 2006.

[11] S. Agarwal, R. Ramamoorthi, S. Belongie, and H.W. Jensen. Structured importance sampling of environment maps. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 605–612. ACM, 2003.

[12] T. Kollig and A. Keller. Efficient illumination by high dynamic range images. In *Proceedings of the 14th Eurographics workshop on Rendering*, pages 45–50. Eurographics Association, 2003.

[13] V. Ostromoukhov, C. Donohue, and P.M. Jodoin. Fast hierarchical importance sampling with blue noise properties. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 488–495. ACM, 2004.

[14] P. Debevec. A median cut algorithm for light probe sampling. In *ACM SIGGRAPH 2008 classes*, pages 1–3. ACM, 2008.

[15] T.T. WONG. Spherical q2-tree for sampling dynamic environment sequences. *Proceedings of Rendering Techniques*, 2005:21–30, 2005.

[16] V. Havran, M. Smyk, G. Krawczyk, K. Myszkowski, and H.P. Seidel. Interactive system for dynamic scene lighting using captured video environment maps. In *Proc. of Eurographics Symposium on Rendering*, pages 43–54, 2005.

[17] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, 2008.

[18] S. Ryoo, C.I. Rodrigues, S.S. Baghsorkhi, S.S. Stone, D.B. Kirk, and W.W. Hwu. Optimization principles and application performance evaluation of a multithreaded gpu using cuda. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pages 73–82. ACM, 2008.

[19] D. Luebke. Cuda: Scalable parallel programming for high-performance scientific computing. In *Biomedical Imaging: From Nano to Macro, 2008. ISBI 2008. 5th IEEE International Symposium on*, pages 836–838. IEEE, 2008.

[20] J. Wind, K. Riege, and M. Bogen. Spinnstube: A seated augmented reality display system. In *Proceedings of Eurographics Symposium on Virtual Environments*, pages 17–23, 2007.

[21] T. Smith and J. Guild. The CIE colorimetric standards and their use. *Transactions of the Optical Society*, 33:73, 1931.

[22] F.E. Nicodemus. Directional reflectance and emissivity of an opaque surface. *Applied Optics*, 4(7):767–773, 1965.

[23] V. Havran. Heuristic ray shooting algorithms. Ph.D. Thesis, Czech Technical University in Prague, November 2000.

[24] I. Wald, W.R. Mark, J. Gunther, S. Boulos, T. Ize, W. Hunt, S.G. Parker, and P. Shirley. State of the art in ray tracing animated scenes. *Eurographics 2007 State of the Art Reports*, pages 89–116, 2007.

[25] K. Suffern. *Ray tracing from the ground up.* AK Peters, Ltd., 2007.

[26] A. Chalmers, T. Davis, and E. Reinhard. *Practical parallel rendering.* AK Peters, Ltd., 2002.

[27] J.T. Kajiya. The rendering equation. *ACM SIGGRAPH Computer Graphics*, 20(4):143–150, 1986.

[28] D.S. Immel, M.F. Cohen, and D.P. Greenberg. A radiosity method for non-diffuse environments. In *ACM SIGGRAPH Computer Graphics*, volume 20, pages 133–142. ACM, 1986.

[29] Szymon Rusinkiewicz. Local Illumination, Reflection, and BRDFs, 2002. Princeton University [Presentation], URL: `http://www.cs.princeton.edu/courses/archive/fall02/cs526/lectures/radiometry.pdf`.

[30] E. Veach and L.J. Guibas. Metropolis light transport. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 65–76. ACM Press/Addison-Wesley Publishing Co., 1997.

[31] M.F. Cohen and J.R. Wallace. *Radiosity and realistic image synthesis.* Morgan Kaufmann, 1993.

[32] P. Dutre, K. Bala, P. Bekaert, and P. Shirley. *Advanced global illumination*, volume 2. AK Peters, 2006.

[33] Zack Waters. Photon mapping - A tutorial. [Online; accessed 7-May-2012], URL: `http://web.cs.wpi.edu/~emmanuel/courses/cs563/write_ups/zackw/photon_mapping/PhotonMapping.html`.

[34] M. Pharr and G. Humphreys. *Physically based rendering: From theory to implementation.* Morgan Kaufmann, 2004.

[35] W. Jarosz, H.W. Jensen, and C. Donner. Advanced global illumination using photon mapping. In *ACM SIGGRAPH 2008 classes*, pages 1–112. ACM, 2008.

[36] H.W. Jensen. *Realistic image synthesis using photon mapping.* AK Peters, Ltd., 2009.

[37] P. Debevec. Image-based lighting. *IEEE Computer Graphics and Applications*, 22(2):26–34, 2002.

[38] P. Heckbert. Color image quantization for frame buffer display. *ACM Siggraph Computer Graphics*, 16(3):297–307, 1982.

[39] Nokia. Device details - Nokia N900, 2012. `http://www.developer.nokia.com/Devices/Device_specifications/N900/`.

[40] J.I. Echevarria and D. Gutierrez. Mobile Computational Photography: Exposure Fusion on the Nokia N900. *Eurographics*, 2011.

[41] IMS CHIPS. HDRC – More than you can see. Technical report, HDRC Imager and Camera Features, 2002. `http://www.ims-chips.de/content/pdftext/HDRC_Imager_Camera_Feature3.pdf`.

[42] V. Havran, K. Dmitriev, and H.P. Seidel. Goniometric diagram mapping for hemisphere. *Short Presentations (Eurographics 2003)*, 5, 2003.

[43] M. Pharr and G. Humphreys. Infinite area light source with importance sampling. In *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2004.

[44] D. Burke, A. Ghosh, and W. Heidrich. Bidirectional importance sampling for illumination from environment maps. In *ACM SIGGRAPH 2004 Sketches*, page 112. ACM, 2004.

[45] K.M. Górski, E. Hivon, and B.D. Wandelt. Analysis issues for large cmb data sets. *Arxiv preprint astro-ph/9812350*, 1998.

[46] Górski, K.M. HEALPix. *Jet Propulsion Laboratory, California Institute of Technology, NASA*, 2012. `http://healpix.jpl.nasa.gov/`.

[47] L. Wan, S. Mak, T. Wong, and C. Leung. Spatio-temporal sampling of dynamic environment sequences. *Visualization and Computer Graphics, IEEE Transactions on*, (99):1–1, 2011.

[48] A. Secord, W. Heidrich, and L. Streit. Fast primitive distribution for illustration. In *Proceedings of the 13th Eurographics workshop on Rendering*, pages 215–226. Eurographics Association, 2002.

[49] F.C. Crow. Summed-area tables for texture mapping. *ACM SIGGRAPH Computer Graphics*, 18(3):207–212, 1984.

[50] I. Sato, Y. Sato, and K. Ikeuchi. Acquiring a radiance distribution to superimpose virtual objects onto a real scene. *Visualization and Computer Graphics, IEEE Transactions on*, 5(1):1–12, 1999.

[51] C. Loscos, G. Drettakis, and L. Robert. Interactive virtual relighting of real scenes. *Visualization and Computer Graphics, IEEE Transactions on*, 6(4):289–305, 2000.

[52] G. Patow and X. Pueyo. A survey of inverse rendering problems. In *Computer graphics forum*, volume 22, pages 663–687. Wiley Online Library, 2003.

[53] J. Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.

[54] Vlastimil Havran. Zora: architecture for augmented reality, 2011. [Personal communication].

[55] A. Adams, D.E. Jacobs, J. Dolson, M. Tico, K. Pulli, E.V. Talvala, B. Ajdin, D. Vaquero, H. Lensch, M. Horowitz, et al. The frankencamera: an experimental platform for computational photography. *ACM Transactions on Graphics (TOG)*, 29(4):29, 2010.

[56] Nokia. Qt - Cross-platform application and UI framework, 2012. `http://qt.nokia.com/`.

[57] R. Hartley, A. Zisserman, and Inc ebrary. *Multiple view geometry in computer vision*, volume 2. Cambridge Univ Press, 2003.

[58] Matt Loper. CamChecker - A Camera Calibration Tool, 2003. `http://matt.loper.org/CamChecker/`.

[59] Davide Scaramuzza. OCamCalib: Omnidirectional Camera Calibration Toolbox for Matlab, 2012. `http://sites.google.com/site/scarabotix/ocamcalib-toolbox`.

[60] OpenCV. Open Source Computer Vision, 2012. `http://opencv.willowgarage.com/wiki/`.

[61] George S. Fishman. *Monte Carlo: concepts, algorithms, and applications.* Springer, 1996.

[62] E. Reinhard, T. Kunkel, Y. Marion, J. Brouillat, R. Cozot, and K. Bouatouch. Image display algorithms for high and low dynamic range display devices. *Journal of the Society for Information Display*, 15(12):997–1014, 2007.

[63] Tomáš Nikodým. Ray tracing algorithm for interactive applications. Bachelor's Thesis, Czech Technical University in Prague, May 2010.

[64] Tomáš Barák. Global Illumination via Instant Radiosity. Master's Thesis, Czech Technical University in Prague, May 2012. [Not yet published].

[65] FCAM. FCam API - Getting started. [Online; accessed 3-May-2012], URL: `http://fcam.garage.maemo.org/gettingStarted.html`.

Appendix A

User manual

The user manual provides information about the communication protocol and the configuration of the server.

A.1 Communication protocol

This section lists all requests supported by the server. The requests are send by requesting the following url:

`http://ADDRESS:PORT/REQUEST`

where ADDRESS is the IP address of the device and PORT is the port (by default 8080). Options for the REQUEST are listed below.

A.1.1 Information request

Request: `info`
Format of reply: HTML
Description: requests information about current application settings.

A.1.2 Change requests

Request: `change?[param]=[value]`
Description: requests a change of the server configuration.

Supported parameters and their values are listed below. If the request is sent without any parameters, the server sends a list of supported requests and an HTML form to change the configuration. For any other change request, the server responds with an HTML page confirming the change.

Request: `change?algorithm=[value]`
Values: `hemigon`, `q2tree`, `pharr`
Description: requests a change of the sampling algorithm being used.

Request: `change?gui=[value]`
Values: `on`, `off`
Description: turns on or off the debug mode. In debug mode, the tone-mapped polar image and sample positions are displayed on the N900 display.

Request: `change?extrapolation=[value]`
Values: `none`, `average`
Description: changes the extrapolation method being used. If the value is `none`, the regions outside of the camera's field of view contribute no illumination. If the value is `average`, then the average pixel color and intensity are used for the pixels outside of the camera's field of view.

Request: `change?samples=[value]`
Value: number of samples (integer)
Description: changes the number of samples (light sources).

A.1.3 Data requests

If the client requests data, the server adds the client socket to a queue and send back the requested data when a new frame is processed. That means that the request can be sent again on receiving a reply to the previous data request if we want to receive the positions of light sources for every frame. Two distinct formats of data are supported: XML and plain text.

Request: `xml`
Format of reply: XML
Description: requests the sampling data in the XML format.

Request: `data`
Format of reply: plain text
Description: requests the sampling data in the plain text format.

A.1.4 Image requests

Request: `image?[value]`
Value: index of the image in a burst of images with varying exposures. Ranges from 0 (shortest exposure) to 2 (longest exposure). If no value is specify, it defaults to 2.
Format of reply: JPEG

Description: reads one of the images captured in a burst of images of varying exposures. The images are captured by pressing the shutter button on the phone.

Request: `capture`
Format of reply: JPEG
Description: captures a single shot image and sends it back to the client.

A.1.5 Screenshot requests

Request: `screenshot`
Format of reply: HTML
Description: captures a screenshots of the Nokia N900 display (only the image widget is captured, containing the environment map and positions of samples, but excluding any GUI components such as buttons).
Remarks: the screenshot is assigned a unique name and is stored in a PNG format on the server device (Nokia N900).

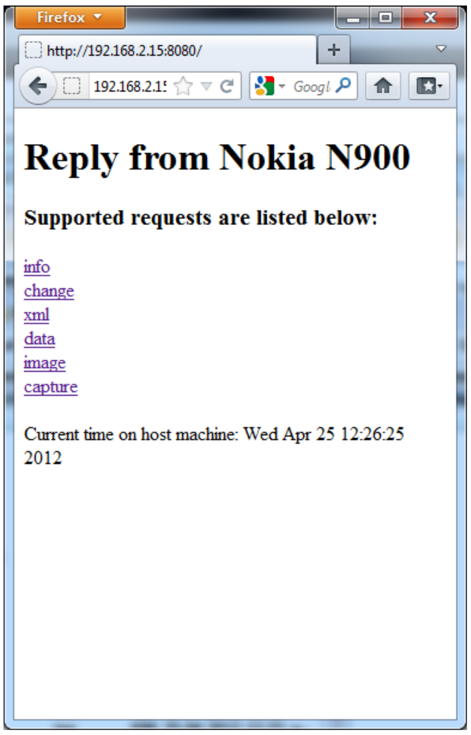
A.1.6 Example

If the device is connected via USB, the default IP address if 192.168.2.15. The following request changes the sampling algorithm being used to the Q2-Tree sampling algorithm.

```
http://192.168.2.15:8080/change?algorithm=q2tree
```

A.2 Configuration

Because the communication protocol is fully compliant with standard web technologies, the configuration of the server can be done in a web browser. Figure A.1 shows a set of screenshots from the configuration. Details about the protocol are given in Section A.1.



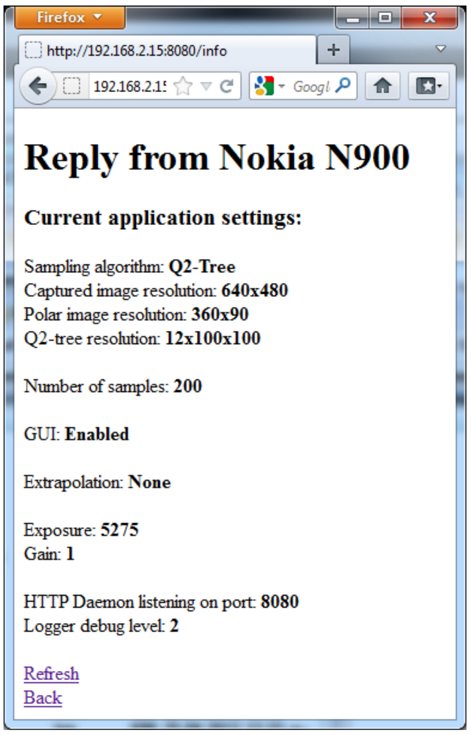
Reply from Nokia N900

Supported requests are listed below:

info
change
xml
data
image
capture

Current time on host machine: Wed Apr 25 12:26:25 2012

(a)



Reply from Nokia N900

Current application settings:

Sampling algorithm: **Q2-Tree**
Captured image resolution: **640x480**
Polar image resolution: **360x90**
Q2-tree resolution: **12x100x100**

Number of samples: **200**

GUI: **Enabled**

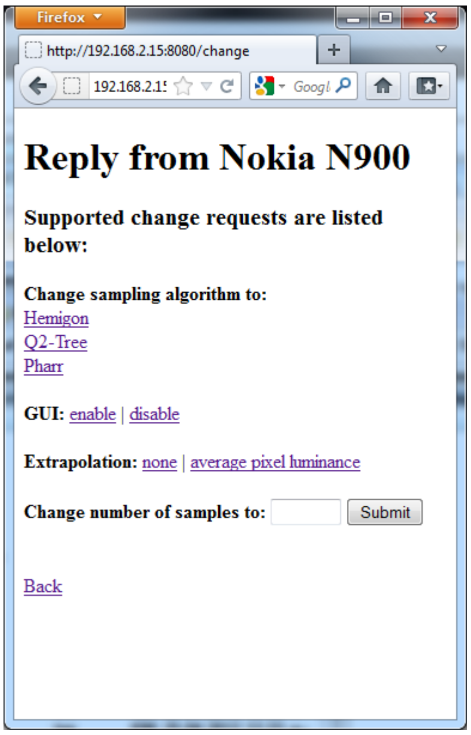
Extrapolation: **None**

Exposure: **5275**
Gain: **1**

HTTP Daemon listening on port: **8080**
Logger debug level: **2**

Refresh
Back

(b)



Reply from Nokia N900

Supported change requests are listed below:

Change sampling algorithm to:
Hemigon
Q2-Tree
Pharr

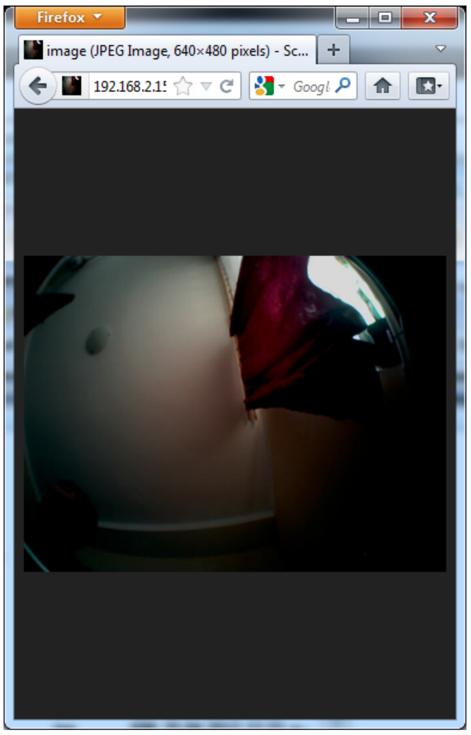
GUI: enable | disable

Extrapolation: none | average pixel luminance

Change number of samples to: Submit

Back

(c)



(d)

Figure A.1: A set of screenshots from the web based configuration of the server.

Appendix B

Installation guide

In this section, we describe the installation and configuration of tools that are required to compile the application from source code. The compilation of client applications is trivial; we provide the `.pro` project files so the projects can be compiled in Qt Creator without any difficulties. A bit more tricky is the compilation of the server application for the Nokia N900 smartphone. It the following paragraphs, we briefly describe the setup that needs to be done prior to the first deployment. Note that this is required only if you want to enable features such as debugging on the target machine; for the deployment of binary packages, this setup is not required.

First of all you need to download the Nokia Suite and install it on your PC. Then you will need to install the MAD developer tool on your Nokia N900 smartphone. By default, the smartphone uses Unix Networking, so if you use Windows, then every time you restart the phone, you will need to run the MAD developer and set the networking mode to Windows Networking. You should also click the Configuration button and make sure the IP address is set to 192.168.2.15. When you connect the phone or change the networking mode, a dialog box should appear; select PC Suite Mode every time you see this dialog. Now switch back to your desktop computer. In the network configuration, you should now see a new Local Area Network with no internet access. In the properties dialog, select IPv4, click properties and set the IP address manually to 192.168.2.14, subnet mask 255.255.255.0. The connection is now set up. When you connect the smartphone via a USB cable and set the networking mode in the MAD developer, you should be able to connect to the phone on address 192.168.2.14. If for some reason you fail to configure the connection via USB, it is still possible to connect to the phone using WiFi.

To setup the Qt Creator, open the Options dialog and select Linux Devices. Follow the instructions in the dialog. To connect to the device for the first time, you will need a password. The generate the password, click on the Developer Password button in MAD developer. You are advised to generate a pair of SSH keys and deploy your public key to the smartphone, so that the Qt Creator can connect to the device without a password.

The configuration is explained in more details here [65].

Appendix C

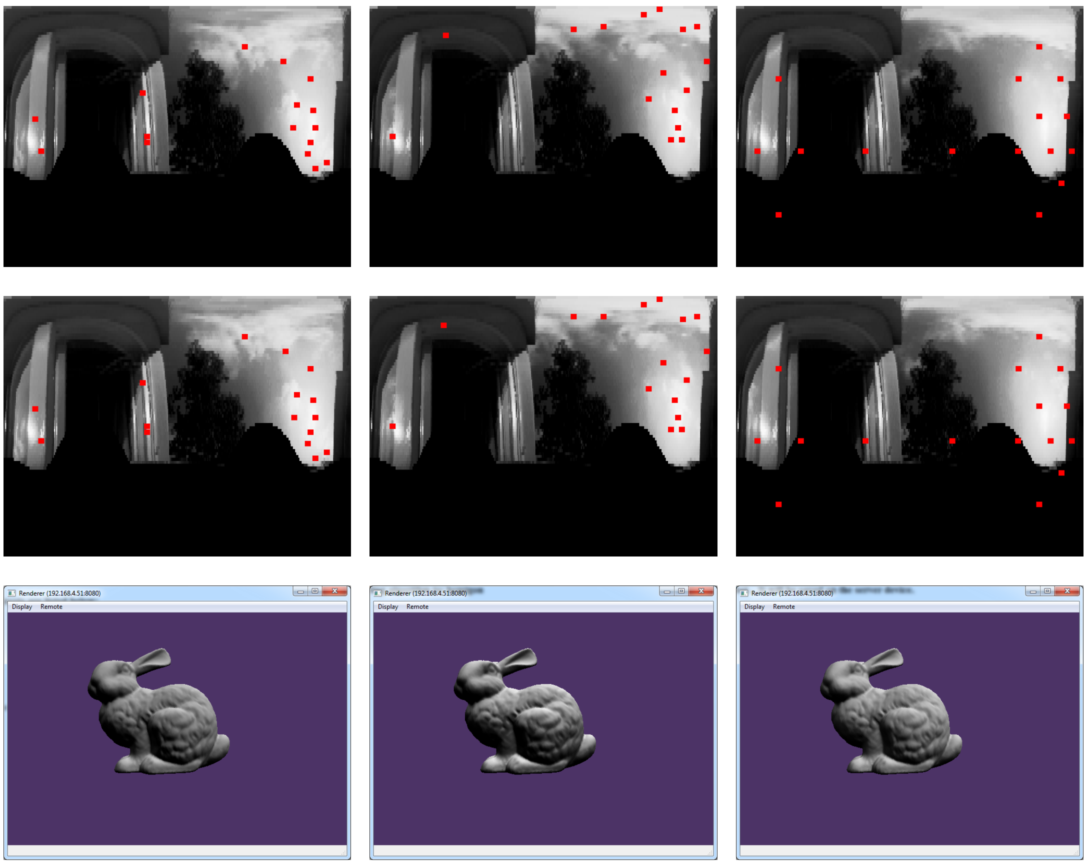
Additional results

The results are presented in Chapter 5. In this appendix, we provide some additional information and figures that did not fit the results chapter.

Figure C.1 shows the results for an outdoor scene. Each column corresponds to one of the three implemented algorithms. In the first two rows, two frames of a sampled environment map are shown. These were captured in a live environment, one second apart, so the position of clouds changed slightly. It can be observed that while the Q2-Tree sampling algorithms maintains the same sampling pattern (only the intensity of light sources changes), the two PDF-based sampling algorithms changed the sampling pattern slightly. In the bottom row, a render of a bunny lit by the directional light sources obtained by sampling the environment map is shown. Note that this experiment was performed in a live environment, so the lighting conditions might not be exactly the same for all three sampling algorithms, as they were captured several seconds apart. Figure C.2 shows a burst of three images of varying exposures taken by the camera during this experiment.

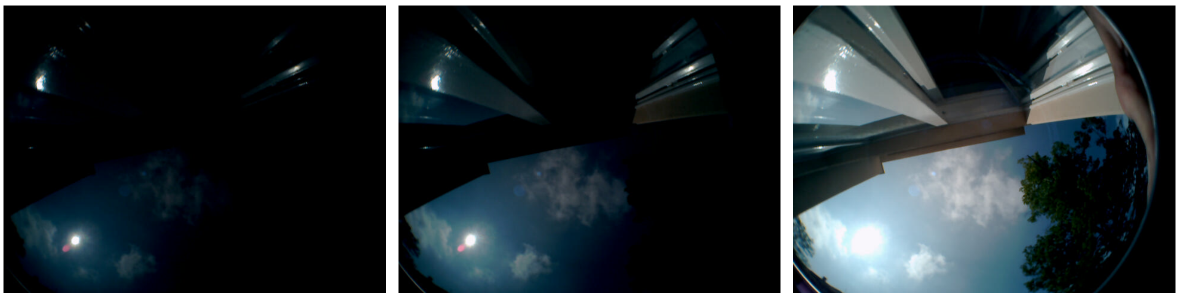
Figure C.3 shows a photograph of the testing setup that was used to capture the data in Figure 5.1. Figure C.4 gives a bust of LDR images that was fused into a HDR image and mapped into polar coordinates to produce the environment map in Figure 5.1. The tests were performed in a dark room with one relatively bright light source - a desk lamp. The illumination at the spot where the camera was positioned during the experiments was about 21 lux (measured by the Nokia N900, as described in Section 4.2.2).

Tables C.1, C.2, and C.3 provide detailed results of the performance testing described in Section 5.1.3. For more information, refer to Section 5.1.3.



(g) Pharr (h) Hemigon (i) Q2-Tree

Figure C.1: The two upper rows show the sampling patterns produced by the three algorithms for two consecutive frames; a render of a bunny lit by each respective set of light sources is given below. From left to right: Pharr, Hemigon, Q2-Tree.



(a) (b) (c)

Figure C.2: A burst of three images of varying exposures taken with the Nokia N900 smartphone. These images were taken in the same environment as Figure C.1.

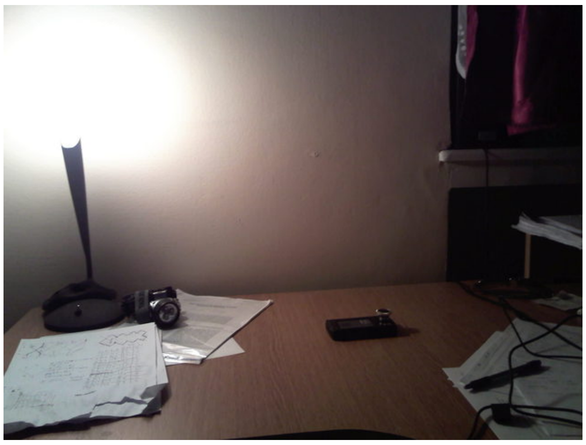
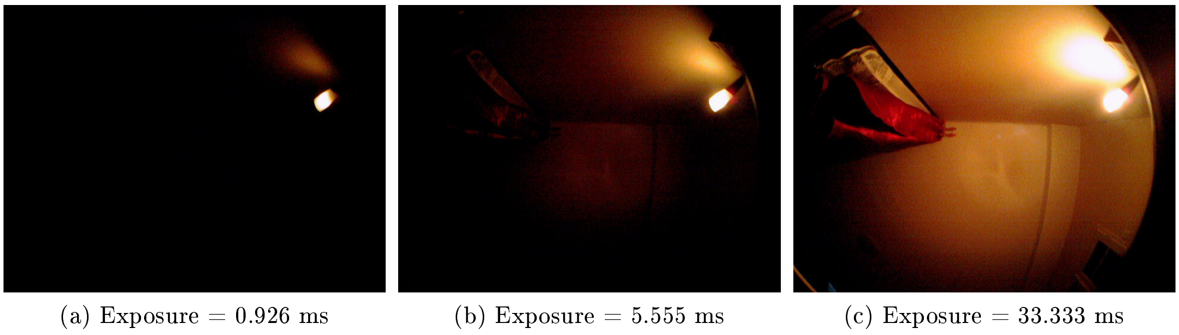


Figure C.3: A photograph of the Nokia N900 smartphone placed in a testing environment. The smartphone captures and processes light probes of the environment; positions of samples are sent over WiFi.



(a) Exposure = 0.926 ms (b) Exposure = 5.555 ms (c) Exposure = 33.333 ms

Figure C.4: A burst of three images of varying exposures taken with the Nokia N900 smartphone. The gain remained unchanged for all three images (gain = 17).

Number of samples	**Overall processing time**
100	130
1000	130
10000	200
20000	280
50000	440

Table C.1: Pharr; processing time versus number of samples. The time is in milliseconds.

Number of samples	**Overall processing time**
100	460
1000	460
5000	480

Table C.2: Hemigon; processing time versus number of samples. The time is in milliseconds.

Number of samples	**Overall processing time**
100	150
1000	170
10000	1250
20000	4100
30000	13000
40000	31000
50000	61000

Table C.3: Q2-Tree; processing time versus number of samples. The time is in milliseconds.

Appendix D

Contents of the data media

.git: git repository

docs: documentation

  data: figures, results, etc.

  diagrams: diagrams in source format

  literature review: text of the semester project (LaTeX)

  release: text of the master thesis (PDF)

  thesis: text of the master thesis (LaTeX)

exe: executables

ext: 3rd-party open source tools (source code)

src: source code

  HttpClient: client module

  N900: application for the Nokia N900 device

  Renderer: OpenGL renderer

  Visualization: visualization application

  intrinsics.txt: camera intrinsics matrix and distortion coefficients

tools: 3rd-party open source tools (binary)