Universiteit Utrecht

Real-Time Dynamic Radiosity for High Quality Global Illumination

Master Thesis

ICA-3220516

Marries van de Hoef

Supervisors:

dr. W.O. Hürst

dr. M. Wand

December, 2013

Version 1.0

Abstract

The radiosity class of techniques is relatively underrepresented in the current studies towards real-time dynamic global illumination, while it provides unique advantages. This thesis builds upon the existing theory for the redistribution of radiosity and the existing progressive refinement adaptation for graphics hardware.

The dynamic radiosity theory is expanded by this thesis to differentiate between types of redistribution based on the emitting or receiving role of the static and dynamic patches. The new theory grounds the proposed *cross redistribution radiosity* algorithm. The redistribution of radiosity originating from static patches is substituted by different types of redistribution. The advantage of this new technique is that the number of rendered hemicubes no longer depends on the number of static patches but only on the number of dynamic patches, which proves to be a significant performance increase. The novel *cross projection function* is essential for realizing the reduction of rendered hemicubes.

Several hardware accelerated radiosity adaptations are developed for comparison. The hardware accelerated adaptation of the progressive refinement algorithm is improved significantly, focusing on quality and general applicability. Furthermore, a novel hardware accelerated adaptation of incremental radiosity is introduced, which introduces the concept of reshooting. Finally, the cross redistribution radiosity theory is supplemented with a fast hardware accelerated adaptation, with the cross projection adaptation as main innovation.

A qualitative analysis demonstrates that all implementations are capable of delivering very high quality global illumination, although the scene properties are restricted. In certain situations cross redistribution radiosity generates artifacts due to undersampling. The execution time is benchmarked for all implementations, which proves that the cross redistribution radiosity adaptation performs well within real-time bounds. A comparative analysis of competing real-time high quality global illumination techniques is favorable to our cross redistribution radiosity method, within the imposed scene restrictions.

Contents

Chapter 1

Introduction

Global illumination is the research topic this thesis focuses on. It is one of the most studied subjects in the rendering research area for the past twenty-five years with hundreds to thousands of published papers. The topic concerns the simulation of both direct and indirect light, which is required to generate realistic illumination. Direct light is received directly from the light source, whereas indirect light is reflected one or more times by other surfaces before it is received. As generally each surface reflects light (otherwise it would be perfectly black), each surface generates indirect light for all other visible surfaces. This complex recursive interdependence of indirect illumination forms the main challenge behind the generation of a global illumination solution.

Many techniques have been devised which are able to solve the global illumination problem correctly in theory. As the global illumination function is a complex continuous function, implementations are always an approximation. The four main techniques from which most other techniques derive are radiosity, path tracing, photon mapping and instant radiosity. These techniques are generally considered to be able to generate high quality global illumination in practice, although descendants may compromise on the quality.

Current challenge

The generation of a high quality global illumination solution can take minutes or hours to complete. To use global illumination in real-time simulations such as games, the solution must be generated very quickly. The scene typically changes each frame in games, and thus requires a new global illumination solution for each frame at thirty frames per second.

Several techniques have been developed which are able to provide global illumination in such short time spans. To achieve the short execution time, major quality concessions are made. For example the calculated illumination is very low frequency and inaccurate, or some limitations may apply such as the restriction to a single bounce of indirect illumination. These low quality global illumination techniques have already been utilized in real-time games.

The current challenge for global illumination algorithms is to increase the ratio between quality and execution time, where the execution time is low. The main objective is to generate a perceptually high quality global illumination at real-time frame rates for general scenes, which would enable the usage in real-time applications such as games.

Radiosity

Algorithms working towards the goal of high quality real-time global illumination have been created in all four main branches of global illumination techniques. Most research is targeted at the photon mapping, instant radiosity and path tracing types of techniques, while the radiosity branch is relatively underrepresented.

However, radiosity has several unique properties which are very beneficial for the application in real-time simulations. As the illumination is stored on the geometry surface, the solution is view-independent which allows for the amortization of the calculation over multiple frames. Furthermore, this extensive caching allows the solution to be modified when a change in the scene occurs, as opposed to regenerating the entire solution. Radiosity also allows for better scaling to many bounces of indirect illumination, as opposed to other (mostly ray based) approaches where the number of possible light paths increases exponentially with the number of bounces.

These advantages of radiosity are not without downsides, such as memory usage and costly visibility determination. However, for usage in games we expect the strengths of radiosity to outweigh the disadvantages.

Objective of this thesis

The goal of this thesis is to introduce a radiosity class technique which generates high quality global illumination solutions at real-time frame rates. This target is very ambitious considering the amount of research already performed in the global illumination area, and the scarcity of previous radiosity research targeting real-time speed. Therefore, this thesis focuses on aspects matching the strengths of radiosity, such as many bounce indirect illumination. Furthermore, phenomena such as specular reflectance and refraction are excluded. The used scenes are relatively simple and the assumption is made that only a small portion of the scene changes each frame. We believe these restrictions are not too prohibitive and still conform to a realistic usage scenario.

Thesis overview

Relevant related work is discussed in Chapter 2. The four main families of global illumination techniques are discussed first, including the state of the art techniques which focus on real-time performance. Subsequently, the radiosity family of techniques is studied in-depth to provide sufficient theoretical background. Finally, radiosity techniques which target real-time performance are examined.

In Chapter 3, the theory on the modification of radiosity solutions is expanded. The new theory is required for the subsequent introduction of our novel cross redistribution radiosity technique and the explanation of its correctness.

Efficient hardware accelerated adaptations of the techniques are required to achieve real-time performance on consumer hardware. Several radiosity based adaptations are provided in Chapter 4. The existing adaptation for progressive refinement radiosity is improved. Subsequently a novel hardware accelerated adaptation for incremental radiosity is proposed, and finally the adaptation for the new cross redistribution radiosity technique is introduced.

The implemented techniques are subjected to a comparative visual and execution time analysis in Chapter 5. Additionally, cross redistribution radiosity is compared with competing real-time global illumination techniques, although this is very challenging due to the major differences.

The conclusion in Chapter 6 will state that are new cross redistribution radiosity is able to generate high quality global illumination at real-time frame rates. The technique does have several limitations for which potential solutions are proposed as future work.

Contributions

The five main contributions of this thesis are listed below.

- The theory of redistributing radiosity is expanded to allow for the specialization of redistribution for specific types of transport.

- Cross redistribution radiosity is formulated using the newly created theory. The main advantage is that the number of hemicubes rendered no longer depends on the number of static patches in the scene, but only on the number of dynamic patches.
- The hardware accelerated adaptation of progressive refinement is improved. The main benefit is a significant quality increase.
- Incremental radiosity is provided with a hardware accelerated adaptation, based on the progressive refinement adaptation. The concept of reshooting is introduced which changes the properties of incremental radiosity to reduce the execution time.
- The new theory on cross redistribution radiosity is modified to a hardware accelerated adaptation. The efficient implementation of the cross projection function is the largest contribution for this adaptation.

Chapter 2

Related Work

Section 2.1 will provide an overview of the major high quality global illumination techniques. The real-time state of the art technique in each class of global illumination algorithms will be compared to our novel cross redistribution technique in Section 5.4.

Radiosity will be discussed thoroughly in Section 2.2. The most relevant radiosity techniques aiming for real-time performance are described in Section 2.3.

2.1 High Quality Global Illumination Techniques

The main four distinct classes of global illumination techniques are discussed in this section. Each class is theoretically capable of generating a perfect global illumination solution. The unique properties of each deriving technique determine for which situations it is specialized.

This thesis focuses on the radiosity family of techniques for the reasons discussed in the introduction. The other techniques are relevant for the comparative analysis in Section 5.4.

2.1.1 Path Tracing

Path tracing is a puristic approach to solving the global illumination problem, which exclusively uses ray tracing.

Ray tracing The origin of ray tracing lies in the ray casting technique by Appel [1], which was improved in the distributed ray tracing technique by Cook et al. to include complex effects by using numerical integration [9]. The integration is effectively done by shooting many rays with parameters spread over the integration domain and averaging the result. The domains suggested in the original paper include the specular distribution function (for glossy specular effects), the area of a light source (for penumbras), and time (for motion blur), amongst others. The integration can be performed simultaneously for all domains, thus the number of rays required depends only on the largest domain.

Path tracing The integration domain of diffuse interreflection (i.e. indirect diffuse light) between surfaces is not included in the distributed ray tracing technique, even though it is an obvious extension. It is likely that it was excluded on purpose because the domain is very large and would require a very large number of rays to properly (and recursively) sample the domain. The solution to the problem of the large domain was brought by Kajiya in the form of stochastic sampling in his path tracing technique [23]. The indirect illumination is approximated by taking a single sample with the expected value of the correct result. By iterating the algorithm and averaging the results, the correct result will eventually be obtained.

As path tracing is able to simulate direct light, diffuse indirect light and specular/refractive indirect light, it provides a complete global illumination solution.

Variance reduction Several extensions of path tracing attempted to reduce the stochastic variance and thus increase the convergence speed. Bi-directional path tracing [26] combines paths from the light source and paths from the eye to increase the number of contributing paths in scenarios with difficult light paths. Metropolis light transport introduced the metropolis sampling strategy in path tracing [43]. The technique iteratively mutates a path to create a proportional distribution of light over the image. This reuse of paths is very efficient for complex lighting conditions.

Hardware acceleration The use of graphics hardware is essential to achieve real-time frame rates. While graphics hardware is specialized to use rasterization as visibility detection mechanism, it can also perform ray tracing. In 2002, Purcell et al. introduced the first basic ray tracer fully accelerated using commodity graphics hardware, despite the restrictions posed by the primitive graphics hardware [34]. The evolution of the GPU to a more general purpose streaming processor allowed for faster ray tracing algorithms which include the construction of the acceleration structure on the GPU [47].

There are specific challenges to a hardware accelerated *path tracing* implementation, including divergent rays and probabilistic path termination. Novák et al. proposed an efficient method to regenerate rays for threads with terminated paths [32]. This approach was optimized and extended to metropolis light transport by van Antwerpen [42].

Real-time implementations The variance in path tracing manifests itself as per-pixel noise in the final image. The number of samples must be doubled to halve the variance. The reduction of variance by taking sufficient samples and using variance reduction techniques is the main challenge to achieve real-time frame rates. A prominent public implementation striving for real-time results is the Brigade renderer [3]. Brigade is currently only able to deliver real-time noise-free images in very favorable lighting conditions. For most scenes the amount of noise is prohibitive.

2.1.2 Photon Mapping

Photon mapping divides the global illumination rendering in two phases, which are joined using a storage mechanism. The storage of light is employed at the final bounce of the illumination before it hits the eye.

Backward ray tracing The concept was introduced by Arvo in his backward ray tracing technique [2]. In the first pass, rays are traced from the light source (i.e. backwards) into the scene. The contribution of each ray is stored on the surface in an illumination map. In the second pass, conventional ray tracing is used and the indirect lighting is sampled from the illumination map. The first pass is view independent which allows for reuse of illumination maps between different view points.

Photon mapping The backward ray tracing technique only included indirect specular reflections on diffuse surfaces, but the caching principle was extended to diffuse interreflections in later techniques [46, 19]. The evolution of the caching methods and the concept of rays shot from the light source eventually culminated in the advent of the photon mapping technique by Jensen [22, 21]. The optional concept of final gathering was introduced, which shifts work from the first phase to the second. Less photons are required to be shot, but gathering in the style of distributed ray tracing must be used during the second phase.

Hardware acceleration The caching employed in photon mapping requires the insertion and gathering of photons in a spatial acceleration structure which is non-trivial on graphics hardware. Purcell et al. created the first hardware accelerated photon mapping algorithm [35].

A significant speed increase was achieved by McGuire and Luebke in their image space photon mapping technique [29], which utilizes the rasterization hardware in GPUs as much as possible. The photon density estimation, which retrieves the final illumination from the photon map, is executed by rendering photon volumes and is further improved by Mara et al. in [27]. Even though real-time results are achieved, the quality is compromised because the number of photons must be limited.

Redistribution The selective photon tracing technique by Dmitriev et al. adaptively modifies the photon map after geometric changes occur [15]. The method to detect changes is an approximation and can only detect modifications within a few light bounces from the light source. Increasing the detection to more bounces requires exponentially more work [14]. The speed improvement by modifying the photon map is limited as the final density estimation phase is currently the most expensive part.

2.1.3 Radiosity

This thesis builds on techniques in the radiosity category of global illumination, therefore an elaborate description of the radiosity field will be given separately in Section 2.2. This section will provide a concise description of the fundamentals of radiosity.

The main characteristic of the radiosity style of global illumination techniques is the extensive use of caching. Between each bounce of light, the results are stored on the surface. The surface is discretized in patches which facilitate the storage. A *form factor* is calculated for each pair of patches which represents the fraction of radiosity transported between the patches. Using the form factors, light is iteratively transported between the patches thus converging to the final global illumination solution.

2.1.4 Instant Radiosity

Although the name suggests this technique to be part of the radiosity family of techniques, instant radiosity is unrelated. The concepts of instant radiosity bear more resemblance to photon mapping, but instead of caching the illumination at the final bounce, it is cached one bounce earlier.

Instant radiosity The theoretical framework of instant radiosity was introduced by Keller [25]. Rays are shot from the light sources into the scene, similar to photon mapping. At each bounce in the scene, a *virtual point light* (VPL) is created representing the radiance from that bounce. The created VPL's are used to illuminate the scene using a rasterization based dynamic lighting technique. Shadow mapping is employed to ensure indirect light to be shadowed correctly, which requires a shadow map to be generated for each VPL.

Difference with photon mapping The main difference with photon mapping is that instant radiosity treats each VPL as *outgoing* light, whereas the photons in the photon map represent *incoming* light. The photons must be sampled using a density estimation method, while the VPLs are used to light other points in the scene. Instant radiosity requires less VPLs to be rendered in comparison to photons, but for each VPL a shadow map must be rendered.

Real-time techniques There are several extensions for instant radiosity which optimize the technique for real-time usage by introducing approximations. The reflective shadow maps technique [12] replaces the initial ray tracing phase by a rasterization phase, although this limits the algorithm to single bounce indirect lighting. The generation of shadow maps for all VPLs is optimized by approximating the shadow maps using the imperfect shadow maps technique [36]. These optimizations of instant radiosity have the disadvantage that the quality of the global illumination solution is compromised.

Quality The instant radiosity technique by Novák et al. solves a major quality problem of instant radiosity [31]. Most instant radiosity techniques produce significant errors for short range indirect illumination due to the clamping of the VPLs to avoid artifacts. This short range illumination is restored by Novák et al. using a screen space technique.

2.2 Radiosity

This section gives an overview of radiosity and the extensions which are most relevant for this thesis. It is attempted to create a coherent explanation and for this purpose numerous small deviations from the original papers have been introduced. Furthermore the formulations of the radiosity equations have been revised to ensure consistency across all techniques.

This section will start with the original radiosity formulation in Section 2.2.1, and its primary expansion in Section 2.2.2. The incorporation of non-diffuse reflectance is discussed in Section 2.2.3. The major speed improvement introduced by progressive refinement is explained in Section 2.3.1, followed by the extension to dynamic environments in Section 2.2.5 on which this thesis expands. Finally, a hierarchical approach is discussed in Section 2.2.6 which introduces important concepts for competing hardware accelerated radiosity techniques.

2.2.1 Initial Formulation

Radiosity is both the name for the first complete model for diffuse global illumination, and the quantity which represents the total radiation leaving a surface. The radiosity technique is based on methods in thermal engineering for the calculation of radiative heat exchange in enclosures. The environment is discretized into patches which exchange radiosity. Goral et al. defined the mathematical basis of the radiosity technique in [17] which consists of two parts: the transmission of radiosity between two patches (using form factors), and the global exchange of radiosity between all patches.

Form factor Equation 2.2.1 is the form factor equation which represents the fraction of radiosity transferred from patch j to i. In this equation, A denotes the patch area, ϕ denotes the angle between the surface normal and the line between $\mathrm{d}A_i$ and $\mathrm{d}A_j$, and r denotes the distance between $\mathrm{d}A_i$ and $\mathrm{d}A_j$. The cosine factors project the direction on the respective surface to correct for oblique angles, following Lambert's cosine law. The π divisor is the energy conservation constant as part of the perfectly diffuse Lambertian lighting. r^2 accounts for the quadratic distance falloff.

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\phi_i \cos\phi_j}{\pi r^2} \mathrm{d}A_j \mathrm{d}A_i \tag{2.2.1}$$

This form factor equation conforms to the following reciprocity relationship: $A_i F_{ij} = A_j F_{ji}$. Solving the form factor equation was done analytically by Goral et al. using a double contour integral. The equation does not allow for occlusions between the patches and is therefore limited to the interior of convex shapes.

Global exchange Radiosity is globally exchanged using Equation 2.2.2, where B is the surface radiosity per unit time and per unit surface area, E is the emissive radiosity of the surface, ρ is the albedo (diffuse reflection coefficient) of the surface, and F_{ij} is the aforementioned form factor. The incoming radiosity from all other patches is computed, multiplied with the albedo and added to any emission of the patch itself (for directly illuminating patches). The equation results in a system of linear equations which was solved by Goral et al. using Gaussian elimination.

$$B_i = E_i + \rho_i \sum_{j=1}^{\mathrm{N}} B_j F_{ij} \qquad \text{for } i = 1, \mathrm{N} \tag{2.2.2}$$

The radiosity is computed for each wavelength band separately. Commonly there are three wavelength bands used which correspond to red, green and blue illumination.

2.2.2 Hemicube

The hemicube generalizes the form factor computation [8], thus allowing for arbitrary scenes with internal occlusion.

Differential form factor The hemicube requires Equation 2.2.1 to be modified to the differential form factor equation as shown in Equation 2.2.3. The added HID function incorporates the occlusion between surfaces by returning zero or one depending on the presence of an occluding object between the points on the surfaces. This function is commonly omitted from the equation, as will be done in the remainder of this thesis.

$$F_{ij} = \int_{A_j} \frac{\cos\phi_i \cos\phi_j}{\pi r^2} \mathrm{HID}\, \mathrm{d}A_j \tag{2.2.3}$$

This equation only solves the integration over A_j. Cohen and Greenberg state that the integrand of the integral over A_i is nearly constant if the distance between the patches is large compared to their size and if there is no partial occlusion, and therefore the outer integral of equation 2.2.1 has little influence. If the conditions for this simplification (a large distance and no partial occlusion) are not met, the patches can be subdivided until they are met.

Geometric analog The differential form factor equation is solved using projection and rasterization which is an efficient method to process general scenes. To understand the validity of this approach, a geometric analogy is constructed as depicted in Figure 2.2.1. When projecting patch j on a unit hemi*sphere* S, the area of the projection $A_{p(j,S)}$ equals $A_j \cos\phi_j / r^2$. Projecting the hemisphere orthographically on the base plane accounts for the $\cos\phi_i$ factor. Combined with a division by π to correct for the total surface of the projected hemisphere (a disk), the result is equal to Equation 2.2.3.

Hemicube Projecting to a curved surface such as a hemisphere is complex. However, if the patch is projected on an intermediate surface I, that projection can be projected again on the hemisphere to create the original projection $p\,(j, S)$. This transitive relation is formally stated as $p\,[p\,(j, I)\,, S] = p\,(j, S)$ and is used by Cohen and Greenberg to introduce the hemicube as an intermediate surface. The projection of the hemicube to the hemisphere is done analytically and is precomputed.

The projection to the hemicube is done using rasterization and a depth buffer algorithm, which requires the hemicube faces to be subdivided into a grid of pixels. Each pixel on the hemicube has a corresponding *delta form factor* which accounts for the projection on the hemisphere, the subsequent projection to the base of the hemisphere and the division by π. To calculate the form factor, the delta form factors for all covered pixels are accumulated.

Global exchange The system of linear equations from Equation 2.2.2 is solved by Cohen and Greenberg using the Gauss-Siedel iterative method, which can intuitively be seen as each patch iteratively gathering the radiance from all other patches.

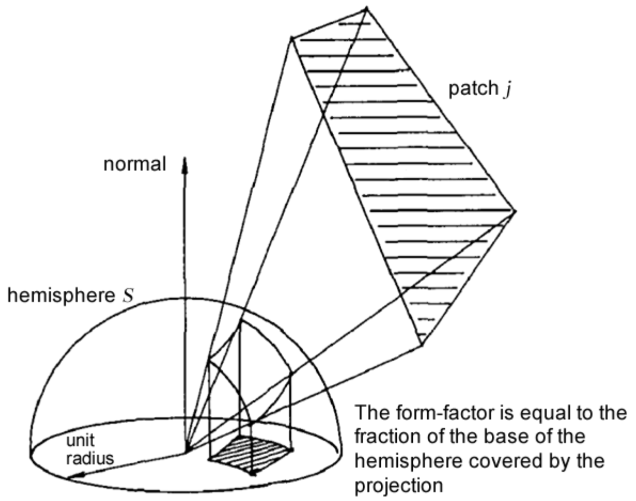

patch j
normal
hemisphere S
unit radius
The form-factor is equal to the fraction of the base of the hemisphere covered by the projection

Figure 2.2.1: Form factor calculation using projection. Image from [8].

Substructuring The algorithm was extended with substructuring in [6] to allow for more detail without increasing the number of patches. The patches are subdivided into multiple *elements* and one hemicube is rendered for each element. The radiance of the patch is the area weighted average of the radiance of its elements. The patches (not the elements) are rendered on the hemicube. Equation 2.2.4 shows the modified global exchange equation which incorporates elements into Equation 2.2.2. q is the index for elements, j is the index for patches. B_j is the area weighted average from all corresponding B_q.

$$B_q = E_q + \rho_q \sum_{j=1}^{\mathrm{N}} B_j F_{qj} \tag{2.2.4}$$

The patches can be much larger when substructuring is used, thus speeding up the hemicube rendering process. The introduction of the elements can be seen as a discretization to numerically integrate the outer integral of Equation 2.2.1, therefore reducing the number of patches required.

The elements are subdivided adaptively and iteratively based on the radiosity gradient, which suggests partial occlusion, and the proximity to other elements relative to the size. The adaptive subdivision refines the representation of sharp lighting transitions such as shadows and it assists in meeting the conditions for the differential form factor equation.

When rendering the radiosity scene for display, the elements are rendered instead of the patches for optimal image quality.

2.2.3 Directional Reflectance

The radiosity techniques described previously operate under the assumption that all light is reflected in a perfectly diffuse manner, i.e. the radiosity of a patch is equal in all directions. Other reflectance functions can be incorporated in radiosity as well, but since radiosity is an iterative technique the directional information must be retained between iterations.

Directional storage The support for directional reflectance was introduced to radiosity by Immel et al. [20]. Radiosity is stored separately for each outgoing direction. The domain of outgoing directions is discretized into a fixed number of directions represented as a grid of pixels on a hemicube.

Each patch is assigned such a hemicube which stores its directional radiosity, therefore enabling persistence of directional information between iterations.

Sillion et al. use a continuous storage solution for directional information [40] which encodes the directional information using spherical harmonics. The number of spherical harmonics coefficients stored by each patch determines the available directional detail.

Reflectance function The mathematical formulation of radiosity must be adapted to include a Bidirectional Reflectance Distribution Function (BRDF) to model the directional reflectance. The form factor representation acquires a dependency on the direction of incoming light. Instead of being a scalar value for each pair of patches, the form factor must depend on the specific direction and can no longer use the integral over the patch area A_j of the other patch. To present this new equation intuitively, we combine the form factor equation with the radiosity exchange equation, similar to Immel et al. [20]. The resulting Equation 2.2.5 is a combination of Equation 2.2.2 and 2.2.3, where the albedo term ρ now incorporates the BRDF. The parameters $\vec{w}$ and $\vec{v}$ denote respectively the outgoing and incoming direction. The $^1/_\pi$ factor has been omitted since this correction was applicable to the Lambertian reflectance function used for perfect diffuse reflectance. In the new formulation, energy conservation must be enforced in the BRDF.

$$B_i(\vec{w}) = E_i(\vec{w}) + \sum_{j=1}^{\mathrm{N}} \int_{A_j} \rho_i(\vec{w}, \vec{v}) \frac{\cos\phi_i \cos\phi_j}{r^2} B_j(-\vec{v})\, \mathrm{d}A_j \tag{2.2.5}$$

Equation 2.2.5 is conceptually identical to the rendering equation formulated by Kajiya for the path tracing global illumination technique [23].

Perfect specular reflection The directional accuracy is limited due to the discretizations and storage requirements, obstructing the simulation of perfect specular reflections associated with mirror-like materials. This limitation can be circumvented by resorting to ray tracing for perfect specular surfaces [40].

Distributed ray tracing An alternative solution is to integrate distributed ray tracing in the radiosity technique to simulate the directional reflectance. The resulting hybrid between radiosity and distributed ray tracing was introduced by Wallace et al. [44] and subsequently improved in [39, 37]. The radiosity technique handles radiance storage in diffuse patches and interactions between diffuse patches while distributed ray tracing governs all directional reflectance. The resulting technique introduces partial view dependence since all directional surfaces visible from the view point must be recursively reevaluated after camera movement.

2.2.4 Progressive Refinement Radiosity

The Gauss-Siedel method of global radiosity exchange described in Section 2.2.2 can be intuitively understood as an iterative gathering process. The convergence speed of this approach is low as all patches must render a hemicube to complete each radiosity propagation step. Cohen et al. reformulated the global exchange equation to change the perspective to patches which shoot radiosity as opposed to gathering patches [7]. When a single hemicube is processed, its radiosity is distributed to *all* other patches. This enables the radiosity to be propagated much faster, as long as the most influential patches are selected for shooting first. The initial convergence speed is orders of magnitude higher compared to the previous gathering approach.

Additionally, the generated form factors are not stored between shooting iterations which removes the $O\left(n^2\right)$ memory requirement present in previous radiosity approaches.

Repeated shooting After a patch has shot its radiosity, it can receive new radiosity from other patches in subsequent iterations. The patch must shoot radiosity again to propagate the new radiosity. However, it cannot shoot the old radiosity again since the receiving patches have no recollection of the amount of radiosity they have already received from this *particular* shooting patch and thus cannot discern the new radiosity from the old radiosity. Therefore the shooting patches can only shoot the radiosity they have not shot before. This requires the stored radiosity to be subdivided in *shot* radiosity and *unshot* radiosity. In the algorithm, each patch stores its unshot radiosity B_i^u and its total radiosity B_i. The *shot* radiosity B_i^s can be deducted using the relationship $B_i = B_i^u + B_i^s$.

Reformulated equations The reciprocity relationship $A_i F_{ij} = A_j F_{ji}$ as introduced in Section 2.2.1 is essential to formulate the new shooting approach using the established radiosity equations. The hemicube rendered at the shooting patch i creates form factors F_{ji} which describe a radiosity transfer from patch j to patch i. By applying the reciprocity relationship, the transfer direction is inverted and we obtain the form factor $F_{ij} = F_{ji}\frac{A_j}{A_i}$ for radiosity transport towards the receiving patch.

Equation 2.2.6 describes the radiosity exchange for a single shooting iteration. ΔB_i is the radiosity difference for the receiving patch i, which must be added to both B_i^u and B_i. The starting condition of this incremental radiosity formulation is $B_j = B_j^u = E_j$, where patch j denotes the shooter. After the shooting iteration over all i is completed, the corresponding B_j^u is set to zero.

$$\Delta B_i = \rho_i B_j^u F_{ji} \frac{A_j}{A_i} \qquad \text{for } i = 1, \mathrm{N} \tag{2.2.6}$$

Substructuring The concept of substructuring described in Section 2.2.2 is also applied in progressive refinement radiosity. The patches are still emitting radiosity and the elements are still receiving radiosity, but the perspective has changed. The hemicubes are rendered from the perspective of the patches, and the elements are projected on those hemicubes. In this configuration less hemicubes are rendered but with higher detail, which is more efficient for contemporary rendering systems to parallelize.

The adaptive subdivision was originally executed exclusively for elements. In progressive refinement radiosity *patches* can also be subdivided. This occurs when the inverted form factor F_{ji} exceeds unity or when the size and intensity of the patch causes illumination inaccuracies.

The source of inaccuracies lies in the inversion of the perspective. Previously the form factor equation integrated over the *emitting* patches, but in the shooting perspective it integrates over the *receiving* elements thus treating the shooting patch as a differential area. Intuitively, the hemicube renders from the perspective of a single point while the shooting patch is actually an area. This simplification causes illumination inaccuracies when the emitted radiosity is intense and the shooting patch is insufficiently subdivided.

Whereas the adaptive subdivision of elements could previously be performed in between iterations, it must be interleaved in progressive refinement radiosity due to its incremental nature.

Disk approximation Using the hemicube method, the integral of Equation 2.2.3 is approximated by accumulating the form factors for all pixels occupied in the hemicube. The projection of an element must cover a significant number of pixels to avoid aliasing artifacts, which requires the hemicube to be rendered at a high resolution.

An alternative method is to shoot a small number of rays to approximate partial occlusion. Using this method the *projected* size of the visible surface can no longer be deducted from the number of occupied pixels on the hemicube. Instead, the size is determined analytically by approximating the surface area using a set of disks, as proposed by Wallace et al. [45]. This disk approximation is formulated in Equation 2.2.7. Note that the equation sums over the *shooter* area. While progressive refinement radiosity uses the shooter perspective, the disk approximation employs the receiver perspective again. This receiver perspective solves the illumination inaccuracies discussed under the substructuring paragraph.

$$F_{ji} = A_i \frac{1}{m} \sum_{k=1}^{m} \frac{\cos\phi_{i,k} \cos\phi_{j,k}}{\pi r_k^2 + {}^{A_j}/_{m}} \tag{2.2.7}$$

By substituting Equation 2.2.7 in Equation 2.2.6, the revised progressive refinement equation is obtained as shown in Equation 2.2.8. It should be noted that the area terms are simplified. This equation is the basis for the shooting techniques in this thesis.

$$\Delta B_i = \rho_i B_j^u A_j \frac{1}{m} \sum_{k=1}^{m} \frac{\cos\phi_{i,k} \cos\phi_{j,k}}{\pi r_k^2 + {}^{A_j}/_{m}} \qquad \text{for } i = 1, \mathrm{N} \tag{2.2.8}$$

2.2.5 Dynamic Radiosity

After a modification of the scene, previous radiosity methods require the entire algorithm to be restarted. It would be more efficient to modify the radiosity solution to include the effects of the changes. This concept is essential in this thesis, and will be built upon in Chapter 3.

Surface property change Support for modifications was proposed by Puech et al. [33], and was based on the progressive refinement radiosity algorithm. The allowed changes are restricted to surface properties, i.e. emission and reflectance. Modifications to the geometry are not supported as they modify the form factors. The effect of the surface changes is a correction of the shot radiosity from the changed patch, which is applied to both the total and unshot radiosity. The unshot radiosity will be propagated in subsequent progressive refinement iterations. Note that the radiosity difference can be negative, therefore causing *negative* light to be propagated.

A change in emission is corrected using Equation 2.2.9, where the radiosity difference ΔB_i must be added to both B_j^u and B_j. The new emission value is represented by E_i'.

$$\Delta B_i = E_i' - E_i \tag{2.2.9}$$

For a change of reflectance parameter, the reflected radiosity $B_i^s - E_i$ must be retrieved and the influence of reflectance term ρ_i must be reversed. The radiosity difference is obtained by multiplying with the change of the reflectance parameter. ρ_i' is the new reflectance parameter.

$$\Delta B_i = (\rho_i' - \rho_i) \frac{B_i^s - E_i}{\rho_i} \tag{2.2.10}$$

Geometry modification The solution of Puech et al. was extended simultaneously by George et al. and Chen to include modifications to the geometry [16, 5]. This thesis expands upon their work. A general theoretical framework is proposed to analyze dynamic radiosity algorithms in Section 3.1. The work of George et al. and Chen is explained using this framework in Section 3.2.

2.2.6 Hierarchical Radiosity

The differential form factor equation defined in Equation 2.2.3 operates under the assumption that the distance between the patches is large compared to their size. Patches are subdivided until this assumption is met or the subdivision limit is reached. Only the lowest subdivision level is used for radiosity transportation, also for transportation over large distances. As basis of the their hierarchical radiosity algorithm, Hanrahan et al. note that higher levels in the subdivision hierarchy can be used for radiosity transportation with distant patches without increasing the introduced error [18]. A set of links is created between entries in the patch hierarchies which describes the transportation paths of radiosity. Intuitively, this approach can be seen as the selective grouping of patches for radiosity transportation over large distances. Hanrahan et al. formulate the technique analogous to the N-body problem and thus derive that the amount of links created is $O(n)$, as opposed to the $O(n^2)$ complexity of traditional radiosity transportation. Clustered hierarchical radiosity extends this concept by allowing the grouping of patches from different surfaces [41].

This thesis does not build on hierarchical radiosity as modifying the set of links is expected to be slow and complex using graphics hardware. It is furthermore expected that modifying the non-hierarchical radiosity solution is faster than recomputing a hierarchical solution.

2.3 Real-Time Radiosity

All radiosity techniques striving to achieve real-time performance focus on utilizing graphics hardware to accelerate their execution. The techniques most relevant for this thesis will be discussed in this Section.

Section 2.3.1 will explain an evolution of the progressive refinement algorithm for graphics hardware which is expanded upon in the remainder of this thesis. The Enlighten radiosity implementation is currently used in videogames, and will be discussed in Section 2.3.2. Finally Section 2.3.3 examines the antiradiance technique, which is the main radiosity-style competitor for the cross redistribution radiosity algorithm introduced in this thesis. The cross redistribution radiosity introduced in this thesis will be compared against antiradiance in Section 5.4.1.

2.3.1 Progressive Refinement Radiosity on Graphics Hardware

The progressive refinement radiosity algorithm is adapted for efficient execution on GPUs by Coombe et al. [10]. Radiosity is stored in the texels of a texture allowing for a fine subdivision of surfaces without introducing vertex overhead. Several modifications to the progressive refinement algorithm are introduced which will be discussed below. Section 4.3 features an improved version of this algorithm and explains additional details.

Stereographic hemisphere projection The shooting patch does not render a hemicube, but instead employs stereographic rendering to project the entire hemisphere at once. The single projection is preferable over the five projections of the hemicube since the rendering cost for visibility determination is mainly determined by the overhead of a projection pass. This method introduces some error since graphics hardware is limited to rendering straight polygons whereas projection on a hemisphere results in curved lines. The error depends on the tessellation of the geometry and the distance to the view origin.

Gathered writes The traditional progressive refinement algorithm writes to all patches which are visible in the rendered hemicube, resulting in scattered memory writes. Graphics hardware however, is optimized for coherent memory writes. Coombe et al. introduce a method to unite the shooting perspective of progressive refinement radiosity with coherent memory writes. After rendering the hemisphere from the shooting perspective, the algorithm does not loop over all hemisphere pixels but instead loops over all receiving elements. Each element checks for its presence in the hemisphere to ensure correct visibility. The disk approximation is used to avoid retrieving the projected size of the receiving element.

To cull surfaces which are entirely occluded in the hemisphere, hardware occlusion queries can be employed during the hemisphere rendering. If no pixels of the surface are rendered to the hemisphere, the occlusion query reports this surface as occluded, thus allowing it to be omitted from the subsequent loop over the receiving elements.

Shooting patch selection Following traditional progressive refinement, Coombe et al. use substructuring the reduce the amount of shooting patches. The selection of shooting patches is done in groups to amortize the selection cost. For each surface a mipmap chain is generated for the unshot radiosity texture. The highest mipmap level consists of one pixel and represents the average radiosity. This value is converted to the total unshot radiosity of the surface and used in a selection process to find the surface with the highest unshot radiosity. All patches on the selected surface shoot their radiosity in turn.

Adaptive subdivision The use of textures as radiosity storage complicates using different subdivision levels across a surface, as textures are inherently uniformly subdivided. Adaptive subdivision is incorporated by Coombe et al. through the use of coarse geometric subdivision. After a significant radiosity gradient has been detected during the shooting process, the geometry of that patch is subdivided. The four new leaves each contain a new small radiosity texture (with e.g. 16x16 pixels) for which the shooting of radiosity is repeated.

The adaptive subdivision introduces a significant overhead in the algorithm. Furthermore, the quality is compromised as radiosity interpolation is not possible across different radiosity textures. This feature is not used in the implementation presented in this thesis.

2.3.2 Enlighten

Enlighten is a commercial middleware radiosity package used in real-time simulations such as games [28]. Execution time is critical for Enlighten, since it must run simultaneously with demanding simulations. The radiosity implementation has been customized specifically for this purpose. The type of optimizations used, such as the low resolution proxy mesh, indicate how the techniques introduced in this thesis may be adapted for usage in games.

Algorithm The radiosity algorithm itself is executed on the CPU using a low resolution proxy mesh of the original mesh which resides in GPU memory. The algorithm computes exclusively indirect illumination, the direct illumination is performed on the GPU using regular lighting techniques. The framebuffer of the previous frame is sampled sparsely on the GPU and subsequently those samples are transferred to CPU memory as input for the radiosity algorithm. The samples are projected on the low resolution proxy mesh which initializes the radiosity algorithm. Only one iteration of radiosity propagation is executed per frame. Multiple light bounces are simulated by using the previous frame as light input for the computation.

After the radiosity solution has been computed, the solution is transferred back to the GPU in a lightmap format. The lightmap is sampled on the high resolution mesh with the use of a smart upsampling technique.

Hierarchical radiosity Although unconfirmed, we expect that Enlighten employs a hierarchical radiosity technique which is explained in Section 2.2.6. This is indicated by the dependency on precomputation and its focus on speed. Hierarchical radiosity is the most efficient radiosity solution when relying on precomputation.

However, dynamic geometric is not easily supported due to this dependance on precomputation. The use of light probes allows for the dynamic geometry to receive indirect illumination but it cannot participate in the radiosity computation. To facilitate participation, the hierarchical radiosity links would have to be inserted and removed throughout the hierarchy, which is a computationally expensive process.

2.3.3 Antiradiance

The main computational bottleneck in radiosity algorithms is the visibility computation. The antiradiance technique omits visibility computations altogether by shooting antiradiance from the backside of patches to create shadow [13]. The antiradiance shot from the back is equal to the radiance received at the front and thus cancels out the radiance erroneously received by shadowed surfaces. The technique expands on hierarchical radiosity to ensure low computational complexity.

Directional storage Antiradiance cannot be propagated diffusely as shadow has a specific direction. The direction of the antiradiance is exactly opposite to the direction of the incoming radiance. To retain the directional information for subsequent iterations, the radiosity is stored in bins which represent discretized directions. Directional reflection is easily incorporated in the algorithm since the directional storage is already accounted for.

Due to the limited accuracy provided by the directional storage of radiosity, the accuracy of the solution is suboptimal. This manifests itself in overly blurred shadows.

Shadow propagation In each iteration of the antiradiance algorithm radiosity is transported once over all links. In the first iteration all surfaces receive radiance from the light sources. In the second iteration, the received radiance is transported as antiradiance from the backside. If a surface is shadowed by two overlapping occluders, it will receive antiradiance twice and thus be overly shadowed. This effect is corrected for by shooting out the received antiradiance as regular radiance from the backside. The second occluder will receive antiradiance from the first occluder and consequently shoot positive radiance from its backside, which corrects for the surplus of received shadow. Multiple layers of occluders cause a fluctuating convergence to the final solution. Intermediate results are thus less suitable for display.

Link generation The execution of antiradiance on the GPU is relatively straightforward as only propagation over existing links is required. In the original antiradiance algorithm by Dachsbacher et al. the links have been precomputed on the CPU. Meyer et al. introduced an algorithm to execute the link creation on the GPU including adaptive subdivision [30]. The techniques do not support dynamic geometry, thus requiring the entire algorithm to be repeated when geometry is modified. The technique of Meyer et al. is sufficiently fast to achieve real-time performance in simple scenes while repeating the entire algorithm for each frame.

Chapter 3

Dynamic Radiosity

The main objective of this thesis is to explore the possibilities for radiosity-based techniques which can generate high quality global illumination solutions at real-time frame rates. It is expected that the property of radiosity that the solution can be modified is essential for achieving real-time speed.

To guide the investigation, a new theoretical framework is introduced in Section 3.1. This framework distinguishes and analyzes the modifications required to adapt the radiosity solution and it formulates the correctness requirements.

The existing dynamic radiosity technique is analyzed in Section 3.2. After identifying the fundamental performance problem of the existing technique, the framework is used to guide the creation of the novel cross redistribution radiosity technique as presented in Section 3.3. The theoretical framework is used to show that the quality of the modified radiosity solution is equal to a newly generated solution and consequently that the radiosity solution does not degrade after many subsequent modifications.

3.1 Requirements for Adaptivity

After a geometric change, the interactions between all pairs of patches must potentially be revised. The redistribution function $\boldsymbol{R}(X, Y)$ accounts for the geometrical modifications. Here, X is the set of patches *emitting* radiosity and Y is the set of patches *receiving* radiosity. It should be noted that each patch can emit and receive radiosity. The subdivision in sets X and Y merely denotes the role of the patches. Patches can be assigned both roles simultaneously and thus be in both X and Y.

Basic mathematical implementation The basic implementation $\boldsymbol{R}_g(X, Y)$ for the redistribution function $\boldsymbol{R}(X, Y)$ is formulated in Equation 3.1.1. Vector notation is used to include the set of receiving patches Y into the formulation. (This was handled less explicit in the radiosity equations in Section 2.2). The result of the equation is a vector of radiosity difference values, denoting redistribution radiosity for all patches in Y. A radiosity difference value is used exactly like the ΔB_i radiosity difference value in progressive refinement radiosity, i.e. it is added to both B_i^u and B_i. As opposed to progressive refinement radiosity, this equation is not iterative. The incremental form factor ΔF_{ij} is defined as $\Delta F_{ij} = F'_{ij} - F_{ij}$, where F'_{ij} is the form factor after the modification and F_{ij} is the unmodified form factor. ρ_i is the albedo of patch i.

$$\boldsymbol{R}_g(X, Y) = \left\{ \rho_i \sum_{j \in X} B_j^s \Delta F_{ij} \right\}_{i \in Y} \tag{3.1.1}$$

Intuitively explained, this formula modifies the radiosity which has already been transferred using the old form factor, to the result it would have if it were transferred using the new form factor.

It should be noted that the redistribution function does not result in the final modified solution. Iterations of regular progressive refinement radiosity are necessary for the changes to propagate through the scene and thus for the solution to converge to its final modified state.

Distinguishing types of patches Different sets of patches are distinguished. The total set of patches is defined as P and split into a set of static patches C and a set of geometrically modified patches M, such that $P = C \sqcup M$.

After a modification, radiosity must potentially be transferred between all types of patches, i.e. $\boldsymbol{R}(P, P)$. We distinguish between the different types of patches by separating the redistribution in the four following parts: $\boldsymbol{R}(C, M)$, $\boldsymbol{R}(M, C)$, $\boldsymbol{R}(M, M)$ and $\boldsymbol{R}(C, C)$. Those functions combined are equal to $\boldsymbol{R}(P, P)$, as formulated in Equation 3.1.2.

$$\boldsymbol{R}(P, P) = \Big(\boldsymbol{R}(C, M) + \boldsymbol{R}(M, M)\ ,\ \boldsymbol{R}(M, C) + \boldsymbol{R}(C, C)\Big) \tag{3.1.2}$$

Each part of the redistribution process has distinct properties:

$\boldsymbol{R}(C, M)$ As the position and orientation of M changes, all form factors for radiosity transport from C to M change.

$\boldsymbol{R}(M, C)$ Similar to $\boldsymbol{R}(C, M)$, all form factors change.

$\boldsymbol{R}(M, M)$ Form factors for radiosity transport within the set M change in most situations, but there are exceptions. The form factor remains unmodified when both patches have been transformed using the same rigid transformation, unless the visibility between the patches has changed due to an unrelated patch.

$\boldsymbol{R}(C, C)$ The relevant form factors only change when the visibility between a pair of patches is changed by the intrusion of an unrelated patch, which must be in M.

Implementation considerations The redistribution function does not require a converged solution. The use of the shot radiosity term B^s in Equation 3.1.1 allows the function to execute correctly on partially converged solutions.

However, in the case of multiple consecutive modifications, the redistribution function cannot always be performed when a previous redistribution function is in progress. The requirement is that radiosity must always be modified in chronological order from the viewpoint of the emitter. This is also caused by the use of the shot radiosity term B^s in Equation 3.1.1, as the term may not be modified halfway through the execution of the equation. In practice each emitting patch must perform the redistributions in chronological order.

Another observation is that the algorithm is entirely incremental, thus results are only modified and not recalculated. If an implementation uses an approximation for radiosity redistribution, a small approximation error might accumulate over the course of many subsequent redistribution iterations, resulting in a significant incorrectness. High accuracy is required to prevent this accumulation of error. We can only deviate from this strict accuracy requirement if it can be proven that the error will not accumulate over time.

3.2 Analysis of Incremental Radiosity

Dynamic radiosity techniques were partially discussed in Section 2.2.5. In this section, the incremental radiosity technique by Chen [5] will be analyzed using the theoretical framework proposed in Section 3.1. The analysis will also hold for the other dynamic radiosity technique by George et al. [16] due to its similarity.

Incremental radiosity is an extension of progressive refinement radiosity, and thus uses a shooting perspective to transport radiosity. The form factors are calculated implicitly with every radiosity transfer. After redistributing radiosity to adjust for a geometrical modification, progressive refinement radiosity is employed to converge to the new solution.

Progressive redistribution function The basic redistribution function $\boldsymbol{R}_g(X, Y)$ was defined in Equation 3.1.1. This function can intuitively be understood as the gathering of redistribution radiosity. Incremental radiosity *shoots* redistribution radiosity instead. This gives more scheduling freedom as it is easier to keep the B^s term constant throughout the execution, as explained in Section 3.1. Due to the freedom, radiosity transfers that are expected to have a high influence on the redistribution can be prioritized. Less important transfers can be deferred.

Equation 3.2.1 defines the progressive implementation $\boldsymbol{R}_p(X, Y)$. The reciprocity relationship $A_i F_{ij} = A_j F_{ji}$ has been used to employ the shooting perspective. This formulation bears similarity to the original progressive refinement radiosity distribution equation, as described in Section 2.2.4. Most symbols were explained for Equation 3.1.1. Additionally, A_i denotes the surface area of patch i.

$$\boldsymbol{R}_p(X, Y) = \sum_{j \in X} \left(\left\{ \rho_i B_j^s \Delta F_{ji} \frac{A_j}{A_i} \right\}_{i \in Y} \right) \tag{3.2.1}$$

Intuitively explained, Equation 3.2.1 shoots redistribution radiosity from a shooter j to all receivers i, and repeats this for all shooters.

Patch set specialization The incremental radiosity algorithm employs the progressive redistribution function for all redistribution involving modified patches, thus it uses $\boldsymbol{R}_p(C, M)$, $\boldsymbol{R}_p(M, C)$ and $\boldsymbol{R}_p(M, M)$. It is ignored that form factors can be unchanged in the $\boldsymbol{R}(M, M)$ redistribution. In the case of $\boldsymbol{R}(C, C)$, a lot of incremental form factors are zero. While the same progressive redistribution function $\boldsymbol{R}_p(C, C)$ is used, it is specialized to filter out unmodified form factors. This is done implicitly during form factor calculation. Instead of rendering an entire hemicube, the rendered hemicube region is clipped to areas which are affected by the geometry modification. Hemicube rendering is combined for $\boldsymbol{R}_p(M, C)$ and $\boldsymbol{R}_p(M, M)$, and it is combined for $\boldsymbol{R}_p(C, M)$ and $\boldsymbol{R}_p(C, C)$. This is possible because both partial redistribution functions share the same shooter patches. To calculate the incremental form factor, two hemicubes are rendered for each shooting patch. One to calculate the old form factor and another to calculate the new form factor.

3.3 Cross Redistribution Radiosity

Incremental radiosity uses the shooting perspective to redistribute radiosity and consequently renders a hemicube for each shooter. More specifically, $2 \cdot |M|$ hemicubes are completely rendered and $2 \cdot |C|$ hemicubes are partially rendered, which results a complexity of $O(2|M| + 2|C|)$. Anticipating on the hardware accelerated adaptations in Chapter 4, it is noted that partial rendering of hemicubes is inefficient on graphics hardware and should not perform much better than rendering a complete hemicube in our usage scenario.

In this section the novel cross redistribution radiosity algorithm will be presented which has an $O(4|M|)$ hemicube complexity. It is expected that in common usage scenarios $|M| \ll |C|$, and thus this technique will prove to be a significant performance improvement.

3.3.1 Redistribution Functions

Cross redistribution radiosity does not render hemicubes for patches in C, but only for patches in M. To accomplish this, the redistribution functions must be constructed carefully. The progressive

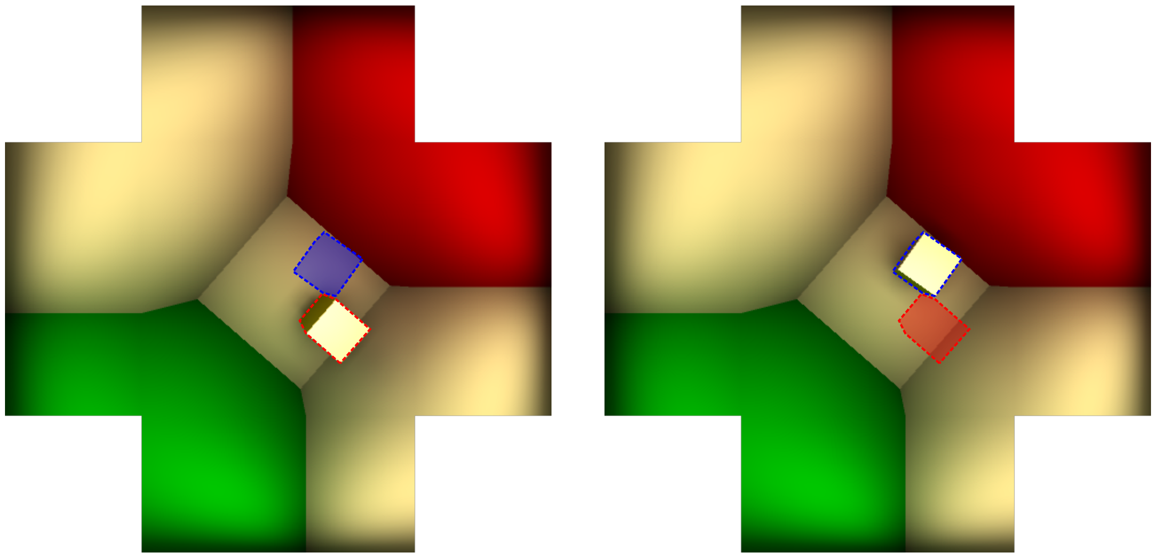


Figure 3.3.1: Two hemicubes, displaying a scene before (left) and after (right) a geometry modification. The transparent areas produce form factor modifications for a single shooting patch in $\boldsymbol{R}_p(C,C)$. The patches in the red area are unoccluded and in the blue area surfaces are occluded.

redistribution functions $\boldsymbol{R}_p(M,C)$ and $\boldsymbol{R}_p(M,M)$ can be used as they render hemicubes for all shooting patches M. To implement the $\boldsymbol{R}(C,M)$ redistribution function, we use the basic gather-based redistribution function $\boldsymbol{R}_g(C,M)$ as this function requires a hemicube to be rendered for all *receiving* patches M. The $\boldsymbol{R}(C,C)$ redistribution function poses a challenge, as neither the gather-based nor the progressive redistribution function can implement this function without rendering a (partial) hemicube for each patch in C. Instead, the new cross projection redistribution function $\boldsymbol{R}_x(C,C)$ is used to redistribute radiosity between patches in C whilst only using hemicubes for the patches in M.

3.3.2 Cross Projection Redistribution Function

In the incremental radiosity technique, the $\boldsymbol{R}_p(C,C)$ function was specialized to render partial hemicubes in the direction of the modified geometry M. The other directions would yield a form factor difference of zero, and therefore are not relevant in the redistribution process. There are two specific areas on the hemicube where the form factors for $\boldsymbol{R}(C,C)$ are changed, as illustrated in Figure 3.3.1. In the hemicube containing the scene after the modification, the area which is unoccluded by the movement yields form factor changes. In the pre-modification hemicube, the form factor changes occur in the area which is occluded by the modification.

An important observation is that all redistributed radiosity passes *through* the modified geometry in either the new or old position. This fact is the basis for the cross projection redistribution function. The redistribution is done for each patch i in C using the patches of the modified geometry (defined in the set M) which are visible from patch i. Those visible patches from M combined cover all areas with form factor modifications, i.e. the transparent areas in Figure 3.3.1.

When viewed from the perspective of a patch in M, the redistribution radiosity passes through it forming a cross-like shape, hence the name cross projection. The cross projection redistribution function can only be applied for a $\boldsymbol{R}(C,C)$ type of redistribution, due to this specialized approach.

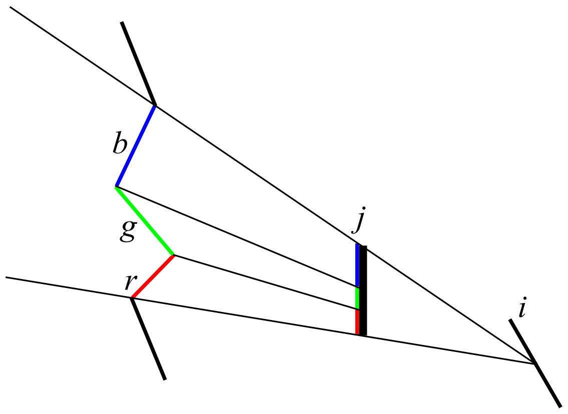


Figure 3.3.2: Cross projection redistribution example showing intermediate projection. In this figure $M = \{j\}$ and $C = \{r, g, b, i, ...\}$.

General approach Figure 3.3.2 depicts the $\boldsymbol{R}_x(C, C)$ redistribution from the part of C which is visible through a single patch $j \in M$ towards a single patch $i \in C$. Instead of directly redistributing the radiosity from the part of C which is visible through j towards i, an intermediate step is introduced. The radiosity from C is projected on j, using patch i as origin for the perspective projection. Subsequently, the radiosity projected on j is transported to i.

The justification of the hemicube, as explained in Section 2.2.2, states that the form factor is transitive under multiple projections when using the same perspective origin. Consequently, the combined form factor for the visible C patches, $F_{ir} + F_{ig} + F_{ib}$ equals F_{ij}. In other words, when the radiosity on j (intermediately projected from r, g and b) is transferred to i, the result is equal to the direct transportation of radiosity from r, g and b to i. This relation also holds when patches in C are partially projected on j because the parts projected outside of j do not contribute to the redistribution.

The final projection from j to i is equal to the $\boldsymbol{R}(M, C)$ redistribution. This is solved using the progressive redistribution function $\boldsymbol{R}_p(M, C)$.

Mathematical formulation Equation 3.3.1 formulates the cross projection redistribution function. The equation bears similarity to the progressive redistribution function $\boldsymbol{R}_p(M, C)$. The shot radiosity B_j^s has been replaced with the projected radiosity function $P(C, j, i)$. The incremental form factor $\Delta F_{ji} = F'_{ji} - F_{ji}$ has been expanded and negated. The negation accounts for the fact that the appearance of patch j causes the removal of radiosity originating from patches behind patch j (i.e. the creation of shadow) and vice versa. j' denotes the new position of patch j. The cross projection is defined exclusively for transport from C to C. The patches in set M, which cause the redistribution, are explicitly required.

$$\boldsymbol{R}_x(C, C) = \sum_{j \in M} \left(\left\{ \rho_i \frac{A_j}{A_i} \left(-F'_{ji} P(C, j', i) + F_{ji} P(C, j, i) \right) \right\}_{i \in C} \right) \tag{3.3.1}$$

The projected radiosity function $P(C, j, i)$ is defined in Equation 3.3.2. The function calculates the area-weighted sum of the shot radiosity projected from C on the surface of j. PerspProjArea (k, j, i) denotes the surface area of a projection on a surface where k describes the patch being projected, j denotes the surface k is projected upon, and i is the origin for the perspective projection. The projected area is normalized using the total area of j.

$$P(C, j, i) = \sum_{k \in C} \left(B_k^s \cdot \text{PerspProjArea}\left(k, j, i\right) / A_j \right) \tag{3.3.2}$$

3.3.3 Hemicube Usage

The radiosity projection function stated in Equation 3.3.2 is implemented using a hemicube rendered at the backside of patch j. The patches in C have been rendered on this hemicube using their shot radiosity. The hemicube is used to create a projection on j from the perspective origin of i. To correct for the different projection origin, a reprojection technique is used which will be discussed in Section 4.5.

The remainder of the cross projection redistribution function is implemented using two hemicubes at the front of patch j to calculate F_{ji} and F'_{ji}, similar to $\boldsymbol{R}_p(M, C)$.

The total number of hemicubes required for the redistribution amounts to $O(4|M|)$. Two are rendered at the front of each patch in M, of which one uses the old position and the other the new position. These hemicubes are used to shoot redistribution radiosity for $\boldsymbol{R}_p(M, C)$, $\boldsymbol{R}_p(M, M)$ and $\boldsymbol{R}_x(C, C)$. Additionally, they are also used to gather radiosity for $\boldsymbol{R}_g(C, M)$. The remaining two are the hemicubes rendered at the backside of each patch in M in both the old and new location. These are used by the intermediate projection in $\boldsymbol{R}_x(C, C)$.

3.3.4 Implementation Considerations

In Section 3.2 several implementation considerations were stated. Firstly, B^s must be constant throughout the redistribution. The redistribution formulation of incremental radiosity allows the calculation to be subdivided without violating the constant B^s rule, therefore creating freedom in the scheduling of the calculations. Cross redistribution radiosity does not offer this flexibility because of the partial gathering formulation. The entire redistribution must be completed before a new redistribution can be initiated.

To prevent errors in the solution to accumulate over multiple redistributions, both the formulation and implementation must take care. There are multiple potential sources of error in cross redistribution radiosity. Firstly, it employs a partial gathering formulation while using a progressive approach for the regular radiosity propagation. The implementation of gathering and shooting will yield slightly different outcomes, resulting in error. Fortunately, this error does not accumulate. The error is introduced when the radiosity, which was gathered during a redistribution, is shot during the regular progressive propagation. However, for the next redistribution where gathering occurs as part of the $\boldsymbol{R}_g(C, M)$ phase, the $\boldsymbol{R}_p(M, C)$ phase reverts *all* previously shot radiosity including any error introduced.

The second potential source of error is the projection function used in the cross projection redistribution function. The accumulation of error depends on the implementation of the projection function, which will be discussed in Section 4.5.

Chapter 4

Hardware Accelerated Adaptations

This thesis targets real-time performance of the proposed algorithms on consumer hardware. To achieve this mark, the use of graphics hardware is essential. Modern GPUs have a multitude of processing power available compared to the CPU, both in terms of arithmetic compute power and memory bandwidth. However to utilize this processing power, algorithms must be adapted specifically for the GPU architecture. Graphics hardware accommodates high computational throughput at the cost of the flexibility. Main factors contributing to this trade-off are the use a single instruction multiple data (SIMD) architecture, and the use of fixed-function hardware such as rasterizers. Even though the trend in graphics hardware is to increase flexibility, it is highly unlikely that the SIMD paradigm will be replaced in the foreseeable future.

Developing a good hardware accelerated adaptation is far from trivial for complicated algorithms such as global illumination techniques. A lot of factors must be taken into accounts such as reducing bottlenecks, coherence and concurrency in memory access, SIMD occupancy for parallelism and latency hiding, and limitations in functionality.

Four hardware acceleration adaptations of radiosity algorithms have been developed for this thesis to allow comparative benchmarking. The progressive refinement adaptation is an improvement of the technique discussed in Section 2.3.1. The incremental radiosity and cross redistribution radiosity adaptations are new.

Gather radiosity An adaptation in the spirit of hemicube radiosity, which is essentially a pure gathering approach. It is discussed in Section 4.2.

Progressive refinement radiosity As detailed in Section 4.3, this adaptation is an improvement upon the existing progressive hardware adaptation discussed in Section 2.3.1.

Incremental radiosity This adaptation features a reformulation of incremental radiosity to reduce the computational load. As explained in Section 4.4, it uses elements of the progressive refinement adaptation.

Cross redistribution radiosity Elements from both the progressive refinement adaptation and the gather adaptation are used for cross redistribution radiosity. The most essential part of this adaptation is the realization of the cross projection redistribution function. This adaptation is discussed in Section 4.5.

All four adaptations use the same underlying principles which are detailed in Section 4.1.

4.1 Common Properties

Our implementation framework supports exclusively rectangular quadrilaterals. Arbitrary meshes using vertex-based radiosity are not supported for simplicity reasons. Radiosity is stored in textures.

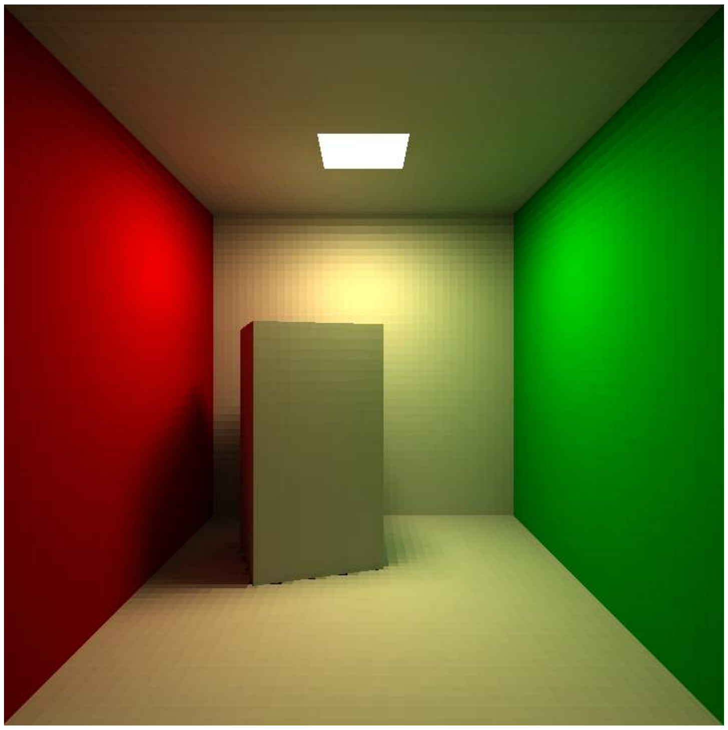


Figure 4.1.1: Radiosity interpolation is disabled which exposes the discrete storage of radiosity. Small squares are subtly visible on the surfaces. Each square is one radiosity element.

Each quadrilateral has a texture region assigned. Linear interpolation is used to interpolate between radiosity samples. Figure 4.1.1 shows a Cornell scene without linear interpolation to expose the used storage structure.

Cross-texture radiosity interpolation It is important that radiosity is interpolated across the boundary of adjacent coplanar surfaces to avoid pronounced interruptions in interpolation. To achieve this cross-texture interpolation, the texture is positioned such that the edges of the quadrilateral intersect with center of the outer boundary of texels[1], as visualized in Figure 4.1.2. Two adjacent quads will have their boundary texels at exactly the same position. As the radiosity computation is based on the texel position, the boundary texels will contain identical radiosity values and therefore radiosity will seemingly interpolate across quadrilaterals without interaction between quadrilaterals being required. This approach is a 2D version of the interpolation approach used in [11].

The disadvantage of this interpolation approach is that the surface area A differs for border elements. This must be accounted for in the radiosity computations.

Having the radiosity samples computed at the edges of quadrilaterals causes artifacts in corners. The corners are singularities in the radiosity computation as the distance to other patches is infinitesimally small. A small offset for the radiosity computation of corner texels solves this problem.

Draw call batching To increase SIMD efficiency in graphics hardware, the use of many small operations is to be avoided. Instead of rendering all quadrilaterals separately, they are batched

[1]A texel is a texture element, i.e. one pixel in a texture.

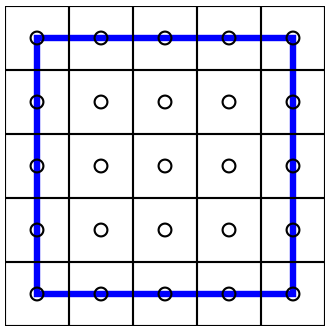


Figure 4.1.2: Texel centers intersect with the blue quad.

into groups for each independently moving object. Batching disallows the switching of state such as texture bindings inside a batch. Consequently, the textures of the batched quadrilaterals are combined into one texture or texture array.

4.2 Gather Radiosity

The gather adaptation functions as a reference implementation for verifying the quality and as a baseline for the performance comparison. The technique is inspired by the hemicube algorithm as explained in Section 2.2.2.

Hemicube rendering Form factor calculation is combined with radiosity propagation to avoid the large storage requirements for the form factors. The radiosity of the scene is rendered to a hemicube for each patch iteratively, as shown in Algorithm 4.1. The form factor is present implicitly in the area covered by each patch. To retrieve the radiosity value, the hemicube is multiplied by a precomputed multiplier map and subsequently integrated. The multiplier map contains the delta form factors for the hemicube to hemisphere conversion as explained in Section 2.2.2.

Integration All values in the processed hemicube must be summed to yield the final radiosity value. The integration method is chosen based on the capabilities of the GPU. The preferred method uses a parallel reduction algorithm in a compute shader to sum all values in a SIMD friendly manner. For compatibility with older graphics hardware, an alternative method is implemented. This method computes a complete mipmap chain for the hemicube. The lowest mipmap level contains the average value for the hemicube, which is multiplied by the hemicube size to obtain the summation. Mipmap generation is not the preferred method as it requires each level to written to and subsequently read from memory, thus adding unnecessary bandwidth usage to an already bandwidth limited computation.

Algorithm 4.1 Gather radiosity

```
for $b \leftarrow 0, maxBounces$ do
    for $i \leftarrow 0, N$ do
        Render all $B_j$ to a hemicube placed on patch $i$
        Multiply the hemicube with the precomputed delta form factors
        Integrate the hemicube
        Store the integration result in $B_i$
    end for
end for
```

Multisampling Graphics cards are equipped with fixed function hardware for multisample anti-aliasing. This method provides more visibility samples without the corresponding increase in bandwidth usage. Using this technique allows the hemicube size to be reduced significantly.

Hemicube orientation The discretization inherently used in rasterization may produce small errors, especially when small intense light sources are used. Even though these errors are barely noticeable individually, they are objectionable when forming patterns over the surface. To counter these patterns, the hemicubes are rotated randomly based on their position.

4.3 Progressive Refinement Radiosity

The hardware accelerated progressive refinement adaptation discussed by Coombe et al. [10] in Section 2.3.1 is used as basis for the adaptation presented in this section. The adaptation by Coombe et al. has several limitations which are addressed in the new adaptation. The improvements result in an adaptation which supports more diverse scenes, has a higher quality and executes faster.

The general approach of both the adaptation by Coombe et al. and the new adaptation is equal and is discussed in Section 4.3.1. The new adaptation is detailed in Section 4.3.2, and the improvements it made over the adaptation by Coombe et al. are discussed in Section 4.3.3.

4.3.1 Gathering Shot Radiosity

An iteration in the classical progressive refinement algorithm writes radiosity to visible elements in the scene. It should be noted that substructuring is used and thus shooting *patches* are subdivided into receiving *elements*. The rendered hemicube determines which elements radiosity is being shot to and their form factor. A parallel implementation would write to the corresponding element for each pixel in the hemicube. Because one element is represented in multiple pixels, such an implementation would lead to concurrent memory writing which is difficult to solve efficiently in such parallel environments. A gathering approach is chosen instead of the shooting strategy. In this formulation, the parallel implementation is executed for each element potentially receiving shot radiosity. Thus the element gathers the radiosity being shot towards it.

The original progressive refinement algorithm calculates the form factor using the coverage of the element in the hemicube. This is difficult to implement efficiently for massively parallel environments such as the GPU due to the searching behavior required. Instead, the disk approximation is used as defined in Section 2.2.4. This alternative method analytically calculates the form factor instead of deriving it from the coverage of the element in the hemicube. A binary visibility test is used, which does not influence the quality as long as the elements are relatively small. An additional benefit of the disk approximation is that the resolution of the rendered hemicube can be relatively low as the form factor accuracy no longer depends on it.

4.3.2 Algorithm

Pseudocode for the progressive refinement adaptation introduced in this thesis is listed in Algorithm 4.2. The adaptation by Coombe et al. is similar to this algorithm. The main differences are explained in Section 4.3.3. The following functions are defined in the algorithm.

ShootingIteration This is the main function which executes one iteration in the progressive algorithm. Instead of shooting for only one patch, radiosity is shot for all shooters in a quad to amortize for the cost of selecting a shooter.

FindShootingQuad The progressive algorithm sorts shooting for patches which are expected to contribute the most. To find the most contributing quad, the quad with the highest area-weighted unshot radiosity value is to be found. By generating a mipmap chain, the average

Algorithm 4.2 Progressive refinement radiosity

```
 1: function ShootingIteration()
 2:     $q \leftarrow$FindShootingQuad()
 3:     if $q.averageUnshotRadiosity < convergenceThreshold$ then
 4:         return
 5:     end if
 6:     $S \leftarrow$GenerateShootingPatches($q$, unshot)
 7:     Clear the unshot radiosity texture of $q$
 8:     for all $s \in S$ do
 9:         Render quad IDs to a hemicube placed on patch $s$
10:         for all elements $e$ do                          ▷ Executed in parallel on the GPU
11:             ReconstructionShader($s$, $e$, unshot)
12:         end for
13:     end for
14: end function

15: function FindShootingQuad()
16:     for all Quads do
17:         Generate a complete mipmap chain for the unshot radiosity texture
18:         Write quad ID to a $1 \times 1$ render target with highest mipmap level as (inverse) depth value
19:     end for
20:     Transfer the quad ID and average unshot radiosity to the CPU from the $1 \times 1$ render target
21:     return quad for the retrieved quad ID
22: end function

23: function GenerateShootingPatches(quad $q$, radiosityType $type$)
24:     $S \leftarrow$ Subdivide $q$ into shooting patches, depending on $q.averageUnshotRadiosity$
25:     for all $s \in S$ do                                 ▷ Executed in parallel on the GPU
26:         Sum the $type$ radiosity for all elements in $s$ and store in an intermediate texture $T$
27:     end for
28:     return $S$
29: end function

30: function ReconstructionShader(shooter $s$, element $e$, radiosityType $type$)
31:     $id \leftarrow$ Sample $s.hemicube$ in the direction of $e$
32:     if $id \neq e.quadId$ then
33:         return
34:     end if
35:     $F_{se} \leftarrow$ Compute form factor using disk approximation
36:     $B_s^u \leftarrow$ Retrieve $type$ radiosity for $s$ from the intermediate texture $T$
37:     $\Delta B_e \leftarrow \rho_e B_s^u A_s F_{se}$
38:     Add $\Delta B_e$ to both $B_e$ and $B_e^u$
39: end function
```

unshot radiosity value is computed for each quad. The depth testing capabilities of the GPU are used to find the quad with the maximum average unshot radiosity. The average unshot radiosity value is used as depth value. Due to the (inverse) depth testing, only the highest depth value is not overwritten in the render target. To retrieve which quad the highest depth value belonged to, the quad ID is stored as color value in the render target. Transferring the final quad ID and average unshot radiosity to the CPU introduces a stall as the GPU must synchronize with the CPU.

GenerateShootingPatches The quad must be subdivided into individual patches. Adaptive subdivision is used which allows very bright quads to be processed in more detail, creating soft shadows. The amount of radiosity to be shot by a shooter is gathered from its elements and stored to be accessible during the radiosity computation in the ReconstructionShader function. The *type* parameter is used in later algorithms. In progressive refinement radiosity only the *unshot* type is being used. As all unshot radiosity has been saved in the intermediate texture, the original unshot radiosity can already be cleared.

ReconstructionShader In this function the actual radiosity computation is performed. The shooter hemicube is used for the visibility test by comparing IDs. Quad IDs are used instead of Element IDs to increase the accuracy of visibility testing with low hemicube resolutions. The ReconstructionShader function can be modified to process multiple shooters at once, thus allowing for batching optimizations.

4.3.3 Improvements

The improvements over the progressive refinement adaptation by Coombe et al. are subdivided in four subjects.

Dynamic shooter subdivision is the inclusion of the GenerateShootingPatches function for reasons explained in

In the implementation of Coombe et al., shooting patches always consist of 4×4 elements. This allows the shooting radiosity to be retrieved from the third lowest mipmap level of the unshot radiosity texture. The main problem with this approach is that the shooting accuracy is constant. In scenarios with a large penumbra, the observed penumbra is not smooth. The adaptation in this thesis dynamically determines the shooter size depending on the amount of radiosity shot. This is done in the GenerateShootingPatches function in Algorithm 4.2. The dynamic subdivision creates a high quality penumbra while keeping the number of shooters low for shots where this accuracy is not required. Furthermore, the subdivision is no longer restricted by mipmap levels as the shooting radiosity is gathered in a separate texture (denoted by T in Algorithm 4.2).

Reconsideration of stereographic rendering The adaptation by Coombe et al. uses stereographic projection, which creates a projection on the hemisphere. The main benefit is that the scene is projected once, whereas a hemicube requires projection for each face. Stereographic projection produces curved polygon edges which is not supported by graphics hardware. The projected geometry must therefore be sufficiently subdivided to reduce the projection error. The optimal level of geometry subdivision depends on the distance to the hemisphere which differs for each shooter. Coombe et al. made a trade-off between quality (due to projection error) and performance (reduced by the increase vertex processing).

Contemporary graphics hardware contains fixed-function support for *dynamic* tessellation, which can vary the tessellation level at run time and thus allows for a high quality projection with only the minimal amount of subdivision required. Stereographic tessellation with dynamic tessellation has been implemented for the adaptation in this thesis. It performed better than static tessellation, but unfortunately not by a large margin. The cause is expected to lie in the execution overhead of dynamic tessellation and the lack of predictability, which is difficult for graphics hardware to schedule efficiently.

Intuitively, we would expect we stereographic projection method to render faster than the hemicube method because it only requires a single projection. However, benchmarking revealed that hemicube rendering is faster than stereographic projection with dynamic tessellation. Profiling suggests that the graphics driver combines the 5 separate projections of hemicube rendering to a single draw call, reducing the multiple projection overhead.

The adaptation presented in this thesis uses regular hemicube rendering as it is the fastest method.

Adaptive form factor subdivision The adaptation by Coombe et al. does not subdivide the disk approximation formula (as defined in Equation 2.2.7) to calculate the form factor. This creates small artifacts in corners. The adaptation in this thesis does subdivide the form factor, but only in specific cases to reduce the performance overhead. The form factor subdivision is initiated when the ratio between the area of the shooter and distance towards the shooter is above a threshold.

Cross texture interpolation As previously discussed in Section 4.1, all adaptations in this thesis supports cross texture interpolation. The adaptation by Coombe et al. does not support this, which clearly had to be taken into account in the construction of their scenes.

4.4 Incremental Radiosity

For the hardware accelerated incremental radiosity adaptation, the theoretical analysis presented in Section 3.2 is reformulated extensively in Section 4.4.1. The adaptation is discussed in Section 4.4.2 and uses elements from the progressive refinement adaptation previously discussed.

4.4.1 Reformulation

The reformulation for incremental radiosity presented in this section has two benefits. It includes one progressive refinement iteration for all patches without significant overhead and it partially *reshoots* radiosity which is faster to compute.

Reshooting is applied for all patches in the set M (which contains geometrically modified patches). As all form factors involving M are expected to change, radiosity towards M can be reshot completely instead of modifying it (thus for the $\boldsymbol{R}_p(C, M)$ and $\boldsymbol{R}_p(M, M)$ redistribution). Reshooting only requires the form factor F'_{ji} to be evaluated instead of the complete incremental form factor $\Delta F_{ji} = F'_{ji} - F_{ji}$. It should be noted that the form factor includes the visibility test, which is computationally most demanding. Furthermore, when using reshooting all patches must shoot their radiosity to complete the recalculation for the patches in M.

Reshooting can be combined effortlessly with a regular progressive refinement iteration. Reshooting shoots shot radiosity B^s_j while a regular progressive iteration shoots unshot radiosity B^u_j. Combining these two can be done by shooting the total radiosity B_j.

$\boldsymbol{R}_p(C, M)$ redistribution The reshooting for $\boldsymbol{R}_p(C, M)$ is formulated in Equation 4.4.1, where $\boldsymbol{B}'$ and $\boldsymbol{B}^{u\prime}$ are vectors of B_i and B^u_i for all $i \in M$ which are partially calculated (i.e. only using radiosity shot from patches in C, radiosity from M is added later in the $\boldsymbol{R}_p(M, M)$ redistribution). It should be noted that the radiosity in $\boldsymbol{B}'$ and $\boldsymbol{B}^{u\prime}$ is overwritten and not accumulated.

$$\boldsymbol{B}' = \boldsymbol{B}^{u\prime} = \sum_{j \in C} \left(\left\{ \rho_i B_j F'_{ji} \frac{A_j}{A_i} \right\}_{i \in M} \right) \tag{4.4.1}$$

$\boldsymbol{R}_p(M, M)$ redistribution The $\boldsymbol{R}_p(M, M)$ redistribution is also done by reshooting and must be performed after the $\boldsymbol{R}_p(C, M)$ redistribution. This is implemented by shooting $\boldsymbol{B}'$ using regular progressive refinement, which is equal to reshooting combined with shooting unshot radiosity. The equality can be explained by regarding $\boldsymbol{B}'$ from Equation 4.4.1 as the shot radiosity $\boldsymbol{B}^s$ plus an unknown quantity of unshot radiosity. Because the $\boldsymbol{R}_p(M, M)$ redistribution is effectively a regular progressive refinement shooting iteration, it is deferred to after the redistribution phase where it will be included naturally in the progressive refinement iterations.

$\boldsymbol{R}_p(M, C)$ redistribution The $\boldsymbol{R}_p(M, C)$ redistribution is not implemented using reshooting. Instead, the progressive redistribution equation is decomposed into two equations using the incremental form factor $\Delta F_{ji} = F'_{ji} - F_{ji}$. The first equation uses the $-F_{ji}$ part, thus effectively shooting negative shot radiosity using the old form factor. The second equation uses the F'_{ji} part and is combined with a regular progressive pass, exactly like $\boldsymbol{R}_p(M, M)$ redistribution. The first equation is executed before $\boldsymbol{B}^s$ is overwritten in the $\boldsymbol{R}_p(C, M)$ redistribution. The second part is combined with $\boldsymbol{R}_p(M, M)$ redistribution and deferred to after the redistribution phase.

$\boldsymbol{R}_p(C, C)$ redistribution In the $\boldsymbol{R}_p(C, C)$ redistribution, the form factor is only modified when the visibility term changes. The redistribution is combined with a regular progressive refinement iteration. Four cases are discerned:

- The previous form factor F_{ji} was nonzero, and the new form factor F'_{ji} is zero. The moving geometry has introduced an occlusion. Negative shot radiosity is being transferred to the patch to account for the occlusion.
- The inverse case where unocclusion occurs. F_{ji} equals zero whereas the new form factor F'_{ji} is nonzero. In this case the total radiosity (shot plus unshot) is transferred.
- The case where both F_{ji} and F'_{ji} are nonzero. This case is equal to a regular progressive refinement iteration, thus unshot radiosity is transferred.
- And finally the case where both form factors are zero and nothing is transferred.

Overview The reformulated redistribution is summarized to give an overview. The following steps are executed in order:

- First part of the $\boldsymbol{R}_p(M, C)$ redistribution. Shoot negative shot radiosity using the old form factors.
- Clear the stored radiosity of patches in M to accommodate reshooting.
- For the $\boldsymbol{R}_p(C, M)$ redistribution, shoot the total radiosity.
- The $\boldsymbol{R}_p(C, C)$ redistribution shoots radiosity depending on the visibility term in the old and new form factors.
- Finally, a regular progressive refinement iteration performs the $\boldsymbol{R}_p(M, M)$ redistribution and the second part of the $\boldsymbol{R}_p(M, C)$ redistribution.

4.4.2 Algorithm

Algorithm 4.3 lists the pseudocode for the incremental radiosity adaptation. The algorithm must be complemented with subsequent progressive refinement iterations for complete redistribution. Two hemicubes are rendered for each patch in the algorithm. For the patches in C, the two hemicubes are rendered in line 11 and 12. For the patches in M, the first hemicube is rendered in line 4 and the second hemicube is rendered as part of the subsequent progressive refinement iteration.

Algorithm 4.3 Incremental radiosity

```
1: function IncrementalRedistribution()
2:     Split the set of Quads into moved quads $M$ and static quads $C$
3:     for all $q \in M$ do
4:         Shoot negative shot radiosity to $C$ similar to Algorithm 4.2
5:     end for
6:     Reset radiosity textures of $M$ to their emissive values
7:     for all $q \in C$ do
8:         $S \leftarrow$ GenerateShootingPatches($q$, shot and unshot)      ▷ Defined in Algorithm 4.2
9:         Clear the radiosity texture of $q$
10:        for all $s \in S$ do
11:            Render quad IDs to a hemicube placed on the old position of patch $s$
12:            Render quad IDs to a hemicube placed on the new position of patch $s$
13:            for all elements $e \in M$ do                ▷ Executed in parallel on the GPU
14:                ReconstructionShader($s$, $e$, total)        ▷ Defined in Algorithm 4.2
15:            end for
16:            for all elements $e \in C$ do                ▷ Executed in parallel on the GPU
17:                IncrementalReconstructionShader($s$, $e$)
18:            end for
19:        end for
20:    end for
21: end function

22: function IncrementalReconstructionShader(shooter $s$, element $e$)
23:     $oldId \leftarrow$ Sample $s.oldHemicube$ in the direction of $e$
24:     $newId \leftarrow$ Sample $s.newHemicube$ in the direction of $e$
25:     if $oldId \neq e.quadId$ and $newId \neq e.quadId$ then
26:         return
27:     else if $oldId = e.quadId$ and $newId \neq e.quadId$ then      ▷ $e$ is occluded
28:         $R_s \leftarrow$ Retrieve negative shot radiosity for $s$ from the intermediate texture $T$
29:     else if $oldId \neq e.quadId$ and $newId = e.quadId$ then      ▷ $e$ is unoccluded
30:         $R_s \leftarrow$ Retrieve total radiosity for $s$ from the intermediate texture $T$
31:     else
32:         $R_s \leftarrow$ Retrieve unshot radiosity for $s$ from the intermediate texture $T$
33:     end if
34:     $F_{se} \leftarrow$ Compute form factor using disk approximation
35:     $\Delta B_e \leftarrow \rho_e R_s A_s F_{se}$
36:     Add $\Delta B_e$ to both $B_e$ and $B_e^u$
37: end function
```

4.5 Cross Redistribution Radiosity

The adaptation of cross redistribution radiosity is explained in Section 4.5.1 and uses both a shooting and a gathering approach, as described in the theory in Section... It also uses reshooting for the patches in M, similar to the incremental radiosity adaptation. The most interesting part is the adaptation for the cross projection redistribution, which is presented in Section 4.5.2. Finally the total algorithm is listed in Section 4.5.3.

4.5.1 Reformulation

Similar to the incremental radiosity adaptation, reshooting is employed for patches in M (i.e. their radiosity is recomputed completely) which improves computational efficiency. Combining redistribution with a progressive refinement iteration, as done in the incremental adaptation, is only done for the patches in M. The combination is not possible for the shooters in C as no hemicubes are rendered for those patches. Substructuring is only applied for the patches in C, not for those in M as this would make the gathering part very expensive.

The cross redistribution radiosity adaptation first applies the gathering $\boldsymbol{R}_g(C, M)$ redistribution, and subsequently the other redistribution parts are executed combined.

$\boldsymbol{R}_g(C, M)$ redistribution The *shot* radiosity from the patches in C is rendered to a hemicube placed on a patch in M, similar to the gather radiosity described in Section 4.2. The gathered radiosity overwrites the existing radiosity for the patches in M, which can be seen as reshooting but in a gathering fashion. This reshooting approach is beneficial because a regular incremental implementation would require two hemicubes to calculate the incremental form factor.

$\boldsymbol{R}_p(M, M)$ redistribution This part of the redistribution is implemented identically to the incremental radiosity adaptation. Reshooting is used and the redistribution is combined with a regular progressive iteration, which is implemented as a regular progressive refinement iteration using the total radiosity. As opposed to the incremental adaptation, the execution is not deferred until after the redistribution phase.

$\boldsymbol{R}_p(M, C)$ redistribution The $\boldsymbol{R}_p(M, C)$ redistribution uses the incremental formulation from Section 3.2 and combines it with a regular progressive iteration. The redistributed radiosity is equal to the negative shot radiosity using the old form factor, plus the total radiosity using the new form factor. As the radiosity for patches in M is overwritten by the $\boldsymbol{R}_g(C, M)$ redistribution, the shot radiosity must be stored beforehand.

$\boldsymbol{R}_x(C, C)$ redistribution The cross projection redistribution is the most interesting redistribution phase. It redistributes the radiosity between two patches in C using the perspective of a patch in M as explained in Section 3.3.2. The adaptation of the theory is introduced in Section 4.5.2.

4.5.2 Cross Projection

The variations of the cross projection redistribution adaptations are discussed. The first adaptation is recalculates the redistribution completely, while the second adaptation modifies the redistribution incrementally. The first non-incremental adaptation will be used in the remainder of the thesis, as the incremental adaptation introduces too much error.

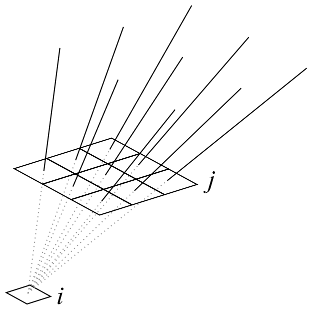


Figure 4.5.1: The construction of sampling rays. Patch j is subdivided into a grid. Each grid cell shoots a ray using patch i as projection origin.

Non-incremental cross projection

The adaptation concerns the $P(C, j, i)$ function in Equation 3.3.1 discussed in Section 3.3.2. The other part of Equation 3.3.1 is executed similar to regular progressive redistribution. The function $P(C, j, i)$ is executed twice, once using the old form factors and once using the new form factors.

The projection function $P(C, j, i)$ calculates the area-weighted sum of the shot radiosity projected from C on the surface of j using patch i as the perspective origin, where $j \in M$ and $i \in C$. As opposed to the implementation provided in Equation 3.3.2, a sampling based approach is chosen to implement $P(C, j, i)$. The surface of j is subdivided and rays are shot through these areas from the perspective of patch i, as shown in Figure 4.5.1. The subdivided grid on patch j can be seen as a grid of pixels used to shoot rays through in ray tracing. The rays gather shot radiosity from patches in C. The area-weighted sum is obtained by taking the average of the gathered radiosity.

Tracing the rays The constructed rays are not traced against the geometry of the patches in C as ray tracing is relatively costly. Instead, the rays sample a pre-rendered image of C. The pre-rendered image is a hemicube positioned on the back of patch j containing the shot radiosity of the patches in C. However, the projection origin of the rays and the hemicube differs. The hemicube is rendered using the center of patch j as projection origin, whereas the rays use patch i as projection origin. To correct the sampling position in the hemicube for the different projection origin, a reprojection technique is used.

Reprojection The goal of the reprojection is to find the correct sampling direction for sampling in the hemicube. The process of reprojection is visualized in Figure 4.5.2. The sampling direction is retrieved using $samplingDirection = p - j.position$, but this requires sampling position p in world space to be known. The sampling position is somewhere along the ray r, but it is unknown at which depth and this information cannot be retrieved without additional visibility testing. An approximation for the depth will be made using the assumption that the depth distribution is coherent. The depth buffer of the rendered hemicube is sampled to find depth information. The sample is taken in the direction of ray r', which is a guess based on ray r. Taking r' parallel to r works well in practice. The sampled depth is used to create an approximated world position p' along ray r. The direction of p' is used as final sampling position.

Figure 4.5.3 displays two images, one without using reprojection and the other with reprojection enabled. The necessity of using reprojection can clearly be seen.

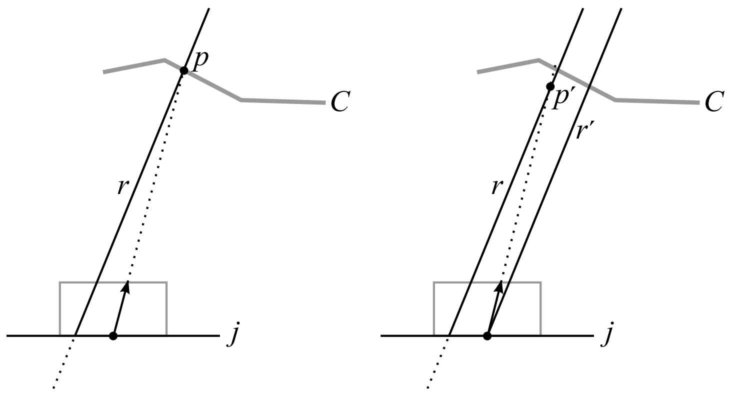


Figure 4.5.2: Reprojecting the sampling direction. The left image shows the desired direction to p. The right image displays the reprojected direction, which is in the direction of p'. The approximated position p' is constructed by sampling the depth in the direction of r', and using the sampled depth as distance along ray r.

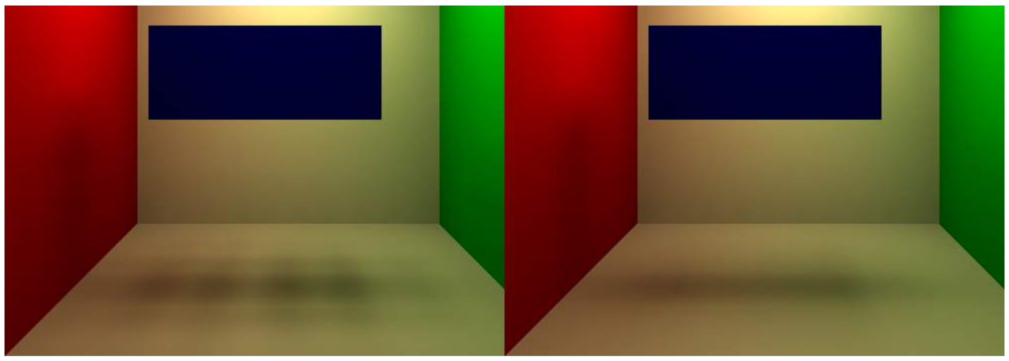


Figure 4.5.3: The effect of reprojection in the radiosity redistribution. Reprojection has been disabled in the left image.

At sharp depth discontinuities, the coherency assumption does not hold and the reprojection approximation is not optimal. However, no artifacts due to this behavior have been observed in the redistribution.

Hemicube rendering and sampling The number of samples for cross projection has to be limited to keep execution time within real-time bounds. To reduce the effects of undersampling, the radiosity hemicube is blurred. Blurring is done implicitly during rendering by employing multisampling in graphics hardware. The hemicube is rendered at resolutions as low as 16×16 per face, while using the highest multisampling level available.

Samples taken from the hemicube are filtered bilinearly for additional blurring. Care must be taken with the edges of the hemicube to avoid erroneous interpolation.

Error analysis Undersampling can be a significant source of error when the light sources are small and sharp. Additionally when the distance between patch i and patch j is small, the sampling rays become divergent thus undersampling is increased. Experiments with dynamically adjusting the quantity of sampling rays have not been fruitful, primarily due to the large number of samples required in the worst case.

Error is not accumulated as the method is non-incremental. For each redistribution, the effect of the previous redistribution is negated and the new redistribution is added, exactly as stated in Equation 3.3.1. However, if the shot radiosity B^s changes (for a patch in C) in between the redistribution iterations, this change is not negated thus introducing error which accumulates. Fortunately, this situation can be avoided. Before a regular progressive refinement iteration is initiated (and thus B^s is expected to change for patches in C), all patches in M are removed from the simulation using the unchanged B^s. After the iteration, the patches in M are inserted again using the modified B^s. As this approach is costly, the implementation in this thesis only uses it for the initial radiosity propagation.

Incremental cross projection

This alternative cross projection adaptation originates from the observation that usually only small modifications are made on the dynamic object between redistributions. Non-incremental cross projection recomputes the redistribution entirely (equal to a deletion and insertion), but it might be more efficient to only calculate the difference. For cross projection, only occlusion and unocclusion is relevant. When patch j moves only slightly to the new position j', the projection of C on j and j' from the perspective of i will have a large overlap. The areas which do not overlap cause occlusion or unocclusion. The occluding area is where j' protrudes over the edge of j, and the unoccluding area is formed by the protruding of j over the edge of j'. To calculate the redistribution for one of these areas, the form factor must be modified and only the radiosity for the protruding area must be regarded.

The radiosity for the (un)occluding area is gathered using a sampling based method. This method is equal to the sampling strategy used for the non-incremental cross projection, with the exception that all rays r which intersect the overlapping area are culled.

The form factor is computed using a disk approximation with a modified area term. The area of the (un)occluding part of the patch is computed by subtracting the overlapping area from the area of the patch. The overlapping area is computed as follows. The patch in its other state (j for occlusion and j' for unocclusion) is projected on the plane of the patch using patch i as perspective origin. The surface of the intersection area is computed using a modified 2D polygon area algorithm which is executed in uv-space. The polygon area algorithm is modified to clip the patch in its other space to the unit square which forms the current patch in uv space. This algorithm is GPU friendly as a constant amount of memory is used and branching is limited. A sweep line algorithm would be the common approach for these types of problems but its GPU compatibility is very poor.

Multiple patches are coplanar which form a quad. (Un)occlusion only occurs when a patch protrudes beyond the quad boundary. Therefore entire quads are used to test (un)occlusion against.

Error analysis Due to the incremental nature of this cross projection adaptation, it is sensitive to accumulating error over multiple frames. When the unocclusion does not exactly negate the occlusion of previous frames, error will accumulate. Unfortunately, the accuracy is not perfect and thus error accumulates which becomes unacceptable after many frames, as visible in Figure 4.5.4.

4.5.3 Algorithm

Pseudocode for the complete cross redistribution radiosity adaptation including cross projection is listed in Algorithm 4.4 and 4.5. Unlike the incremental radiosity adaptation, an additional progressive refinement iteration is not necessary for complete redistribution.

The CrossProjectionReconstructionShader function executes both $\boldsymbol{R}_p(M, C)$ and $\boldsymbol{R}_x(C, C)$ redistribution. The ProjectedRadiosity and Reproject functions have been explained in Sec-

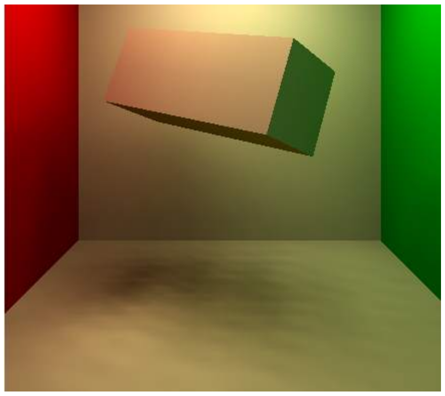


Figure 4.5.4: Error accumulation after 30 frames when using the incremental cross projection adaptation.

tion 4.5.2. The *gridSize* constant in the ProjectedRadiosity function is set by default to 4. It should be noted that the non-incremental cross projection adaptation is used.

Although not listed, the loop over S in the main CrossRedistribution function can be interrupted to decouple the display frame rate from the redistribution rate. This functionality has not been used for the benchmarks. Intermediate display of the redistribution is not visually consistent, thus double buffering must be employed.

Algorithm 4.4 Cross redistribution radiosity, continued in Algorithm 4.5

```
 1: function CrossRedistribution()
 2:     Split the set of Quads into moved quads $M$ and static quads $C$
 3:     for all $q \in M$ do
 4:         $S \leftarrow S \cup$ GenerateShootingPatches($q$, shot)          ▷ Defined in Algorithm 4.2
 5:         Clear the radiosity textures of $q$
 6:     end for
 7:     for all $s \in S$ do
 8:         Render quad IDs                          to a hemicube on the front of $s$, old position
 9:         Render quad IDs and shot radiosity to a hemicube on the front of $s'$, new position
10:         Render shot radiosity to a low resolution hemicube on the back of $s$, old position
11:         Render shot radiosity to a low resolution hemicube on the back of $s'$, new position
12:         GatherRadiosity($s$)
13:         Update shooter texture $T$ for $s$ with unshot radiosity using $B_s$
                                                   ▷ $T$ was filled with shot radiosity in line 4
14:         Clear the unshot radiosity texture of $s$
15:         for all elements $e \in M$ do                      ▷ Executed in parallel on the GPU
16:             ReconstructionShader($s$, $e$, total)             ▷ Defined in Algorithm 4.2
17:         end for
18:         for all elements $e \in C$ do                      ▷ Executed in parallel on the GPU
19:             CrossProjectionReconstructionShader($s$, $e$)
20:         end for
21:     end for
22: end function
```

Continued in Algorithm 4.5

Algorithm 4.5 Continuation of Algorithm 4.4

```
23: function GatherRadiosity(shooter $s$)
24:     Multiply $s.newPositionFrontShotHemicube$ with the precomputed delta form factors
25:     Integrate the multiplied hemicube
26:     $B_s \leftarrow B_s +$ Integration result
27: end function

28: function CrossProjectionReconstructionShader(shooter $s$, element $e$)
29:     $oldId \leftarrow$ Sample $s.oldPositionFrontIdHemicube$ in the direction of $e$
30:     if $oldId = e.quadId$ then
31:         $R_s \leftarrow$ Retrieve negative shot radiosity for $s$ from the intermediate texture $T$
32:         $R_s \leftarrow R_s +$ProjectedRadiosity($s.old$, $e$)
33:         $F_{se} \leftarrow$ Compute old form factor using disk approximation
34:         $\Delta B_e \leftarrow \rho_e R_s A_s F_{se}$
35:     end if
36:     $newId \leftarrow$ Sample $s.newPositionFrontIdHemicube$ in the direction of $e$
37:     if $newId = e.quadId$ then
38:         $R_s \leftarrow$ Retrieve total radiosity for $s$ from the intermediate texture $T$
39:         $R_s \leftarrow R_s +$ProjectedRadiosity($s.new$, $e$)
40:         $F_{se} \leftarrow$ Compute new form factor using disk approximation
41:         $\Delta B_e \leftarrow \Delta B_e + \rho_e R_s A_s F_{se}$
42:     end if
43:     Add $\Delta B_e$ to both $B_e$ and $B_e^u$
44: end function

45: function ProjectedRadiosity(shooter $s$, element $e$)
46:     for $x \leftarrow 0, gridSize$ do
47:         for $y \leftarrow 0, gridSize$ do
48:             Construct ray $r$ from $e$ to grid cell $(x, y)$ on $s$
49:             $r \leftarrow$ Reproject($r$,$s$)
50:             Sample $s.backShotHemicube$ in the direction of $r$
51:             $total \leftarrow total+$sampled radiosity
52:         end for
53:     end for
54:     $areaWeightedRadiosity \leftarrow total/gridSize^2$
55:     return $areaWeightedRadiosity$
56: end function

57: function Reproject(ray $r$, shooter $s$)
58:     $estimatedDepth \leftarrow$ Sample $s.backDepthHemicube$ in the direction of $r$
59:     $p' \leftarrow$ point on ray $r$ at $estimatedDepth$
60:     $reprojectedRay \leftarrow$ Construct ray from $s.centerPosition$ to $p'$
61:     return $reprojectedRay$
62: end function
```

Chapter 5

Measurements and Results

The radiosity adaptations are assessed using the scenes described in Section 5.1.

Section 5.2 examines the visual quality. The method for comparison is discussed first. Subsequently the differences and their causes are analyzed in Section 5.2.1.

In Section 5.3 the execution time will be evaluated. Both the total frame time and the internal composition of the execution time will be analyzed.

To assess the significance of our cross redistribution radiosity method, it will be compared to several other state of the art techniques in Section 5.4.

5.1 Scenes

Four scenes are chosen for the evaluation. The scenes are chosen to highlight the most important aspects for our technique, including many bounce indirect light and light source size. Due to the complexity and variability of global illumination there are many influencing factors, and isolating them all would be nearly impossible. The scenes are constructed based on the aspects which were expected to be most valuable for analysis.

Indirect light An indoor environment where indirect daylight enters at the right side and a rotating box is placed at the left side. This scene is used to examine the results for many bounce indirect illumination. Due to the high albedo of 0.9, comparable to bright white paint, at least 20 light bounces are required for convergence. The scene simulates an indoor environment exclusively lit using indirect sunlight. Indoor scenes used to demonstrate global illumination algorithms often contain many lamps or an abundance of direct sunlight entering through windows. Scenes which rely on many bounce indirect illumination are rarely focused on, while these scenarios are abundant in reality. It should be noted that this scene is exceptionally difficult for global illuminations to compute due to the high number of bounces and the variation in intensity. There are 10296 static elements in the scene and 318 on the rotating box. The cross redistribution adaptation uses 52 elements on the box.

Cornell box with large light source The classic Cornell box scene with a large rotating box in the center. The scene is constructed to be similar to the fully hardware accelerated antiradiance implementation by Meyer et al. [30]. The ceiling light source is relatively large, similar to [30]. It is expected that undersampling in cross redistribution radiosity will not occur in this scene due to the large light source. The number of static elements in this scene is 10254, and the number of elements on the rotating box is 918. For cross redistribution radiosity, the number of elements on the rotating box is reduced to 90.

Cornell box with small light source The large light source is replaced by a small light source to investigate the effects of undersampling in cross redistribution radiosity. The number of elements used is equal to the Cornell box with the large light source.

Low quality Cornell box This scene is identical to the Cornell box with the large light source, however all settings have been adjusted to focus on performance as opposed to image quality. The elements are reduced to 2734 static elements and 270 dynamic elements. For gather radiosity this reduced even further to 774 and 90 elements respectively. Cross redistribution radiosity uses 24 elements on the rotating box. This scene allows for a better performance comparison with competitive techniques which produces lower quality global illumination.

5.2 Visual Quality

To evaluate the qualitative results of the implemented adaptations, generated images are compared with a reference image. The reference image is generated using the gather radiosity adaptation with very high quality settings. Gather radiosity is suitable for generating reference images as it is non-incremental and it scales naturally to very high quality settings. The implementation for gather radiosity shares no radiosity related code with the other implementations.

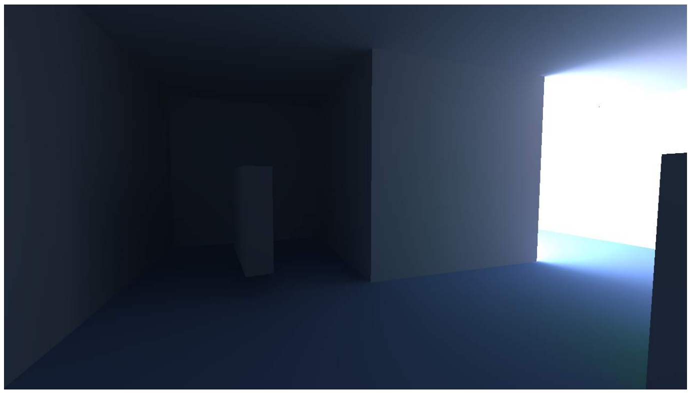
Image capturing The examined frame was captured after several frames of time-independent motion. The gather and progressive refinement adaptations restart their global illumination solution each frame, while the incremental radiosity and cross redistribution radiosity algorithms modify the previous solution each frame. The first solution of the incremental and cross redistribution adaptations is generated using progressive refinement radiosity.

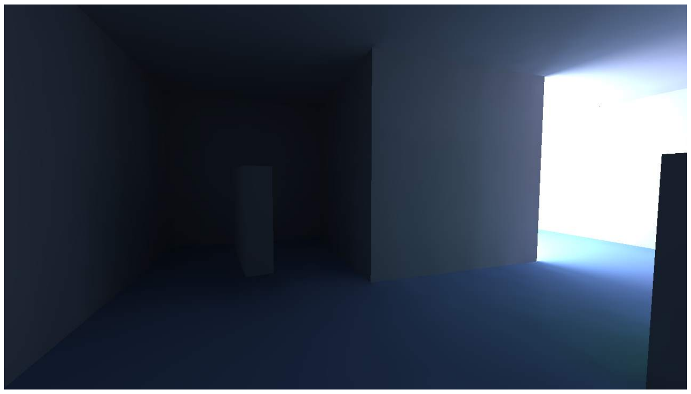
Image quality metric It is non-trivial to create a good image quality metric for comparison of global illumination algorithms. The metric should correctly state the quantity of difference observed by a human spectator. Čadík et al. suggest that all existing image quality metrics are unreliable [4]. A perceptual user study is the best method of quality evaluation, but that lies outside the scope of this thesis. Instead, the results are shown in Figure 5.2.1, 5.2.2, 5.2.3 and 5.2.4 for the reader to observe the differences.

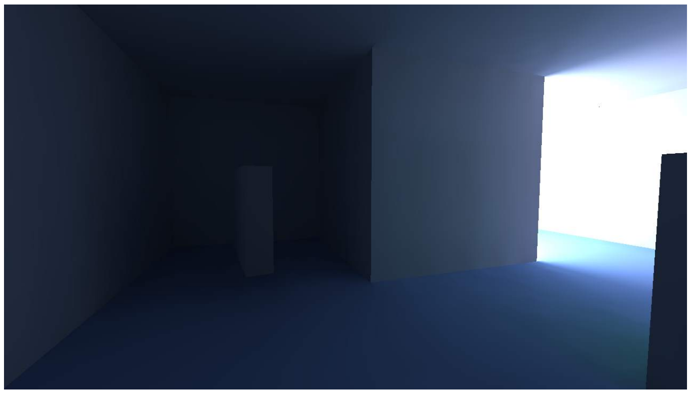
An image quality metric is interesting nevertheless as the numbers indicate the order of magnitude of the objective differences. For this purpose, the normalized root mean squared error of the images in Figures 5.2.1 to 5.2.4 is listed in Table 5.1.

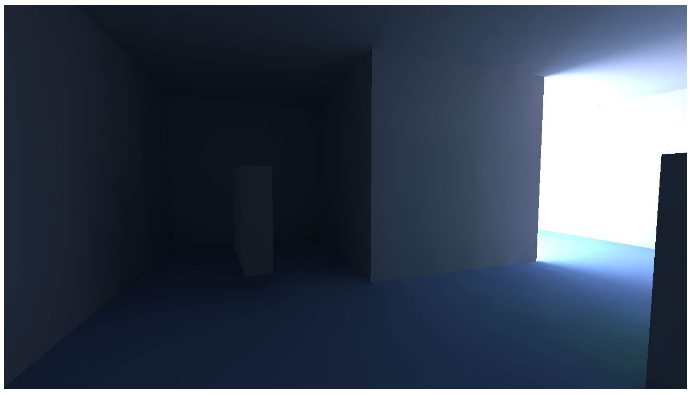
	Indirect light	Cornell, large	Cornell, small	Low quality
Gather	0.0522469	0.00225834	0.0028034	0.0110786
Progressive refinement	0.0631117	0.00675699	0.0098088	0.0149107
Incremental	0.0472801	0.00741281	0.0106912	0.0197874
Cross redistribution	0.0579452	0.00694691	0.0102216	0.0147345

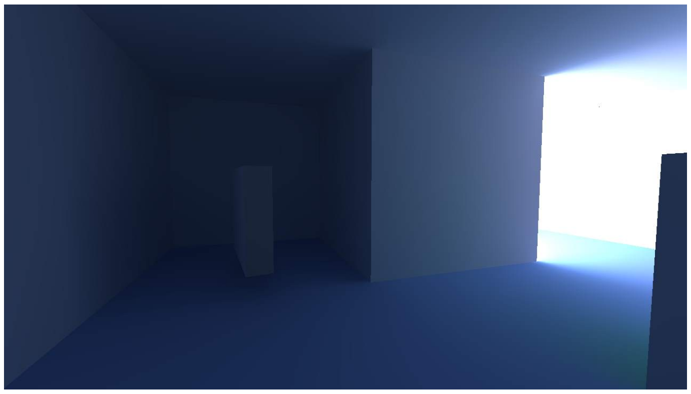
Table 5.1: The normalized root-mean-square error of the images in Figures 5.2.1 to 5.2.4 compared to the respective reference image. It should be noted that the image quality metric does not accurately represent the *perceived* differences.


(a) Gather radiosity


(b) Progressive refinement radiosity


(c) Incremental radiosity


(d) Cross redistribution radiosity


(e) Reference using high quality gather radiosity

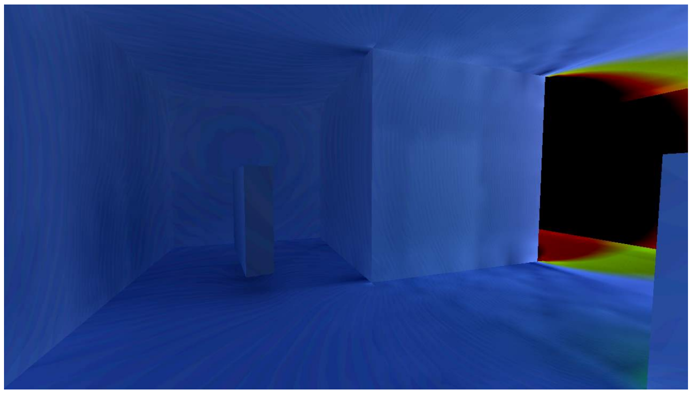
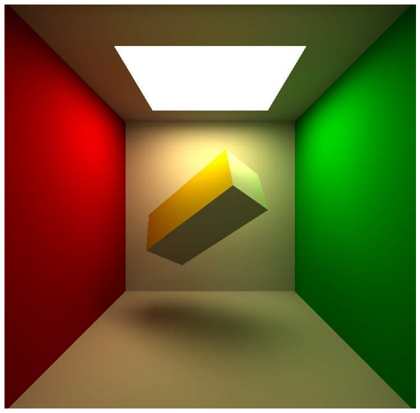
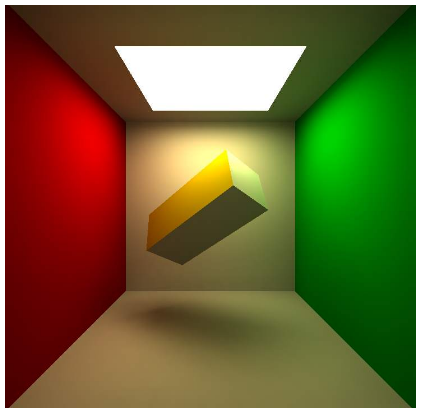
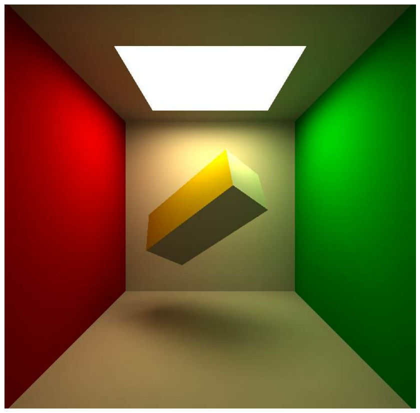
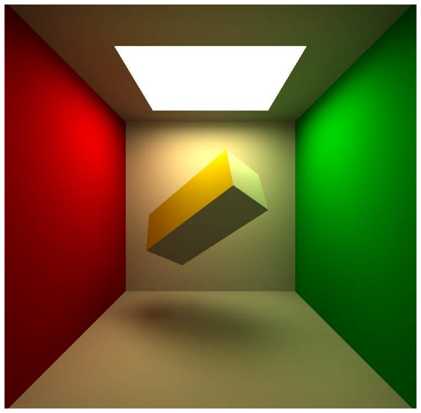

(f) Difference image between cross redistribution radiosity and the reference. The image is multiplied by 5 to emphasize the differences.

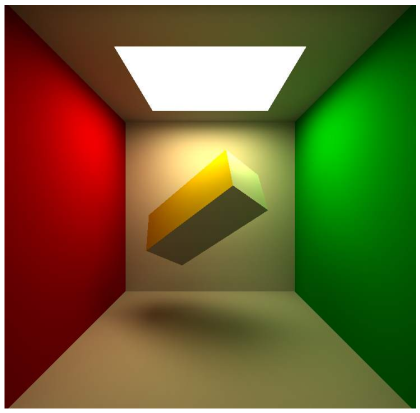
Figure 5.2.1: Screen captures using the indirect light scene.


(a) Gather radiosity


(b) Progressive refinement radiosity


(c) Incremental radiosity


(d) Cross redistribution radiosity


(e) Reference using high quality gather radiosity

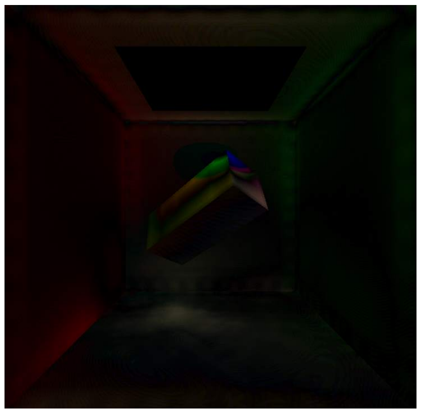
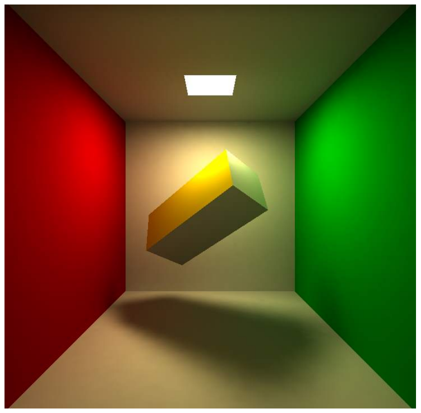
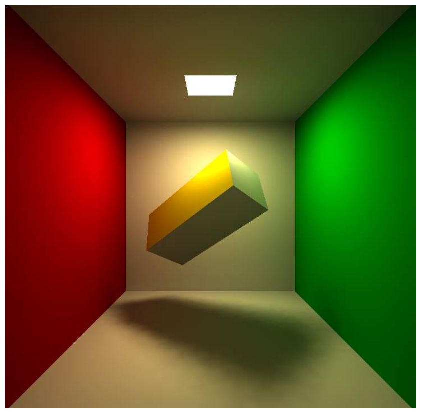
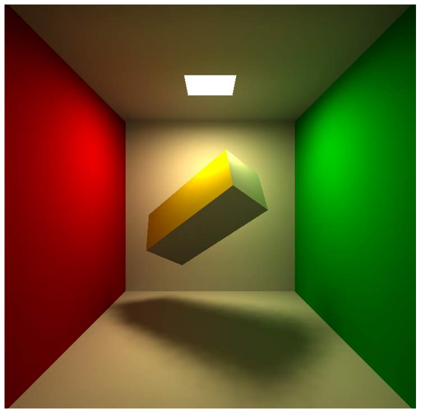
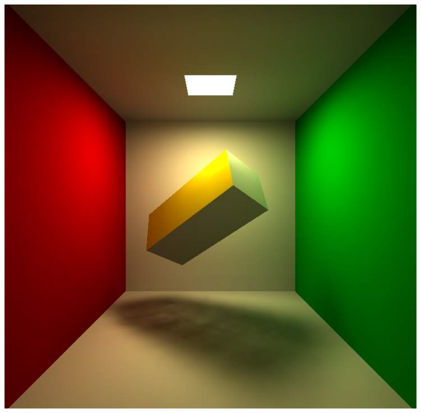

(f) Difference image between cross redistribution radiosity and the reference. The image is multiplied by 5 to emphasize the differences.

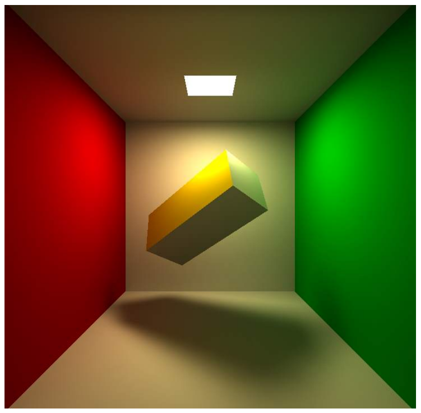
Figure 5.2.2: Screen captures using the Cornell box scene with a large ceiling light source.


(a) Gather radiosity


(b) Progressive refinement radiosity


(c) Incremental radiosity


(d) Cross redistribution radiosity


(e) Reference using high quality gather radiosity

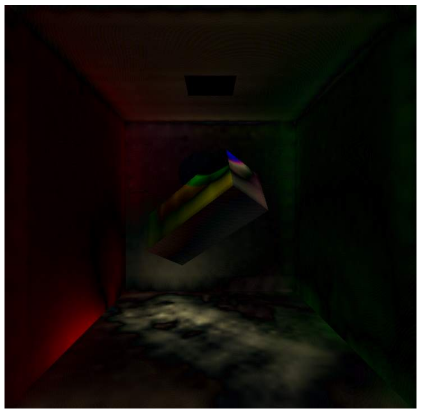
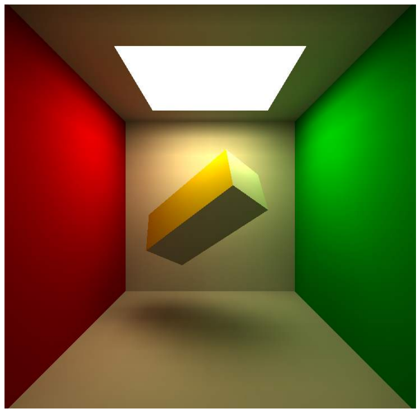
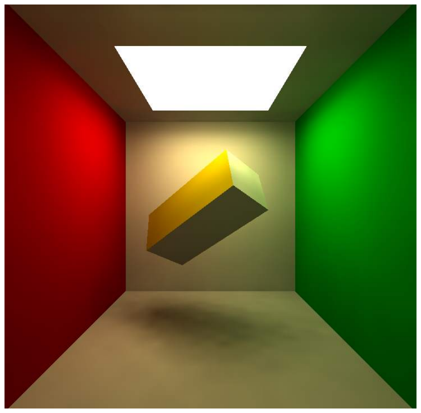
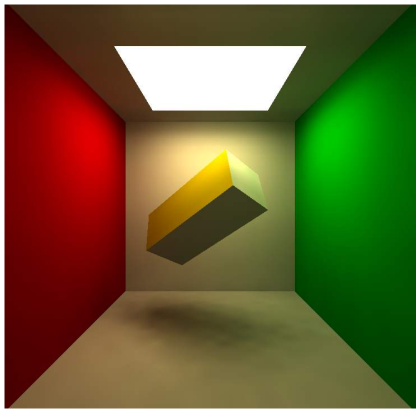
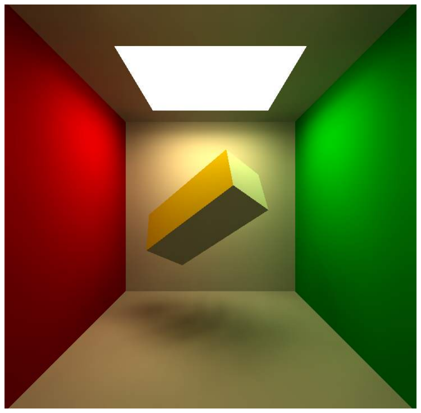

(f) Difference image between cross redistribution radiosity and the reference. The image is multiplied by 5 to emphasize the differences.

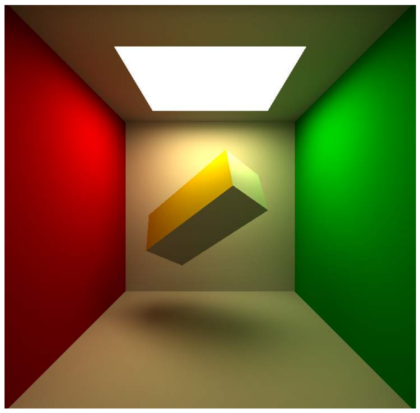
Figure 5.2.3: Screen captures using the Cornell box scene with a small ceiling light source.


(a) Gather radiosity


(b) Progressive refinement radiosity


(c) Incremental radiosity


(d) Cross redistribution radiosity


(e) Reference using high quality gather radiosity

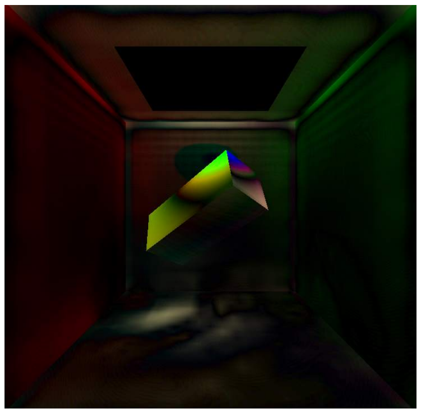

(f) Difference image between cross redistribution radiosity and the reference. The image is multiplied by 5 to emphasize the differences.

Figure 5.2.4: Screen captures using the Cornell box scene with low quality settings.

5.2.1 Visual Analysis

The observable visual differences in Figures 5.2.1 to 5.2.4 will be analyzed in this Section. The objectively measured differences presented in Table 5.1 coarsely correspond to the observable differences.

General observations will be discussed first, followed by differences specific to a scene.

General observations The shadows of the progressive refinement and incremental adaptations are slightly smaller than the reference. This is caused by the visibility detection in the progressive algorithm (line 31 in Algorithm 4.2). Instead of taking a single sample, the four nearest pixels are investigated. This decreases the rate of false negatives, but increases the number of false positives resulting in smaller shadows. The shadow generated by cross projection in the cross redistribution algorithm does not suffer from this.

The visual result of the progressive refinement and incremental approaches are observably identical in the Cornell scenes as stated by the theory. The differences reported by the image quality metric in Table 5.1 can be attributed to minor variations in brightness. The exception is the indirect light scene, which is discussed below.

Indirect light Differences in brightness are clearly observable in this scene and occur for all techniques. The differences are triggered by the very high number of light bounces in this scene. For each technique an explanation is given.

- The gather radiosity adaptation is limited at a number of iterations to keep the execution time within bounds, but it is not enough for convergence.
- In the progressive refinement algorithm, the solution dives below the convergence threshold for shooters.
- The incremental radiosity algorithm performs better as it always executes a progressive refinement iteration on redistribution, regardless of the convergence. However, it has not converged to the reference. While this does improve in subsequent iterations, parity with the reference will not be achieved due to float precision issues.
- The cross redistribution algorithm results in a brightness similar to the brightness of the progressive adaptation. This is obvious as cross redistribution radiosity only modifies the solution. If the reference solution would have been used as the initial solution, cross redistribution radiosity would retain its brightness.

An additional difference observed in this scene is that the surfaces are not smooth for all techniques except gather radiosity. This is caused by the shooter subdivision. Using smaller shooters would solve this problem.

Cornell box with large light source In general, the images are remarkably similar to the reference. The shadow in the cross redistribution radiosity solution is slightly brighter than the other shadows, although this is difficult to observe. This might be caused by the cross projection sampling. The blurring of the back hemicube causes the intensity of the light source to be distributed over a larger area.

Cornell box with small light source The light source is smaller, but also more intense than the large light source. This requires more accuracy in the computation.

- The gather radiosity algorithm has a very subtle noise at the shadow edges. An increase in hemicube resolution would resolve this.

- The noise for the progressive and incremental results are caused by insufficient shooter subdivision.
- The most noticeable artifacts occur in cross redistribution radiosity. The undersampling in the cross projection causes noise in the internals of the shadow. In contrast, the borders of the shadows are less noisy.

Low quality Cornell box The low quality scene shows the degradation of each technique when optimized for performance. Gather radiosity uses as little patches as possible, resulting in a blocky appearance. Noise is increased in progressive and incremental radiosity due to the decrease in subdivision of shooters. The cross redistribution radiosity algorithm also results in noise, mainly caused by the reduction of patches on the dynamic object.

5.2.2 Artifacts

The artifacts produced by cross redistribution radiosity are caused by undersampling in the cross projection function. These artifacts are visible in the low quality scene. Additionally, these artifacts occur during contact shadows as visible in Figure 5.2.5. The white halo directly at the contact points is not produced by undersampling, but by the lack of support for the intersection of surfaces.

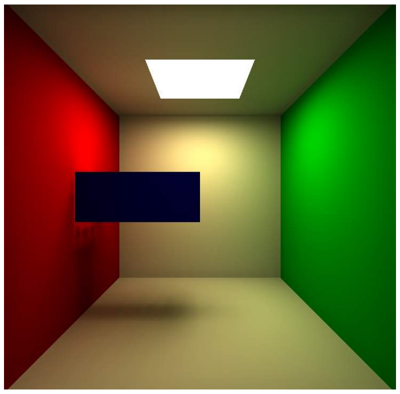


Figure 5.2.5: Contact shadows produce artifacts in the current implementation of cross redistribution radiosity. The white halo at the contact point is not caused by undersampling.

5.3 Execution Time

The average total frame time will be measured using the same scenes and settings used for the visual analysis. The number of samples taken depends on the execution time as less variance is expected for longer execution times. Between 4 and 99 frames are timed. The first 1 to 3 frames are ignored because they produce non representable results due to the driver still compiling shaders and optimizing for the workload.

Hardware The benchmarks are executed on different configurations of graphics hardware. This allows for better comparison with other techniques, it shows the scalability of the techniques and it provides additional data for the performance analysis. The first graphics card, the NVidia GeForce GTX 670 is from 2012 and falls in the high end spectrum. It processes 2459 GFLOPS (billion floating point operations per second) and its memory bandwidth is rated at 192GB/s. The AMD Radeon HD 5770 is a mid-range GPU introduced in 2009 and is specified to have 1360 GFLOPS and a bandwidth of 76.8GB/s. The third graphics card used is the mid-range NVidia GeForce GTX

260 from 2008, rated at approximately 805 GFLOPS with a bandwidth of 112GB/s. The GTX 260 is limited to version 10.0 of the DirectX API, while the other two graphics cards can use all features available in DirectX 11.

The CPUs used in the benchmarks are not relevant as all techniques have their performance bottleneck on the GPU.

Vertex processing The current implementation of the techniques is limited to using quads with textures as radiosity storage. If the techniques were to be implemented using vertex based radiosity storage, complex geometry can be used which increases the vertex count. To investigate the performance implications of a higher vertex usage, additional benchmarks are performed. These benchmarks use subdivided geometry of the same scenes previously used. Visually, the result is identical, but much more vertices are processed. The benchmarks are shown in Table 5.3, where the number of vertices and indices in the subdivided scenes is also listed. These results can be directly compared to the results in 5.2a, which were generated using 80 vertices and 120 indices for the Cornell scenes and 136 vertices with 204 indices for the indirect light scene.

No culling techniques were used, thus all vertices have been processed for all hemicube faces. It should be noted that the vertex processing capability is not necessarily a limitation for the number of vertices in the final displayed scene.

5.3.1 Composition of Execution Time

To analyze which parts of the algorithm dominate the execution time, the most time consuming parts of the algorithms are measured using the GPUPerfStudio 2 profiling tool for graphics hardware. Such measurements are not very accurate as many processes are active simultaneously on the graphics card which can influence each other in many ways. However, the results still provide valuable insight in the distribution of execution time over the parts in the algorithm.

The measurements only consider the hemicube rendering and reconstruction parts of the algorithm. The duration of the other parts are negligible in comparison. The benchmarks have been made in the Cornell box scene with the large light source.

Gather radiosity The algorithm consists of rendering and integrating a hemicube for each element in the scene. Rendering a hemicube with a top face of 64×64 pixels to a 16bits RGBA floating point render target using $4\times$ MSAA (multisampling anti-aliasing) costs 0.043ms which includes resolving the MSAA render target to a regular render target. Integrating the render target (including multiplication with the delta form factors) using compute shaders has a duration of 0.044ms. The DirectX10.0 compatible mipmap-based integration method takes 0.077ms.

Progressive refinement radiosity The algorithm mainly consists of rendering an ID-filled hemicube repeatedly for each patch and reconstructing the shot radiosity for all elements in the scene. Finding the next shooter is amortized over many shooters and thus is negligible in comparison. Rendering an ID-filled hemicube with a top face of 64×64 pixels to a 32bits single unsigned integer render target costs 0.023ms. The shot radiosity reconstruction is amortized over 10 shooters, taking 0.150ms in total or 0.015ms per shooter. For each element in the reconstruction, two RGBA 32bits floating point structures are written to memory.

Incremental radiosity As detailed in Section 4.4, a redistribution pass renders two ID hemicubes for all patches, executes two reconstruction passes for each element and executes an additional incremental reconstruction pass for the elements in C. The execution time of rendering the hemicube and radiosity reconstruction is identical to progressive refinement radiosity. The time necessary for the additional incremental reconstruction is different as two ID hemicubes must be sampled. Measuring the incremental reconstruction was not possible in our implementation, but its time should not be significantly above regular reconstruction.

Cross redistribution radiosity Reconstruction is only executed for all patches on the dynamic object. For each patch, four hemicubes are rendered. One regular ID hemicube, one combined ID and radiosity hemicube and two low resolution hemicubes on the backside. One hemicube is integrated, one regular reconstruction pass is executed for the elements on the dynamic object and finally a cross projection reconstruction pass is performed for the static elements.
The regular ID hemicube is equal to progressive radiosity and thus takes 0.023ms. The combined ID and radiosity hemicube renders using the combined configuration used in gather and progressive radiosity (with the exception that MSAA is disabled) and takes 0.029ms to render. The backside hemicubes are rendered using a top face resolution of 16×16 pixels to a 16bits RGBA floating point render target using $4\times$ MSAA. Each hemicube costs 0.058ms to render, which includes the MSAA resolve for both the color render target and the depth buffer.
The cost of the regular reconstruction pass is negligible as it is only executed for the dynamic elements. The cross projection reconstruction pass costs 1.548ms when amortized over 9 patches, or 0.172ms for a single patch.

5.3.2 Performance Analysis

The results in Table 5.2, Table 5.3 and Section 5.3.1 are used to analyze the algorithms.

General observations The general timing data in Table 5.2 shows a clear ranking in execution speed. The brute-force approach of gather radiosity takes the longest time. Due to substructuring, progressive refinement radiosity performs significantly better. The redistribution of incremental radiosity reduces the execution speed further in all cases. Finally the reduced number of hemicubes required in cross redistribution radiosity clearly performs best in all scenes on all hardware.

Equalities in results As the number of elements is equal in the large and small light source variant of the Cornell box, the performance of both scenes is also nearly equal in all measured scenarios. Similarly, as the number of patches on the dynamic object in the Cornell box is equal regardless of the light source size, cross redistribution radiosity also performs equal in those two scenes in all benchmarks.

Incremental versus progressive refinement The indirect scene shows a large difference in performance between progressive refinement radiosity and incremental radiosity, while the difference is small in the Cornell scene. This is caused by the number of bounces which are required for convergence. The indirect scene requires many bounces to converge, but incremental radiosity only redistributes a few bounces as the moving object only influences the light bounces at the end of the light paths. In the Cornell scenes only a few bounces are required for convergence. As the single redistribution pass in incremental radiosity has the same cost as 2 to 3 shots for each patch in progressive refinement, incremental radiosity is less beneficial in low bounce circumstances.

Bottleneck analysis The AMD Radeon HD 5770 and NVidia GeForce GTX 260 graphics cards have interesting contrasting properties. The 5770 has nearly twice the arithmetic compute power of the 260, but the 260 has nearly twice the memory bandwidth of the 5770. For gather radiosity, the 5770 has a small advantage which suggests that the gather adaptation has a small arithmetic bottleneck (although the bottleneck could also lie somewhere else).

In the case of progressive refinement and incremental radiosity, the situation is different. The 260 has a large advantage over the 5770, suggesting that the algorithms are fully bandwidth limited. The difference is about equal to the difference in bandwidth between the two graphics cards. The

	Indirect light	Cornell, large	Cornell, small	Low quality
Gather	4,757	2,437	2,440	150
Progressive refinement	265	116	87	49
Incremental	52	108	70	14
Cross redistribution	16	24	25	7

(a) Using the NVidia GeForce GTX 670

	Indirect light	Cornell, large	Cornell, small	Low quality
Gather	11,587	6,028	6,077	357
Progressive refinement	1,047	387	302	162
Incremental	146	261	173	43
Cross redistribution	63	93	93	20

(b) Using the AMD Radeon HD 5770

	Indirect light	Cornell, large	Cornell, small	Low quality
Gather	13,315	6,899	6,237	412
Progressive refinement	581	214	193	71
Incremental	83	133	118	25
Cross redistribution	37	53	59	13

(c) Using the NVidia GeForce GTX 260

Table 5.2: Performance benchmarks for all scenes and techniques. The unit is milliseconds.

	Indirect light	Cornell, large	Cornell, small	Low quality
Gather	4,939	2,535	2,510	152
Progressive refinement	393	148	125	52
Incremental	54	118	81	16
Cross redistribution	23	39	41	10

(a) Medium subdivision level. The Cornell scenes use 3004 vertices and 15552 indices. The indirect scene uses 2938 vertices and 14316 indices.

	Indirect light	Cornell, large	Cornell, small	Low quality
Gather	26,222	16,142	16,639	884
Progressive refinement	1,218	948	745	175
Incremental	330	864	574	111
Cross redistribution	88	191	195	40

(b) High subdivision level. 23147 vertices and 131886 indices are used in the Cornell scenes. The indirect scene uses 21660 vertices and 120702 indices.

Table 5.3: Benchmark results using a higher vertex load, generated with the NVidia GeForce GTX 670. The unit is milliseconds.

bandwidth bottleneck hypothesis is supported by the theory. Gather radiosity uses 64bits render targets for the hemicubes and the elements, while progressive and incremental radiosity use *two* 128bits render targets for the elements (and a 32 bit render target for the hemicubes). Unfortunately, the precision of the 128bits render targets is necessary in the current adaptation.

Cross redistribution radiosity also shows a clear bandwidth bottleneck, although the difference is slightly smaller than with progressive refinement and incremental radiosity. The smaller difference may be explained by the difference in DirectX API support. As the 260 card only supports DirectX10.0 it cannot use the faster compute-based integration method and perhaps it may be less suited for complicated algorithms in general.

Additional vertex processing In Table 5.3, the effects of a higher vertex load are displayed. The medium subdivision level shown in Table 5.3a shows a small effect on the execution time. Due to the bandwidth bottleneck, the arithmetic power which was previously unused is now employed for vertex processing thus resulting in only a minor increase in execution time. The increase is slightly larger for cross redistribution radiosity, suggesting that the cross projection is more arithmetically demanding.

The high subdivision level benchmarks displayed in Table 5.3 show a large effect on the execution time. The effect is proportional to the number of hemicubes rendered. As cross redistribution radiosity renders the fewest hemicubes per frame, its advantage over the other techniques is increased. This is especially clear for the indirect scene as the proportion of dynamic patches compared to static patches is lower than in the Cornell scenes.

Real-time performance The execution time required for real-time performance is subject of discussion. We consider it to be at most 33ms, as it corresponds to 30 frames per second. The radiosity computation can be decoupled from the actual frame rate, which allows the redistribution time to be significantly higher without the real-time perception being lost.

Empirical experiments show that all scenes rendered with cross redistribution radiosity using the AMD HD 5770 GPU are perceived as real-time. In this regard, the cross redistribution algorithm is real-time on mid-range graphics hardware from 2008 when using a simple scene. For a scene with many vertices, cross redistribution radiosity is real-time on current high end cards if the dynamic object has a low number of patches, such as in the indirect light scene. When adhering to the strict 33ms definition of real-time, a high end graphics card conforms to this mark for low vertex scenes.

Real-time performance is more easily achieved in the low quality scene, however these scenes do not produce the targeted super high quality global illumination solution.

Cross projection cost Cross redistribution radiosity renders significantly less hemicubes, but requires to execute the expensive cross projection function in return. Cross projection redistribution for a single patch costs 0.172ms, as stated in Section 5.3.1. This part includes two redistribution passes which would separately cost 0.015ms each. The cross projection is very costly compared to the other parts of the algorithm. This cost can be attributed to the 16 samples taken, including the accompanying reprojection.

The benchmarks in Table 5.2 clearly show that the cross redistribution algorithm is faster, thus we can conclude that the cross projection cost is well compensated for by the reduction in the number of hemicubes.

Multisampling cost The hemicubes rendered using 4× MSAA are taking significantly longer than the hemicubes rendered without multisampling, as listed in Section 5.3.1. The main cause for this lies in the resolve pass which is required convert the MSAA render target to a regular texture from which can be read. The resolve pass averages 4 samples for each pixel, which is bandwidth limited in an already bandwidth limited algorithm. However if multisampling were not to be used, the cost would move to other parts of the algorithm. For gather radiosity, four times as large hemicubes would need to be used which increases the both the rendering and integration cost significantly. For the backside hemicubes of cross redistribution radiosity, MSAA functions as a fast form of blur which would otherwise need be replaced by an actual blurring pass.

Integration method The compute based integration strategy is much faster than the mipmap based integration implemented for legacy purposes. As stated in Section 5.3.1, compute integration costs 0.044ms while mipmap integration takes 0.077ms. This is caused by the intermediate memory reading and writing by the mipmap approach. For each level in the mipmap chain, the previous level is read, averaged and written back to memory. In contrast, the compute shader reads all data only once and outputs a single value. This reduction in memory access is very beneficial.

5.4 Comparison with Other Techniques

Comparing between global illumination technique is very challenging because of the multitude of different situations. Most related publications omit a comparison altogether or compare their contribution exclusively to very similar techniques. We will compare our cross redistribution radiosity technique with the state of the art in all other major high quality global illumination branches. Approximate real-time global illumination techniques such as light propagation volumes [24] or voxel cone tracing [11] are not considered as they are limited to single bounce global illumination or cannot provide high quality global illumination.

A comparison with non-radiosity methods is particularly difficult. The fundamental differences between the techniques generally result in a usage of test scenes with different properties. For example, techniques using ray tracing can accommodate complex geometry more efficiently due to the logarithmic scaling of hierarchical acceleration structures. However, rebuilding the acceleration structures is costly so dynamic geometry is restricted. Another fundamental difference is that radiosity is less suited for directional reflectance due to its caching properties, but the caching does facilitate redistribution of radiosity and fast rendering of the final scene.

A comparison focused at many bounce indirect illumination is expected to show a clear advantage for cross redistribution radiosity. Unfortunately, other techniques rarely provide scenes demonstrating many bounce global illumination. A conclusive comparison seems therefore impossible. In spite of this, we present a comparison that provides plausible evidence showing that cross redistribution radiosity is faster at producing high quality global illumination for dynamic geometrically simple scenes where the focus lies on many bounce global illumination.

5.4.1 Antiradiance

The antiradiance technique was explained in the related work in Section 2.3.3. The Cornell box scene with a large light source is similar to the scene used in the paper extending antiradiance to fully dynamic scenes by Meyer et al. [30]. Two images of the antiradiance implementation are shown in Figure 5.4.1. The low quality implementation runs at approximately 33ms per frame using the NVidia GeForce GTX 260, while the high quality settings take 63ms.

The high quality antiradiance implementation can be compared to the Cornell box with the large light source, and the low quality settings can be compared to the low quality scene used in this thesis. A comparison between the antiradiance screen captures and the screenshots provided in Figure 5.2.2 and 5.2.4 illustrates a clear qualitative difference in favor of cross redistribution radiosity. The shadow of antiradiance is blurred significantly which removes important detail in the shadow boundaries.

The execution time of the radiosity adaptations presented in this thesis have been evaluated with the same NVidia GTX 260 graphics card in Table 5.2c. Cross redistribution radiosity takes 53ms to render the comparable scene using high quality settings and 13ms with low quality settings,

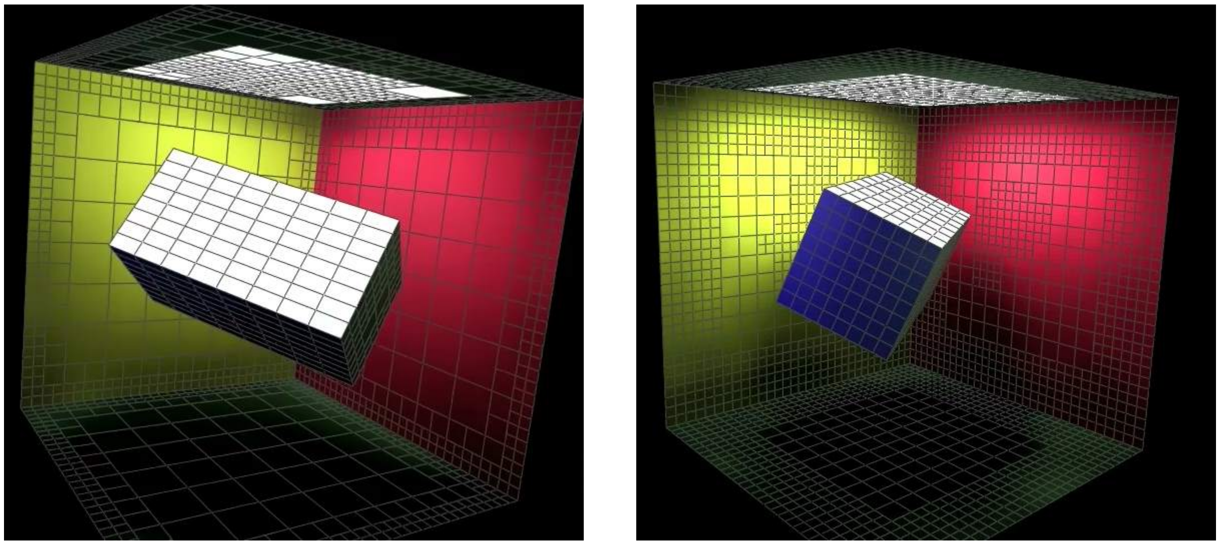


Figure 5.4.1: Antiradiance screen captures from the video supplied with [30]. The left image uses a faster low quality variant while the right image uses slower high quality settings.

both clearly faster than the antiradiance implementation. It is expected that the execution time disadvantage of antiradiance will increase significantly for a scene which requires more bounces. Each additional bounce adds to the execution time because the entire global illumination solution must be regenerated each frame, as opposed to the redistribution of cross redistribution radiosity.

5.4.2 Path Tracing

The commercial Brigade 3 engine is one of the most prominent real-time path tracers. Figure 5.4.2 shows images from a video published October 22nd 2013. The view is directed at the shadowed side of a building in a city.

Two NVidia GeForce GTX Titan graphics cards were used to render the image which are combined capable of delivering 9000 GFLOPS and 576,8 GB/s of bandwidth, much more than the graphics cards used in this thesis. The exact time required to produce a single is not known but it must be close to 33ms. When correcting for the difference in computational power, it is clear that our technique is significantly faster.

The image quality is clearly insufficient due to the prohibitive amount of noise. While post processing filters exist to remove the noise, such as the random parameter filtering method by Sen and Darabi [38], their execution speed lies in the order of minutes. For scenes with a higher number of indirect light bounces, the quality is expected to reduce further because the number of possible paths increases exponentially.

The scene in Figure 5.4.2 is geometrically much more complex than our scenes. Processing such scenes efficiently is one of the strengths of path tracing, as its hierarchical acceleration structure provides logarithmic scaling for geometric complexity in practice. Another advantage of path tracing is the indisputable high quality of the results. However, this can take a very long time to achieve as a potentially enormous amount of samples must be taken to completely remove the noise.

Considering the prohibitive levels of noise and the large amount of computational power used to achieve this, it can be concluded that our cross redistribution radiosity is more efficient at rendering high quality global illumination for simple scenes.

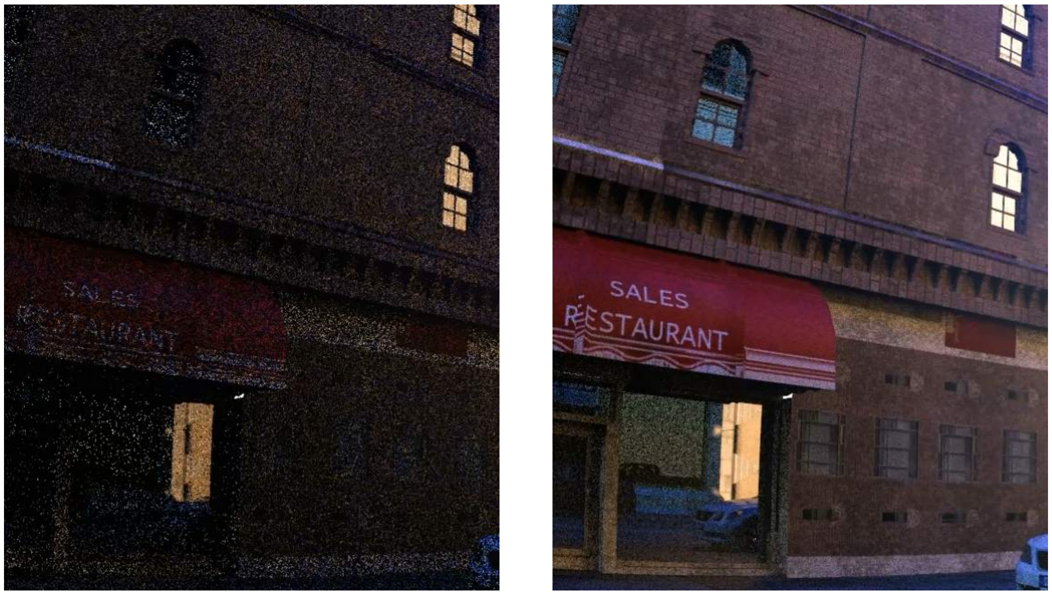


Figure 5.4.2: Screen captures of a video demonstrating the Brigade 3 path tracing engine, originating from http://raytracey.blogspot.com/2013/10/brigade-3.html. The left image shows the path tracing result produced in one frame. The right image displays the image generated after two seconds.

5.4.3 Photon Mapping

Many photon mapping variants have been developed, creating a spectrum of technique which balances execution time versus image quality. The image space photon mapping technique by McGuire and Luebke was explained in Section 2.1.2 and will be used for comparison. The technique has a similar performance compared to cross redistribution radiosity [29]. The execution time reported for the Cornell scene depicted in Figure 5.4.3 varies between 26ms and 97ms depending on the quality settings. The results were generated using the NVidia GeForce GTX 280 graphics card. The radiance estimation phase has been improved by Mara et al. in [27] with about 50 percent. Taking this improvement into account in the results generated by McGuire and Luebke would decrease the execution time to 28ms and 86ms respectively.

Our measurements in Table 5.2 show times between 13ms and 53ms for the GTX 260 and 7ms and 24ms for the GTX 670. The capabilities of the GTX 280 used for the image space photon mapping measurements are between the GTX 260 and GTX 670, so it can be concluded that cross redistribution radiosity is faster.

The quality of image space photon mapping is significantly below the quality of cross redis-

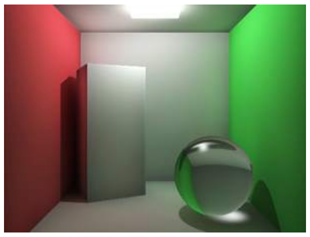


Figure 5.4.3: Cornell scene rendered by image space photon mapping [29]. Direct light is not rendered using photon mapping.

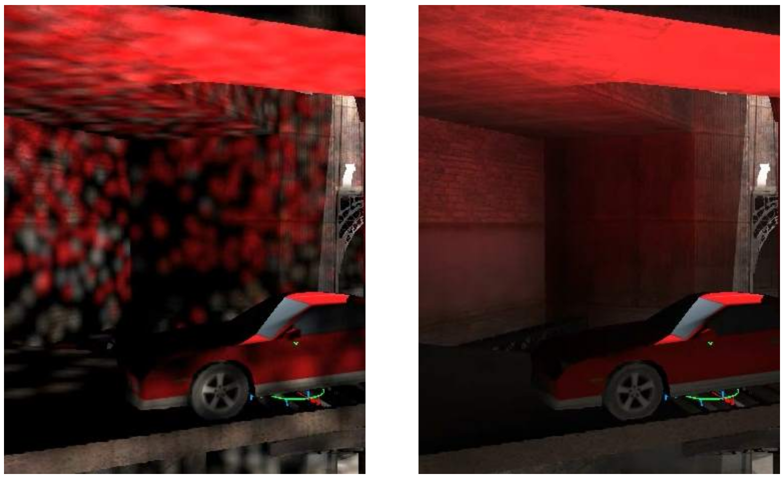


Figure 5.4.4: Photon blurring used in image space photon mapping. Captures taken from the video accompanying [29]. The left image shows the generated photons, which are blurred extremely to produce the smooth result in the right image.

tribution radiosity. The exclusion of the direct light computation from the algorithm is the first indication, which is commonly done when the accuracy of global illumination algorithms is insufficient. The second sign can be found by the fact that the indirect illumination computation is done at significantly lower resolutions (9 to 36 times less pixels). Visually examining the procedure used by image space photon mapping provides the most convincing evidence. Figure 5.4.4 shows the extreme amounts of blurring which must be used to generate a smooth result, thus destroying a lot of detail in the indirect illumination.

Alternative photon mapping techniques perform a final gathering phase which is very expensive and thus not feasible at real-time frame rates. Increasing the number of indirect bounces reduces performance further as potentially exponentially more photons must be shot to achieve the same visual accuracy.

5.4.4 Instant Radiosity

Many variants of instant radiosity compromise the quality of the rendered solution. As explained in Section 2.1.4, the technique by Novák et al. is one of the best qualitative instant radiosity algorithms while still targeting near real-time execution time. Still, the generated quality has significant limitations. The direct illumination is computed using regular shadow maps which excludes the use of area light sources. Furthermore due to the use of imperfect shadow maps, indirect shadows have limited accuracy as depicted in Figure 5.4.5. It should be noted that contact shadows as displayed in Figure 5.4.5 are not handled correctly by the current implementation of cross redistribution radiosity due to undersampling.

Novák et al. evaluate the execution speed using an AMD Radeon HD 5870 graphics card which has a computational power similar to the GTX 670 used in our benchmarks. The instant radiosity technique uses between 61ms and 106ms to generate the global illumination solution for the Cornell scene in Figure 5.4.3. This is significantly slower than the execution time for cross redistribution radiosity (7ms and 24ms respectively) as listed in Table 5.2.

In scenes where more indirect bounces are necessary, the number of VPLs required for the same level of accuracy is expected to increase exponentially. Therefore, many bounce indirect lighting scenes are expected to increase the advantage of cross redistribution radiosity.

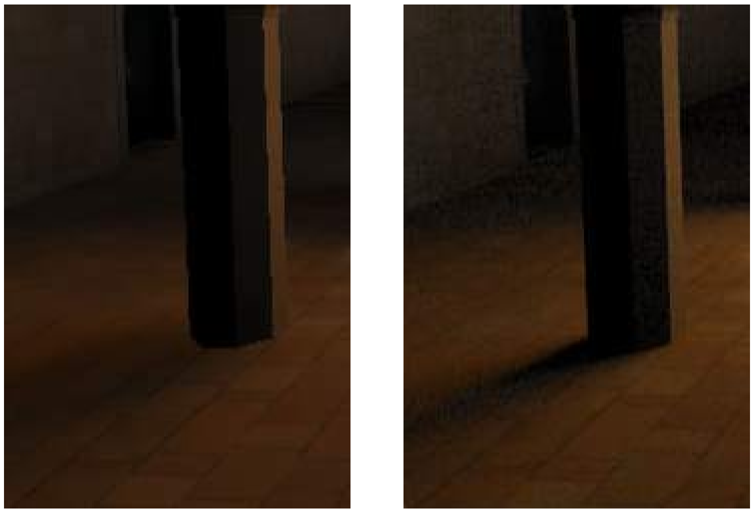


Figure 5.4.5: Loss of shadow detail using imperfect shadow maps, images extracted taken from [36]. The left image is rendered using imperfect shadow maps, the right image is a path traced reference.

Figure 5.4.6: Cornell scene used in the instant radiosity paper by Novák et al. [31].

5.4.5 Comparison Overview

Table 5.4 features an overview of several relevant characteristics of the compared techniques and approximate normalized execution time measurements. Although a perfect comparison between our cross redistribution technique and its main competitors is impossible to make, this section has given an indication of the differences. It has been made plausible that cross redistribution radiosity outperforms the competition when generating high quality global illumination for simple scenes.

	Cross redistribution radiosity	Antiradiance by Meyer et al.	Path tracing	Image space photon mapping	Instant radiosity by Novák
Includes direct illumination	X	X	X		
View independent	X	X			
Caustics		X	X	X	
Scaling to more bounces	Constant / Linear	Linear	Exponential	Exponential	Exponential
Main artifacts	Undersampling noise	Low resolution	High frequency noise	Low resolution	Poor shadow accuracy
Execution time	16ms	48ms	119ms	41ms	84ms

Table 5.4: Comparative table for the examined techniques. The execution time is an approximation based on the average of the high and low quality settings and normalized to a GTX 670 graphics card. A qualitative judgment is left for the reader to make.

Chapter 6

Conclusion

The objective of this thesis is to introduce a new radiosity style global illumination technique which is able to provide high quality global illumination in real-time, under several restrictions such as simple scenes.

Three hardware accelerated radiosity adaptations have been introduced: an improved version of progressive refinement for graphics hardware, a novel incremental radiosity adaptation and the new cross redistribution radiosity. For the latter a supporting theoretical framework has been devised regarding the redistribution of radiosity.

Measurements have shown that the quality for all techniques is high, but not in all situations. The most noticeable artifacts are a consequence of undersampling in the cross redistribution radiosity algorithm. Benchmarks show that the execution time for cross redistribution radiosity is well within the real-time constraints. A comparison with competing techniques proves to be favorable for our novel cross redistribution radiosity.

The radiosity family of global illumination algorithms has acquired its first descendant technique which combines high image quality with real-time results: our cross redistribution radiosity.

6.1 Cross Redistribution Radiosity Evaluation

The main advantages and disadvantages of our novel cross redistribution technique are summarized below.

Advantages

- Redistribution of radiosity requires less computational work than the generation of the solution, therefore contributing significantly to the low execution time.
- Cross projection removes the hemicubes rendered for static patches, only requiring them for dynamic patches. This reduces the number of rendered hemicubes significantly for scenes with relatively few dynamic patches.
- The generated radiosity solution is view independent and can be displayed quickly. This allows for the amortization of the global illumination computation over several frames.
- The number of simulated bounces has a linear effect on the algorithm, as opposed to the exponential effect of most other techniques. Additionally, the number of bounces from the light source to a dynamic patch does not influence execution time.

Disadvantages

- Adaptive refinement of the global illumination solution is complicated due to the caching of illumination on the surface.
- The radiosity storage makes the inclusion of specular reflection and refraction more complex.
- Vertex density of the scene is restricted as many hemicubes must be rendered.
- Undersampling in cross projection yields artifacts for small sharp light sources and for contact points with dynamic geometry. These issues are expected to be solvable as explained in Section 6.2.

6.2 Future Work

Suggestions for improvement focus on cross redistribution radiosity, as it is expected to have most potential. Additionally, some variants for regular incremental radiosity are proposed. Many possible improvements are offered, suggesting much potential in this area.

Cross redistribution radiosity qualitative improvements The main qualitative issue is the undersampling in the cross projection. It is expected that a pre-integration scheme such as mipmapping for the back-hemicubes will avoid undersampling. The appropriate mipmap level must be selected depending on the distance between the element and the shooter. This might allow the sample count to be reduced, which would increase performance.

Restrictions on the types of scenes are currently limiting the usage scenario for cross redistribution radiosity. The constraint to quad-based radiosity should be alleviated by modifying the technique to vertex-based radiosity. This enables organic shapes to be included efficiently. Furthermore, the small scene restriction could be relaxed by employing hierarchical grouping structures for both shooters and receivers, similar to clustered hierarchical radiosity [41]. Several levels of detail may be used to reduce the vertex processing load during hemicube rendering. This modification would not make it a hierarchical radiosity variant as that technique maintains a collection of links which is non-trivial to adapt. Instead, the relevant links would be regenerated when necessary. An additional benefit of link regeneration is that the hierarchical detail can depend on the amount of radiosity transferred.

The incremental cross projection discussed in Section 4.5.2 was not accurate enough, therefore it accumulated error. An implementation which searches for shooters using a variant of ray tracing would offer perfect accuracy and therefore could be implemented incrementally.

As explained in the analysis of non-incremental cross projection in Section 4.5.2, artifacts may accumulate due to intermediate progressive refinement passes. If the hemicubes for the previous redistribution frames are stored, these can be used so that no error will accumulate. Additionally, fewer hemicubes have to be rendered each frame. The disadvantage is an increase in memory usage.

Support for coarse specular reflection could be implemented by using a low order hemispherical basis to store the directional radiosity. Very few coefficients should be used to limit the increase in memory bandwidth.

Cross redistribution radiosity performance improvements The main performance bottleneck is the bandwidth usage of the radiosity algorithms. This is presumably caused by the usage of full-precision floating point render targets for the elements. If these would be reduced to a 16bit floating point format per element, the bandwidth usage would be reduced significantly. To retain the required accuracy, increments in radiosity should be as large as possible. This can be realized by creating larger shooters when low amounts of radiosity are being shot.

Dynamic patches which move a very short distance between frames will render two very similar hemicubes for the before and after scene. By using the reprojection strategy to generate the new hemicube from the old hemicube, the number of rendered hemicubes can be halved. Dynamic patches appearing on the reprojected hemicubes should be excluded from the reprojection and rendered again.

Several implementation modifications might increase performance. The used texture arrays might be less efficient than regular texture atlases. Rendering of multiple hemicubes may be combined more efficiently by storing all hemicubes in the same render target. The next shooter selection, as part of the progressive refinement phases, can be optimized by using compute shaders with a parallel reduction technique. When graphics hardware gains support for draw calls dispatched by the GPU, the readback of the selected shooter may be avoided.

Adaptive shooter subdivision generates many shooters for direct light sources. Adaptive soft shadow techniques may be employed to measure partial occlusion, which would significantly reduce the level of subdivision required.

Incremental radiosity variants The incremental radiosity adaptation for graphics hardware in this thesis uses reshooting for dynamic patches. If reshooting would be employed for the static patches, only a single hemicube must be rendered for each patch and a single pass over all elements is required. The static patches would reshoot their radiosity first, after which the dynamic patches execute a regular progressive refinement shot. The disadvantage of this approach is that all patches must reshoot before other modifications can be made to the solution.

An alternative implementation without any reshooting would adhere more closely to the original incremental radiosity technique. When the differential form factor is calculated explicitly for all elements, not all patches have to shoot radiosity before an additional modification can be made. The only restriction is that each shooter handles all modifications in order. If this approach is combined with a good heuristic for shooter prioritization, potentially very few patches have to shoot to reach perceptual convergence. If directional radiosity is stored, it can be used to search for shooters in the area were most (un)occlusion is expected to occur.

Bibliography

[1] Appel, A. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference* (New York, NY, USA, 1968), AFIPS '68 (Spring), ACM, pp. 37–45.

[2] Arvo, J. Backward ray tracing. In *ACM SIGGRAPH 1986 Course Notes - Developments in Ray Tracing* (1986), pp. 259–263.

[3] Bikker, J. *Ray Tracing for Real-time Games.* PhD thesis, TU Delft, Nov. 2012.

[4] Čadík, M., Herzog, R., Mantiuk, R., Myszkowski, K., and Seidel, H.-P. New measurements reveal weaknesses of image quality metrics in evaluating graphics artifacts. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 147:1–147:10.

[5] Chen, S. E. Incremental radiosity: an extension of progressive radiosity to an interactive image synthesis system. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1990), SIGGRAPH '90, ACM, pp. 135–144.

[6] Cohen, M., Greenberg, D., Immel, D., and Brock, P. An efficient radiosity approach for realistic image synthesis. *IEEE Comput. Graph. Appl. 6* (Mar. 1986), 26–35.

[7] Cohen, M. F., Chen, S. E., Wallace, J. R., and Greenberg, D. P. A progressive refinement approach to fast radiosity image generation. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1988), SIGGRAPH '88, ACM, pp. 75–84.

[8] Cohen, M. F., and Greenberg, D. P. The hemi-cube: a radiosity solution for complex environments. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1985), SIGGRAPH '85, ACM, pp. 31–40.

[9] Cook, R. L., Porter, T., and Carpenter, L. Distributed ray tracing. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1984), SIGGRAPH '84, ACM, pp. 137–145.

[10] Coombe, G., Harris, M. J., and Lastra, A. Radiosity on graphics hardware. In *Proceedings of Graphics Interface 2004* (2004), Canadian Human-Computer Communications Society, pp. 161–168.

[11] Crassin, C., Neyret, F., Sainz, M., Green, S., and Eisemann, E. Interactive indirect illumination using voxel cone tracing. *Proceedings of Pacific Graphics 2011 30* (Sep. 2011), 1921–1930.

[12] Dachsbacher, C., and Stamminger, M. Reflective shadow maps. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2005), I3D '05, ACM, pp. 203–231.

[13] Dachsbacher, C., Stamminger, M., Drettakis, G., and Durand, F. Implicit visibility and antiradiance for interactive global illumination. In *ACM SIGGRAPH 2007 papers* (New York, NY, USA, 2007), SIGGRAPH '07, ACM.

[14] Damez, C., Dmitriev, K., and Myszkowski, K. Global illumination for interactive applications and high-quality animations. In *Eurographics 2002: State of the Art Reports* (Saarbrücken, Germany, 2002), D. W. Fellner and R. Scopigno, Eds., Eurographics, pp. 1–24.

[15] Dmitriev, K., Brabec, S., Myszkowski, K., and Seidel, H.-P. Interactive global illumination using selective photon tracing. In *Proceedings of the 13th Eurographics Workshop on Rendering* (Aire-la-Ville, Switzerland, Switzerland, 2002), EGRW '02, Eurographics Association, pp. 25–36.

[16] George, D. W., Sillion, F. X., and Greenberg, D. P. Radiosity redistribution for dynamic environments. *IEEE Comput. Graph. Appl. 10*, 4 (July 1990), 26–34.

[17] Goral, C. M., Torrance, K. E., Greenberg, D. P., and Battaile, B. Modeling the interaction of light between diffuse surfaces. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1984), SIGGRAPH '84, ACM, pp. 213–222.

[18] Hanrahan, P., Salzman, D., and Aupperle, L. A rapid hierarchical radiosity algorithm. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1991), SIGGRAPH '91, ACM, pp. 197–206.

[19] Heckbert, P. S. Adaptive radiosity textures for bidirectional ray tracing. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1990), SIGGRAPH '90, ACM, pp. 145–154.

[20] Immel, D. S., Cohen, M. F., and Greenberg, D. P. A radiosity method for non-diffuse environments. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1986), SIGGRAPH '86, ACM, pp. 133–142.

[21] Jensen, H. W. Global illumination using photon maps. In *Proceedings of the eurographics workshop on Rendering techniques '96* (London, UK, 1996), Springer-Verlag, pp. 21–30.

[22] Jensen, H. W., and Christensen, N. J. Photon maps in bidirectional monte carlo ray tracing of complex objects. *Computers & Graphics 19*, 2 (1995), 215–224.

[23] Kajiya, J. T. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1986), SIGGRAPH '86, ACM, pp. 143–150.

[24] Kaplanyan, A., and Dachsbacher, C. Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2010), I3D '10, ACM, pp. 99–107.

[25] Keller, A. Instant radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 49–56.

[26] Lafortune, E. P., and Willems, Y. D. Bi-directional path tracing. In *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques* (1993), Compugraphics '93, pp. 145–153.

[27] Mara, M., Luebke, D., and McGuire, M. Toward practical real-time photon mapping: Efficient gpu density estimation. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2013), I3D '13, ACM, pp. 71–78.

[28] Martin, S., and Einarsson, P. A real-time radiosity architecture for video games. In *SIGGRAPH 2010 Advances in Real-Time Rendering in 3D Graphics and Games Course* (July 2010).

[29] McGuire, M., and Luebke, D. Hardware-accelerated global illumination by image space photon mapping. In *Proceedings of the Conference on High Performance Graphics 2009* (New York, NY, USA, 2009), HPG '09, ACM, pp. 77–89.

[30] Meyer, Q., Eisenacher, C., Stamminger, M., and Dachsbacher, C. Data-parallel hierarchical link creation for radiosity. In *Proceedings of the 9th Eurographics conference on Parallel Graphics and Visualization* (Aire-la-Ville, Switzerland, 2009), EG PGV'09, Eurographics Association, pp. 65–70.

[31] Novák, J., Engelhardt, T., and Dachsbacher, C. Screen-space bias compensation for interactive high-quality global illumination with virtual point lights. In *Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2011), I3D '11, ACM, pp. 119–124.

[32] Novák, J., Havran, V., and Daschbacher, C. Path regeneration for interactive path tracing. In *The European Association for Computer Graphics 28th Annual Conference: EUROGRAPHICS 2007, short papers* (2010), pp. 61–64.

[33] Puech, C., Sillion, F., and Vedel, C. Improving interaction with radiosity-based lighting simulation programs. *SIGGRAPH Comput. Graph. 24*, 2 (Feb. 1990), 51–57.

[34] Purcell, T. J., Buck, I., Mark, W. R., and Hanrahan, P. Ray tracing on programmable graphics hardware. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 703–712.

[35] Purcell, T. J., Donner, C., Cammarano, M., Jensen, H. W., and Hanrahan, P. Photon mapping on programmable graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (Aire-la-Ville, Switzerland, 2003), HWWS '03, Eurographics Association, pp. 41–50.

[36] Ritschel, T., Grosch, T., Kim, M. H., Seidel, H.-P., Dachsbacher, C., and Kautz, J. Imperfect shadow maps for efficient computation of indirect illumination. In *ACM SIGGRAPH Asia 2008 papers* (New York, NY, USA, 2008), SIGGRAPH Asia '08, ACM, pp. 129:1–129:8.

[37] Rushmeier, H. E., and Torrance, K. E. Extending the radiosity method to include specularly reflecting and translucent materials. *ACM Trans. Graph. 9*, 1 (Jan. 1990), 1–27.

[38] Sen, P., and Darabi, S. On filtering the noise from the random parameters in monte carlo rendering. *ACM Trans. Graph. 31*, 3 (June 2012), 18:1–18:15.

[39] Sillion, F., and Puech, C. A general two-pass method integrating specular and diffuse reflection. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1989), SIGGRAPH '89, ACM, pp. 335–344.

[40] Sillion, F. X., Arvo, J. R., Westin, S. H., and Greenberg, D. P. A global illumination solution for general reflectance distributions. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1991), SIGGRAPH '91, ACM, pp. 187–196.

[41] Smits, B., Arvo, J., and Greenberg, D. A clustering algorithm for radiosity in complex environments. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1994), SIGGRAPH '94, ACM, pp. 435–442.

[42] van Antwerpen, D. Improving simd efficiency for parallel monte carlo light transport on the gpu. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics* (New York, NY, USA, 2011), HPG '11, ACM, pp. 41–50.

[43] Veach, E., and Guibas, L. J. Metropolis light transport. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 65–76.

[44] Wallace, J. R., Cohen, M. F., and Greenberg, D. P. A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1987), SIGGRAPH '87, ACM, pp. 311–320.

[45] Wallace, J. R., Elmquist, K. A., and Haines, E. A. A ray tracing algorithm for progressive radiosity. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1989), SIGGRAPH '89, ACM, pp. 315–324.

[46] Ward, G. J., Rubinstein, F. M., and Clear, R. D. A ray tracing solution for diffuse interreflection. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1988), SIGGRAPH '88, ACM, pp. 85–92.

[47] Zhou, K., Hou, Q., Wang, R., and Guo, B. Real-time kd-tree construction on graphics hardware. In *ACM SIGGRAPH Asia 2008 papers* (New York, NY, USA, 2008), SIGGRAPH Asia '08, ACM, pp. 126:1–126:11.