Forschungsseminar aus Computergraphik und Digitaler Bildverarbeitung (SS2008)
Maurice Termeer, Erald Vuçini, Peter Kohlmann, Daniel Patel, Muhammad Muddassir Malik
(Editors)

Instant Radiosity
An Approach for Real-Time Global Illumination

Martin Kinkelin (0326997) & Christian Liensberger (0107660)

Abstract

We give an overview of Instant Radiosity, an approach for global illumination which has recently become suitable for real-time rendering of complex scenes on PCs exploiting current 3D hardware. We give an introduction to global illumination, present other approaches for the solution of this problem and dig deeper into Instant Radiosity including recent extensions and optimizations. We also provide some of our own ideas for practical implementations.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

Illumination is the general name for a set of algorithms that are used in computer graphics to bring realistic lighting into rendered scenes. The technique is used to make virtual objects look as if they were real objects placed into the real world. Light allows to alter the scene dramatically as it may be used to completely change the look and feel of the scene and its containing objects. Usually, light is used to add depth, shade and even alter the look of an object to make it appear different (e.g. older, coarser, rounder, etc.) as it originally was by only taking its geometrics into consideration.

In computer graphics there are two distinct approaches of creating illumination in a scene: local illumination and global illumination.

1.1. Local illumination

Local illumination is the traditional approach in solving the problem of real looking objects through illumination. It has been one of the first approaches to solve the problem of displaying light on items in a rendered scene, and dates back into 1977 and even before. All the different approaches for local illumination have three things in common:

- They are very fast to compute.
- They are heuristic and mostly incomplete (the approaches do not rely on a valid physical background).
- They are most interested in light that is in the direction of the viewpoint; all other lighting is mostly ignored.

In most of the ad-hoc illumination equations there are three major terms that are respected when calculating the lighting on an object: the ambient term (it allows to create a general background lighting), the diffuse light and the specular light. These three items combined result in most of the cases in the final equation for local illumination:

$$I := ambient + diffuse + specular \tag{1}$$

As can seen from equation 1 the local illumination term is really simple and fast to calculate. This kind of equation is also predestined to be used in pixel shader programs that execute directly on the graphic card.

An example of a model that is based on local illumination is the Blinn-Phong Specular Reflection or Blinn-Phong Shading [Bli77]. Figure 1 shows the three parts of the Phong Shading equation; the Blinn part of the equation is only an addition to the Phong shader and makes the results look more realistic.

1.2. Global illumination

Global illumination extends the ideas of local illumination: in detail that means that the objects will be enlightened by light that is directly coming from the light source itself (also called direct illumination). But, in addition, the objects' surfaces will also take into consideration light that's coming

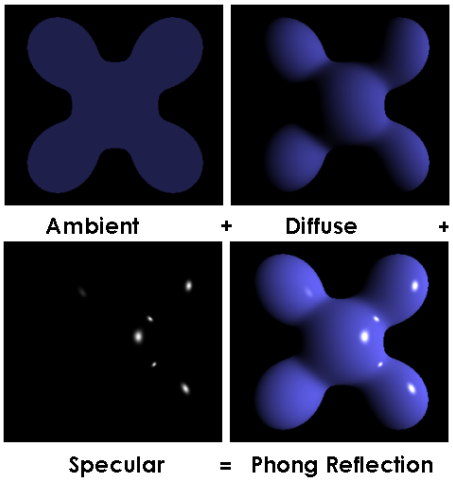

Figure 1: *This image shows the Phong equation. The light source emits white light, the ambient color and the diffuse color are both blue and the specular term is white again. The highlights are very narrow.*

from other object surfaces, when computing their own lighting. These other surfaces are not primary light source by themselves, but they reflect some light that's coming from the primary light source. This behavior is similar to real world scenarios where objects absorb certain parts of the light and reflect others. This technique is called indirect illumination.

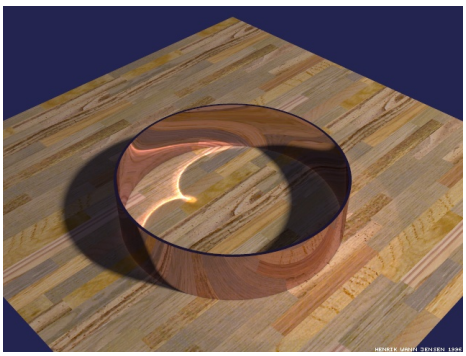

Figure 2: *The metal ring in the image is rendered by using global illumination techniques. Some light that is reflected by the ring is shown at the table.*

Theoretically, shadows, reflections and all other scenarios where the lighting behavior of one surface alters the lighting on another surface are the results of global illumination. In practice, probably due to the history of illumination in computer graphics, only the techniques where light is reflected by one object and consumed by another (the so called inter-reflection) is meant when talking about global illumination.

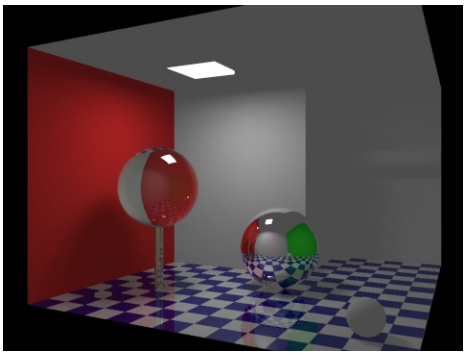

Figure 3: *This scene is rendered without global illumination. The different objects in the scene only consume the light that is emitted by the light source; they don't reflect any of the light that's shining on them.*

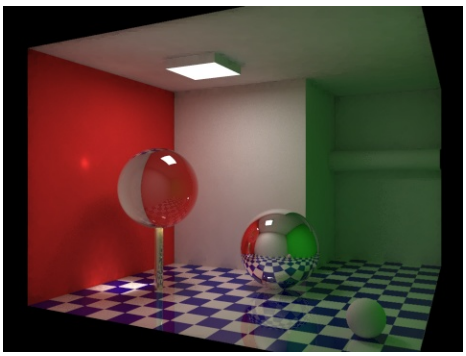

Figure 4: *This scene is rendered with the global illumination techniques. It shows how light is reflected by the surfaces. The rendering shows how color is transferred from one surface to another; this is an effect that happens due to the inter-reflection and is called color bleeding. It can be seen by the green color that's shining on some of the objects in the scene. Another interesting effect are caustics that are seen by the white point on the wall where the light has been focused.*

Since global illumination involves a lot more calculations (a useful collection of formulas can be found in [Dut03]) it is slower than using direct/local illumination. That's also why it has only been used in non real-time graphics, such as movies done by Pixar [CB04] or simulation where the visual output it not required to be real-time.

The light rays' propagation from one object to the other is similar to a heat propagation methods that are used in several simulation processes. Therefore the same formulas can be applied here too. Another cheat that is often used is to create a certain constant that is then used as term to identify how far and how much light propagates from one object to another. This is done to make the equations faster and easier

to calculate; it is mostly only introduced because of current hardware's maximum horsepower.

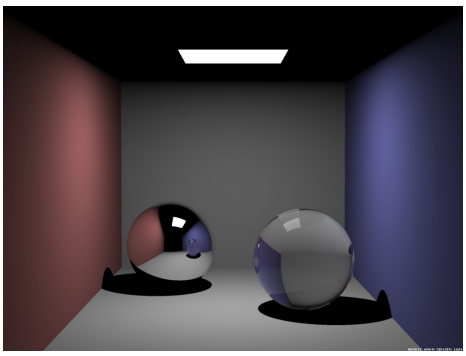

Figure 5: *This image shows a image rendered with ray tracing but without taking indirect illumination into consideration.*

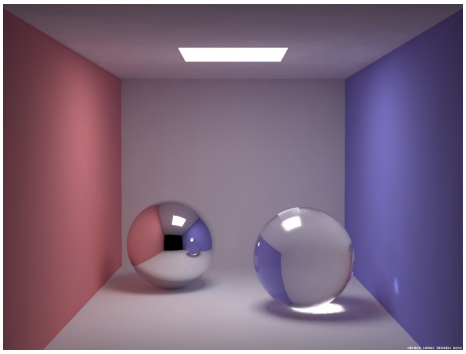

Figure 6: *This image displays the same scene as in Figure 5 but this time the algorithm is taking the indirect illumination into consideration.*

In this paper we want to take the focus on algorithms that allow to create real-time graphics by using global illumination techniques. These are fairly new approaches and include algorithms that do caching to a certain extend to keep the amount of calculations at a level that can be done in real-time on the current hardware.

In Chapter 2, we are going to give an overview of a set of current techniques that are used to create illumination in 3D scenes. This overview should give the reader a way to easily and quickly compare the algorithms that are available today. Chapter 3 talks about Instant Radiosity, a fairly new approach to do real-time radiosity calculations, and its details on how to implement it. Chapter 4 is going to give a deeper dive in how to optimize the approach that has been introduced in Chapter 3. The last chapter, Chapter 5, gives a summary of the different algorithms that have been shown in this paper.

2. Related Work

2.1. Light Maps

Light maps or dynamic light maps [Bea08] are one of the easiest ways to create lighting in scenes. They don't consume any processing power during execution because light maps are computed already ahead of time. This means that before the game (light maps are still used in most of the state-of-the-art games; in combination with other lighting techniques) or 3D scene runs the light map has already been created and baked into the 3D scene.

A light map is a data structure for lighting information that contains the reflected light of most of the surfaces in 3D scenes. The map for a scene is created after the scene has been modeled by the designer. Since light maps are created before the scene is executed the rendering of the light maps can take as long as it requires. Most often light maps are rendered in server farms where the parts of the scenes or the different scenes are split up to several computers who do then the calculations.

The Quake [is96] game from id software [is08] was one of the first games that used light maps to speed up the final rendering while looking quite nicely to the player. Before that games normally used some kind of Gouraud Shading to make the walls look wet and shimmering; as if a distant light shines on them.

Historically seen the most common way to create the light maps is to precompute the vertex lighting by taking the distance of a vertex to the light into consideration. In practice each vertex of the scene is inspected and the distances to the light sources (there might be different static light sources in the scene) is calculated. By using the distance the amount of light that hits the vertex can be calculated and stored into the light map. Objects that might stay in between the light source and the light receiver might also be taken into consideration but that varies from algorithm to algorithm.

Low resolution light maps suffice for most scenes because of the typically low frequency of static light sources. The maps are filtered (bilinear or trilinear filtering might be applied) while being rendered in the scene. This is demonstrated in Figure 7 and Figure 8. Small inconsistencies are not easily seen, especially if the user moves through the scene and does not have much time to look around.

Another important limitation is the fact that only static light sources and static scenes can be calculated and used in the light maps. The scene design has to reflect that limitation and therefore might contain a lot of static objects. This is also seen in Figure 7 and Figure 8 where the fan at the ceiling won't move in the final scene because otherwise the calculated shadows won't fit anymore. In current light maps based games there are also only a few, if any, non-static objects in the scenes. Since the user does not stay very long in the scene it does not alter the perception of realism much.

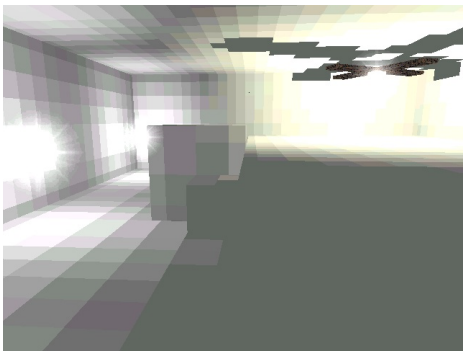

Figure 7: *This image shows a light map as it is created by the off-line algorithm. Any algorithms to determine the shadows might be used when creating the light maps. The results might be saved in a way (for example, in a very low resolution) not to waste too much disk space.*

Figure 8: *This image shows the same light map as in Figure 7, although this time a filter has been applied to smoothen the shadows.*

To create the shadows in light map all possible algorithms can be taken into consideration. Since the time for the creation of the light maps does not matter because of them being computed off-line the best algorithms, such as ray tracing or radiosity should be used and are also often applied when creating the final light maps.

2.2. Ambient Occlusion

The idea of ambient occlusion is to higher the realism in 3D scenes by also taking into account the reduction of light because of occlusions that might happen when the light travels from the light source the light receiver. Unlike other algorithms, like for example Phong Shading, the ambient occlusion algorithm is a global algorithm. This means that the algorithm takes into consideration the whole geometry of the scene. However, the shading that is achieved with ambient occlusion is a very vague approximation of the real-life shading but the results look already a lot more realistic than without applying it; see also Figure 9 for that.

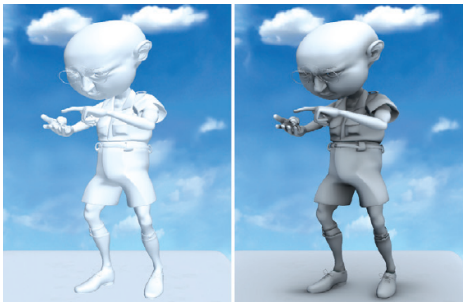

Figure 9: *The scene on the left is drawn only with ambient lighting. The one on the right is including also ambient occlusion and looks way more realistic.*

The most popular approach when doing ambient occlusion is to cast rays in every direction from the surface. The algorithm is run for each ray and determines if the current ray is able to reach the "sky" or at least exit the object and reach the background. In that case the surface is seen as not being occluded in the direction of that ray. The direction of the ray adds illumination to the surface and therefore the surface is made brighter by a constant term. If the ray intersects with an object on his way out that means that the surface is occluded by an object that lies in the direction of that ray and is therefore made darker (by the same constant term as mentioned before).

Langer and Bülthoff have shown in their paper [LB00] that ambient occlusion has the nice side effect to give a better perception of the 3D shapes of the displayed objects. This can also be seen by watching at the sample in Figure 9. The perception of the 3D shape is made easier because surfaces that get less light since they are part of a deepening are also drawn darker. Surfaces that lie in deepenings are also darker when being part of objects in the real world and therefore people tend to see the 3D shape also easier on the virtual objects.

The most correct way to get the amount of light that shines to the surface would be to shoot a ray in all possible directions from the surface. This results in integrating a visibility function (the function that returns whether the surface is visible in the given direction) over the whole hemisphere. Since that is not possible in real-time on the current hardware a solution needs to be found to approximate the results.

One way to stay in a valid time span is to use the so called Monte Carlo method. This method shoots a fixed amount of rays at random directions and takes the results from these rays to create the final results. This method is robust if a random distribution can be computed that is well distributed over the whole hemisphere and not too less rays are shot

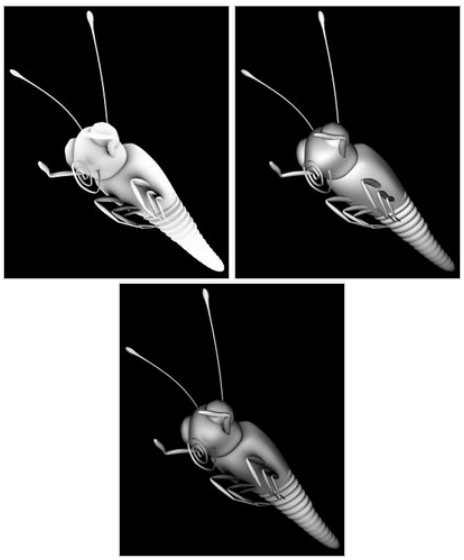

Figure 10: *The animal drawn in the first image is only drawn by using the ambient occlusion. In the next image it is drawn by only using the diffuse color and in the last one the animal is drawn by combining the ambient and diffuse approach.*

from the surface. The amount of rays that are required depends of the current scene and surface and possibly needs to be computed at runtime.

Ambient occlusion can be implemented in a fast way on today's graphic cards as shown in [SA07]. Another interesting resource showing how to combine ambient occlusion with indirect lighting can be found in [Bun].

Crytek has also implemented a new way of ambient occlusion in their recent game, Crysis. They named their approach "Screen-Space Ambient Occlusion" [Mit07] and the idea was to use the value of the z-buffer that they already had in the textures to compute some kind of ambient occlusion. The approach is based on taking the surrounding of a pixel and to do some depth comparison on them. By doing that they were able to compute a darkening factor to get silhouettes around the objects. To get the final ambient occlusion look the effect was only applied to pixels nearby the receiver of the effect. The team also tried to apply the effect on other shadings, like specular and diffuse but found out that the results look best with ambient occlusion.

To reduce the samples they vary the sample position of the nearby pixels that are taken into consideration. The initial positions are distributed like a sphere around the origin pixel. The code to create the effect looks like this:

```
n: the normalized random per pixel
   vector from the texture
i: one of the 3D sample
   positions in a sphere

float3 reflect(float3 i, float3 n)
{
    return i - 2 * dot(i, n) * n;
}
```

Figure 11: *The first image shows a sample scene with Screen Space Ambient Occlusion and the second image shows the same scene by having the effect disabled.*

2.3. Extended Ray Tracing

Ray Tracing algorithms cast rays through the pixels of the image plane into the scene to find points to be rendered into the pixels. Early algorithms did not cover global illumination, but in the meantime several extensions have been published, such as the following two techniques.

2.3.1. Path Tracing

To approximate the rendering equation, Path Tracing makes use of a Monte Carlo approach to sample the hemisphere over an intersection point of a ray (from the camera). Depending upon the BRDF (bidirectional reflectance distribution function) of the material, several secondary rays are cast from the intersection point into the hemisphere and they all contribute to the final color of the image pixel. So a ray is shot from the camera and its path in the scene is traced.

The efficiency of this approach is reduced if small light sources are used. In this case the number of secondary rays needs to be high to reduce noise in the produced image. Thus

the need for a very high total number of rays prevents this algorithm from rendering complex scenes in real-time on current PCs, although it is an elegant and fully dynamic approach. Metropolis Light Transport [VG97] is an extension providing better performance due to re-use of paths transporting high energy.

2.3.2. Photon Mapping

Photon Mapping [Jen96] is not a rendering algorithm; it rather extends algorithms such as Diffuse Ray Tracing or rasterizers by a preprocessing step to compute a photon map which is used during rendering to approximate global illumination.

During preprocessing, particles are emitted from the light sources and their paths in the scene are traced using standard ray tracing techniques. Intersection points of photons and geometry are stored together with the incoming direction in a special, geometry-independent structure called the photon map (e.g. a kd-tree).

During rendering, the photon map is used to approximate the incoming radiance of a point (by looking up nearby photons and their incoming directions). Thus the rendering equation and global illumination can be approximated for every surface point in the scene.

Although this technique may provide reasonable rendering times, the preprocessing step prevents it completely from being applicable to dynamic scenes (it is only independent of the point of view; light sources and geometry must not change).

2.4. Splatting indirect illumination

Reflective Shadow Maps have been developed by Carsten Dachsbacher and Marc Stamminger in 2005 [DS05]. The idea is to extend shadow maps to create an algorithm that can be used for plausible looking indirect illumination. A reflective shadow map is an extension of the concept of shadow maps (that are used to create shadows in current application) by making each pixel of that map an indirect light source. The indirect illumination of a certain point due to these indirect lights is evaluated on-the-fly by doing adaptive sampling in the pixel shader. Since the algorithm uses concepts that are already available in hardware it can be implemented in a very fast way.

In the paper the authors describe that only points that are visible in a shadow map, that has been created from the view of a surface, can send light to that exact surface. Therefore they have come to the idea of using shadow maps to understand what points are in question to send indirect light information. The algorithm creates normal shadow map (holding the depth information), but also a normal buffer, a world space position buffer and the flux. These elements are added as additional components to each point in the map.

The flux is the light that is emitted by a given point in the shadow map. The calculation of the flight is straight forward: for a uniform parallel light the flux is a constant value; and for a spotlight the flux decreases with the cosine to the spot direction due to the decreasing angle. After that the so computed flux is combined with the reflective component of the emitting surface (the material and other components are taken into consideration) to get the final value for the shadow map.

The algorithm introduced in [DS05] is not taking occlusion into consideration. This might lead to wrong results: a point that is not lit by another indirect light source, because there is an object between the light emitter and receiver, might still get the indirect illumination due to that limitation. The authors of the paper claim that this is not happening very often and therefore can be ignored. This has also to do with the assumption that they only use points as passive light sources if they are very near to each other. In that case it is very unlikely that objects might occlude the view from one point to another. Also, ignoring this fact speeds up the algorithm quite a bit; otherwise test for occlusions would needed to be added to the algorithm. An example showing this problem is seen in Figure 12.

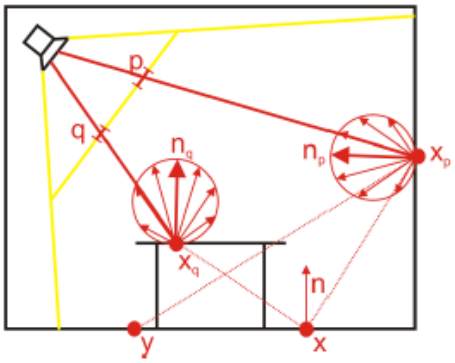

Figure 12: *In this example the point xp shines as indirect light source to y although there is an object in between the two points.*

To keep the algorithm real-time additional considerations needed to be made: the usual shadow map size is around 512x512 pixel or even bigger. That is too much to keep the algorithm real-time. Therefore, only a part of these light sources is evaluated (e.g. 400, as propagated in the paper). Valid results are achieved by using an importance-driven approach, which means that only relevant pixels should be taken into consideration as passive light sources. In general can be said that the distance of two pixel in the shadow map is a reasonable approximation to the distance of the two pixels in world space. If the depth values with respect to the light source differ significantly, the world space distance is much bigger and the influence has been overestimated.

However, and that's the key point of the approximation, the important indirect lights will always be close and therefore must also be close in the shadow map.

Figure 13: *The first image on the left holds the local illumination and shadow mapping. The indirect illumination is approximated from the reflective shadow map (the image in the center) and the last image shows the combination of both.*

The algorithm, as described here, is still to slow to perform in real-time. Therefore the authors in the paper suggest to perform first the indirect illumination in a low resolution image from the camera view. After having gotten the low-resolution image the whole scene is rendered in full resolution (from the camera view again) and a check is performed to see whether the indirect light can be interpolated from the surrounding low-resolution samples.

Such a low-resolution sample is regarded to be suitable for interpolation if the normal of the sample is similar to the normal of the current pixel and if the world space location is close to the one of the pixel. If the sample is found as suitable the indirect illumination is interpolated to get the results of the indirect lighting; otherwise the pixel is discarded and the indirect illumination is then recomputed in the final pass and at full resolution. The low-resolution image is already good enough for a lot of the tests and therefore speeds the algorithm up by a significant factor.

2.5. Precomputed Radiance Transfer

This approach [SKS02] is based on the idea that the radiance transfer of an object can be generally precomputed, depending upon a set of basis functions representing the surrounding spherical lighting environment.

This lighting environment is usually an environment cube map around the object's convex hull. It is assumed to be distant and of low frequency, so that it may be represented by a small set of basis functions (e.g. spherical harmonics, a Fourier-Transformation analogue for the surface of a sphere).

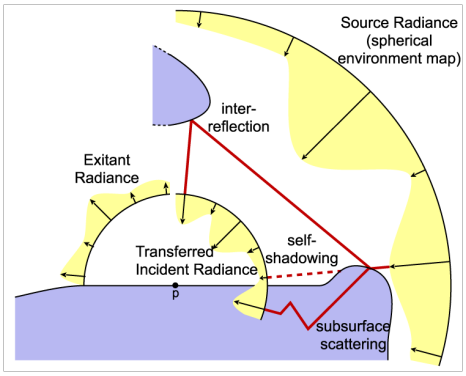

Figure 14: ***Precomputed Radiance Transfer principle***. *The lighting environment surrounding an object is compressed by projecting it onto a small set of basis functions (e.g. spherical harmonics). For each vertex of the object, the radiance transfer is precomputed, i.e. the weight of each basis function in the exitant light, including self-shadowing, inter-reflections and/or subsurface scattering. During rendering, the light exiting towards the camera is calculated by a linear combination of the radiance transfer vector and the projected lighting environment.*

For each vertex of the object, a vector is precomputed using an offline global illumination renderer. This radiance transfer vector is used to assign a weight to each basis function of the lighting environment so that the exitant light for a point on a surface may be simply calculated by a linear combination of the vector and the lighting environment projected onto the basis functions. This vector represents the radiance transfer operator and may include self-shadowing, inter-reflections and subsurface scattering. The method can also be extended to handle non-diffuse BRDFs; in this case, a simple dot product is replaced by multiplying a matrix by the radiance transfer vector and then integrating over the surface's BRDF.

Although this algorithm is able to provide stunning results, it has serious disadvantages. Firstly, the objects need to be static, handling animated objects is difficult. Secondly, the lighting ambient is restricted to low-frequency, distant lighting, which is not feasible for a lot of scenes. Thirdly, radiance transfer between objects is not accounted for, therefore true global illumination could only be achieved if the whole scene was treated as one single object. In this case, the whole scene would need to be static and the lighting environment would need to surround the whole scene, i.e. there could not be any light sources inside the scene.

2.6. Radiosity

Radiosity has been one of the first approaches to the global illumination problem. It is based on the assumption that all surfaces are diffuse, i.e. incoming light is reflected uniformly in all directions. It models the light transfer by regarding the law of conservation of energy: all light which is not absorbed by a surface is reflected onto other surfaces.

To model the light transfer in the scene, the surfaces are first subdivided into smaller patches. Then a so-called *form factor* for each pair of patches is computed, which represents the amount of light exchanged by both patches. These form factors, due to their very high number, consume a large amount of memory and take a lot of computation time (time complexity: $O(N \log N)$ with N = number of patches). Therefore the Radiosity algorithm is suited for static scenes only for which the form factors can be precomputed; it is independent from the point of view and the light sources though and therefore may be a better choice than Ray Tracing in certain situations.

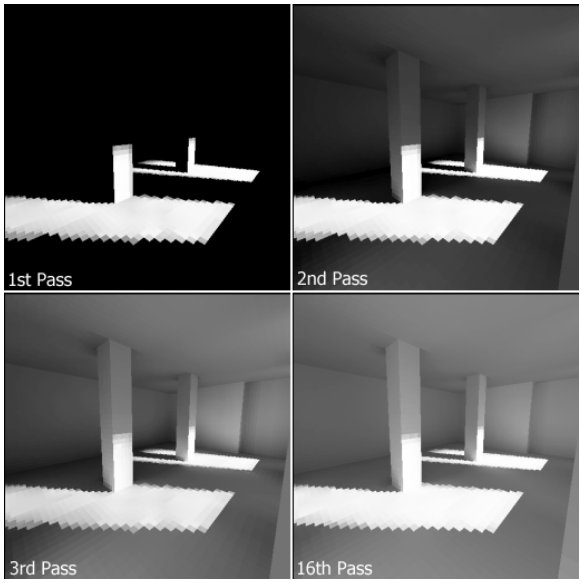

Figure 15: ***Progressive Radiosity rendering****. In each pass, additional incoming indirect light is added to the image.*

The rendering part of the algorithm is very well suited for progressive rendering (see figure 15). In a first pass, only direct illumination is taken into account. In later passes, indirect lighting is added by computing the additional incoming light from all other patches using the form factors. Thus each pass i adds indirect light which takes i bounces to reach a point. So the brightness of the image will increase after each pass until it eventually converges (a few bounces already deliver plausible lighting).

Due to the costly computation requirements, Radiosity is nowadays often replaced by extended Ray Tracing techniques which consume less memory, typically provide better performance and do not necessarily restrict the surfaces to diffuse ones. It may still be used to precalculate accurate light maps for static diffuse global illumination though.

2.7. Instant Radiosity

Instant Radiosity traces paths from the primary light source and creates new virtual light points (VPLs). These light sources are placed at each reflection point (where the light of the primary light source hits a reflector, e.g. a wall) and used to illuminate the scene. This approach is described in great detail in Chapter 3.

3. Instant Radiosity

Instant radiosity is the idea to use the features of radiosity to create a real looking lighting for a scene and that in real-time. The most common approach until now is to apply the radiosity algorithm as an off-line algorithm, which means that the global illumination effects are precomputed as static lighting solutions (or map) before the scene is even rendered. The final results are then placed into the scene when doing the rendering at the client side. Radiosity as off-line algorithm is a technique that is often used when creating light maps because the results simulate real-life lighting quite accurate.

The algorithm that is described in this section is mostly based on the previous work of [LSK*07] and [Kel97]. [Kel97] has shown that instant radiosity is possible on current hardware and laid the ground for most of the subsequent papers. The method constructs random paths from the light sources and creates new virtual point lights (VPLs) where these paths encounter a surface. When used together with shadow maps these point lights represent the full indirect illumination of the scene and are also optimized for usage with the current graphics hardware. [LSK*07] have then tweaked the algorithm to perform even faster.

The following outline shows the different steps of an instant radiosity algorithm:

- Calculate the amount of secondary light sources for the scene.
- Determine the visible area from the light source.
- Equally distribute VPLs (virtual point lights) in the area that is lighted by the light source.
- Compute intensities for the VPLs. All virtual point lights together should only have the same or less intensity than the primary light source.
- Use them as secondary light sources to illuminate the scene with secondary lighting.

The algorithm can be summed up to the following (see also Figure 18): In instant radiosity rays are traced from the primary light source to the first object that is crossing the

	Preprocessing	Dynamic objects	Dynamic lighting	Rendering performance	Restrictions
Light Mapping	Yes (light maps)	No	No	Very high	Static precomputed lighting only
Local Ambient Occlusion	Yes (ambient occlusion textures)	Animated objects need one texture per frame	Yes	Very high	Not physically grounded: darkening of the ambient term only based on the exposedness of a surface point
Global Ambient Occlusion (SSAO)	No	Yes	Yes	High	Not physically grounded (see above), exposedness is estimated
Path Tracing	No	Yes	Yes	Very low	Not real-time capable yet
Photon Mapped Diffuse Ray Tracing	Yes (photon map)	No	Only color invariant	Low	Not real-time capable yet
Splatting Indirect Illumination	No	Yes	Yes	High	Not physically grounded: neighbouring fragments simply influence each other and visibility is completely ignored
Precomputed Radiance Transfer	Yes (radiance transfer vector per vertex)	Animated objects need special treatment	Yes	High (for diffuse surfaces)	Lighting environment needs to be distant and of low frequency; object interaction is not modeled unless the whole scene is modeled as a single static object
Radiosity	Yes (form factors)	No	Yes	Very low	Diffuse surfaces only
Instant Radiosity	No	Yes	Yes	Medium	Diffuse surfaces only

Table 1: *Comparison of common global illumination algorithms*

path of the ray. A virtual light source is then placed where the object was hit by the ray. These virtual light sources are then used as sources for the indirect illumination and are taken into consideration when rendering the scene from the camera's view point. It has been shown [TL04] that one bounce is enough for the passive light sources to make the scene look like being lit by a real-life light source.

Instant radiosity, as proposed in the current papers, can be used without any precomputation and is able to handle dynamic lighting conditions. Since the VPLs are computed for each frame it is not important that the light fulfills certain conditions about its movement. Further, the geometry can be static or dynamic because all of the secondary lights are recomputed for each frame. The recreation process of all VPLs for each frame is very time consuming. Therefore, in the next chapter speed improvements and enhancements for the algorithm will be introduced. They restrict the behavior of the primary light source and the geometry but increase the performance dramatically; especially for scenes with complex geometry.

The instant radiosity algorithm is only used to compute the indirect illumination. After having done that the resulting lighting needs to be combined with the direct illumination to create the high quality lighting. To make the lighting look even more realistic it might be interesting to combine it with some kind of precomputed lighting (for example with ambient occlusion that has been rendered beforehand).

3.1. VPL Distribution

The ideal distribution of the VPLs, as seen from the point light source, would ideally follow the directional intensity distribution of the light source. This means the distribution of the virtual light sources should represent a similar allocation of the total light power.

The implementation in [LSK*07] supports two types of primary light sources: 180 degree spot lights with a cosine falloff and omni directional point lights. The domain of the emitted light directions is mapped into the 2D domain to simplify the computations.

With spot lights the most suitable 2D domain is the unit disc. This representation is gained by flattening the z-coordinate of the hemisphere pointing at the direction of the light source (see first image in Figure 19). For omni direc-

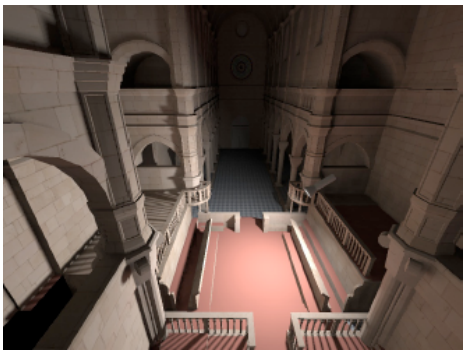

Figure 16: *This figure shows an example of direct and indirect illumination (instant radiosity) combined. The indirect illumination is computed by using the virtual light sources (VPLs). The scene contains approximately around 80,000 triangles and runs at 49 fps in 1024x768. The computations related to indirect illumination take around 71% of the total rendering time [LSK*07]. These numbers have been computed on a dual-core 2.2GHz AMD Athlon 64 with 1GB of memory and a NVIDIA GeForce 8800 GTX with 768 MB of memory.*

Figure 17: *This figure shows the virtual point lights (VPLs) that have been created for the given spot light. They are evenly distributed in the viewing area of the point light.*

tional light the algorithm uses the surface of the unit sphere (see second image in Figure 19).

Most of the operations on the sampling points require some kind of triangulation. [LSK*07] suggest in their paper to use the Delaunay triangulation and the associated Voronoi diagram that is then computed for the entire set of sampling points (that are the points that correspond to the VPLs). These operations are performed in the same domain where the rest of the geometry operations applying to the VPLs are computed, i.e. the unit disc or the unit sphere depending on what light is used. The resulting Voronoi diagram is seen in Figure 19.

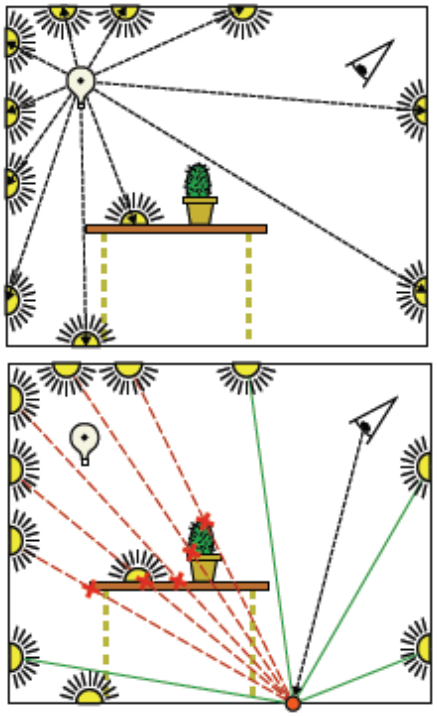

Figure 18: *The principle of instant radiosity: the first image shows how paths are traced from the light source to the first objects that they encounter. A second light source (a virtual point light) is placed on the crossing point. These light source are then used to compute the indirect illumination of the scene. The second image shows the scene rendered from the camera, where the orange dot at the bottom is rendered, and how the VPLs are used to do the indirect illumination of that dot.*

4. Extensions & Optimizations

The original article about Instant Radiosity [Kel97] dates from the year 1997. Therefore some extensions and optimizations of the original algorithm have been published in the meantime.

The rapid development progress of 3D hardware, for instance the ability of current graphic chips to be fully programmed using shader programs, offers great potential to optimize Instant Radiosity for real-time performance of complex dynamic scenes on today's high-end PCs.

4.1. G-buffer (Deferred Shading)

Instant Radiosity creates a lot of virtual point lights which all illuminate the scene. The Deferred Shading algorithm [ST90] suits this scenario very well, because it avoids rasterizing the geometry multiple times, which would be mandatory in a traditional multi-pass approach.

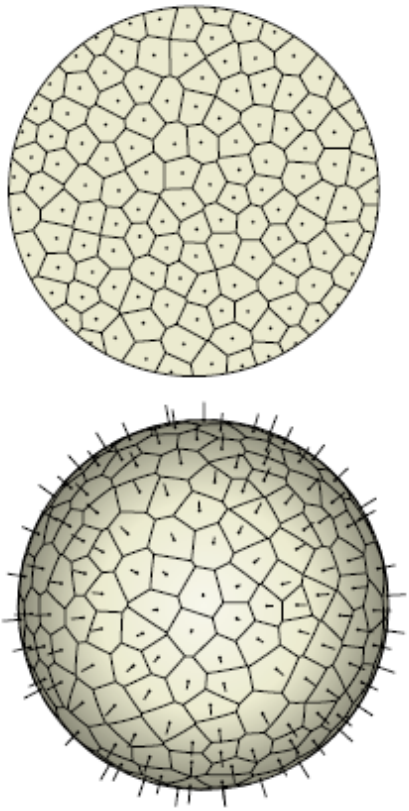

Figure 19: *In each frame the VPL directions are mapped into a 2D domain. Then the computations for the VPLs are done in this domain and finally, when a new VPL is being created, the point is mapped back into the 3D space where a ray then determines the position of the newly created VPL.*
The image on the top shows the sampling that is done when working with a spot light. In that case a unit disc is being created. It is created by dropping the z-coordinates of the hemisphere that is seen from the spot light.
The second image shows the same distribution for a omni directional light source. The sampling points are located on the surface of a unit sphere.

The algorithm is based on the fact that the computation of a local shading model (e.g. Lambertian or Phong) typically requires the position, surface normal and color of each point to be known. The idea is to store these attributes of the geometry's fragments in a so called G-buffer in a first pass and to use them in later passes where the fragments are shaded by multiple light sources. This G-buffer obviously occupies some graphics memory (3 floating-point textures each containing 3 16-bit components require about 30 MB when using a resolution of 1680×1050 pixels).

So in a first pass, the world-space position, normal and color of each fragment are rendered into the G-buffer. These vectors are rendered into 3 floating-point textures containing 3 components each. This requires the graphics chip to support multiple render targets (MRT, in this case at least 3) and floating-point textures as render targets. In the fragment shader program, the fragment's attributes can be written directly into the 3 render targets, requiring a single render pass.

In later passes, a certain number of light sources (limited by the hardware) is processed per pass. The incident illumination from each light source is computed for every fragment by looking up the fragment's attributes in the G-buffer and using the light attributes (e.g. shadow map, shadow map look-up matrix, position, color(s), other parameters). These incident illuminations are then accumulated to obtain the final incident illumination for each fragment.

Deferred Shading is not suited for transparent objects, because more than one fragment may be visible in a single render-target pixel and all of these fragments need to be shaded individually. Transparent objects therefore need to be taken care of separately.

Another limitation consists in the inability to use multisampling-FSAA (Full-Screen Anti-Aliasing) on pre-DirectX-10 graphic chips. This is demonstrated by the Unreal Engine 3 which uses Deferred Shading.

4.2. Interleaved Subsampling and Filtering of Indirect Illumination

Computing the incident illumination from hundreds of VPLs for every fragment still requires multiple passes on current hardware and obviously a lot of processing power.

Laine et. al [LSK*07] propose to use only a substantially smaller subset of VPLs for each fragment. The assignment of VPLs to fragments consists in a high-frequency noise pattern illustrated in figure 20, which allows neighbouring incident illuminations from the different VPL subsets to be low-pass filtered to approximate the incident illumination from all VPLs for each fragment.

The G-buffer is subdivided into a grid of rectangular blocks of $n \times m$ pixels, and each VPL is assigned to one pixel in a block so that the number of VPLs per pixel is roughly the same (about $\frac{\#VPLs}{n \times m}$. Thus the number of VPLs illuminating a fragment is reduced by the factor of $n \times m$, resulting in far less passes (ideally only 1) and computations. Since only the incident illumination from the VPLs is accumulated, only the position and the normal of each fragment is needed, the color is used later on when the fragment is finally shaded.

Due to the need to set shader program parameters for each subset of VPLs (e.g. shadow maps, shadow map look-up matrices, positions, colors, other parameters) we will want to render all fragments belonging to a specific subset of VPLs consecutively. Using the initial G-buffer as input, we would

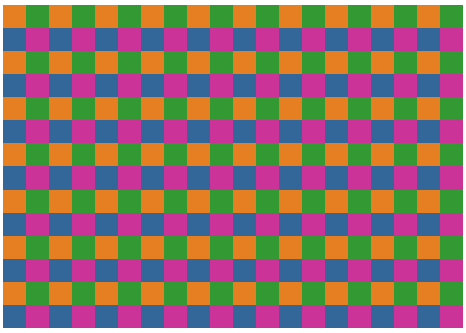

Figure 20: *G-buffer interleaved subsampling. Only a subset of VPLs is evaluated per fragment. In this figure, the image is subdivided into 2×2-pixel blocks. Each of the 4 fragments in a block is assigned to one of the 4 different VPL subsets, i.e. one subset per color in this figure. The resulting high-frequency assignment pattern allows the indirect illumination to be low-pass filtered in a later step.*

have to process only the sparse fragments belonging to the current VPL subset. This could be accomplished by using a stencil test and a pre-computed stencil buffer or by rendering a rectangle for each fragment, but the performance would not excel and we would not be likely to get a high cache-hit rate neither due to the sparse fragments. The direct way of processing all fragments at once may be feasible by packing the attributes of all VPLs (including the shadow map look-up matrices) into one or multiple textures, selecting the appropriate shadow map by indexing a texture array (a new DirectX 10 feature) and looking up the appropriate VPLs' data on a per-fragment basis. Since we believe that the mentioned approaches have not been proposed yet, we cannot publish any performance numbers, but the direct approach may be worth being considered.

Laine et al. overcome this problem by the use of an intermediate buffer called the split G-buffer. In this buffer, the G-buffer is re-ordered so that all fragments are grouped into $n \times m$ tiles, one per subset of VPLs. Laine et. al use 4×4 tiles and a maximum of 256 VPLs, which results in a maximum of $\frac{256}{4 \times 4} = 16$ VPLs per fragment. By rendering each tile separately, the 16 incident illuminations for each fragment are computed and accumulated in a single pass and the final illumination is written into a illumination buffer. This buffer is then re-ordered so that the pixels are laid out just like the ones in the initial G-buffer (performing the inverse of the G-buffer $\rightarrow$ split G-buffer conversion).

The illumination buffer now typically suffers of high-frequency noise because neighbouring pixels represent the incident illumination from different VPL subsets to neighboring fragments. The illumination buffer is thus low-pass filtered using a box filter of the dimensions $n \times m$ and a varying support, i.e. only selected pixels contribute to the averaged value - pixels that represent neighbouring fragments

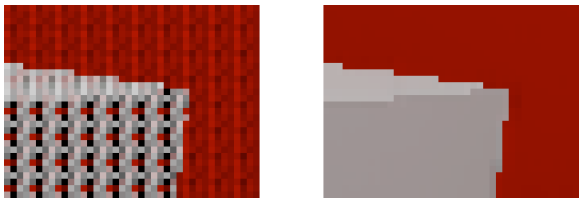

Figure 21: *Indirect illumination filtering*
Left: *The illumination buffer typically contains a structured noise pattern due to interleaved subsampling of indirect illumination.*
Right: *The spatially-varying box filter completely removes the structured noise.*

whose distance in world-space from the centered fragment does not exceed a certain threshold and whose normal does not differ too much from the centered fragment one's (e.g. if $normal_1 \cdot normal_2 > 0.8$). Laine et al. claim that this filter is able to remove the structured noise completely from continuous surfaces (see figure 21).

Now that the final indirect illumination has been computed per fragment, it replaces the constant ambient color of a light in the Phong lighting model. Combined with direct and optionally static (light map) illumination as well as the color (stored in the G-buffer) each fragment can finally be shaded.

4.3. Incremental Instant Radiosity

Rendering a shadow map for each of the hundreds of VPLs in each frame of a dynamic scene obviously requires a lot of processing power. To overcome this bottleneck, Laine et. al [LSK*07] propose to exploit the temporal coherence of VPLs in consecutive frames. In the next frame, most VPLs will typically still be valid even if the primary light source moves and/or the surrounding geometry changes due to movement or animation, meaning that in the next frame most VPLs will not get occluded and still be in the illuminating region of the primary light source. Therefore most VPLs and their shadow maps could be re-used in the next frame, reducing the number of shadow maps to be rendered drastically.

Invalid VPLs will need to be replaced by newly generated ones and we will want to make sure the distribution of VPLs is kept as close to the ideal as possible while re-using VPLs from previous frames. For this reason, Laine et al. use a budget of at least *recalcMin* and a maximum of *recalcMax* VPLs to be newly generated per frame. Thus the VPLs distribution is improved even if all VPLs are still valid and *recalcMin* is greater than 0. On the other hand, an upper bound ensures stable frame rates even in situations where a lot of VPLs have become invalid, at the cost of temporary degradation of quality.

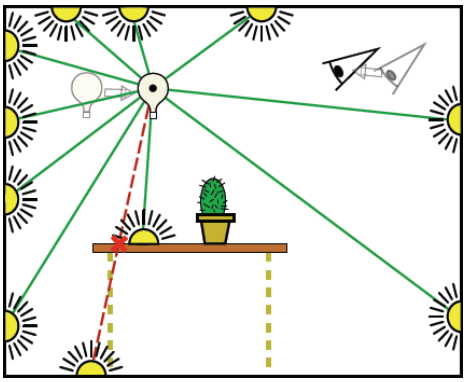

Figure 22: *Incremental Instant Radiosity principle. After movement of the primary light source, the VPL at the ground has become invalid and will thus be replaced by a newly generated one. All other VPLs are still valid and may be reused.*

Determining the validity of a VPL may be accomplished by casting a ray from the light source to the VPL and checking if the path is not occluded. A look-up into the primary light source's shadow map may also be feasible.

Choosing the VPLs to be deleted and selecting the sampling points of the new ones is based on the attempt to keep the VPLs distribution as close to the ideal as possible. Firstly, all invalid VPLs are deleted. If this number is smaller than *recalcMin*, the remaining VPLs to be deleted are the ones whose sampling points are closest to the ones of other VPLs. After deletion, the deleted VPLs are replaced by a maximum of *recalcMax* new ones whose sampling points are in the center of the greatest empty circles in the 2D domain of sampling points. Laine et al. map the hemisphere of spot light sampling points onto the unit disc and the sampling points of omnidirectional point lights onto the surface of the unit sphere. The distribution of VPL samping points in these domains is represented by a Delaunay triangulation and the associated Voronoi diagram. This structure is then used to find VPLs to be deleted and sampling points for new VPLs, for details please refer to their paper.

While this approach works very well for moving light sources and static geometry, it does not address completely dynamic scenes. To perform accurate visibility testing, the shadow maps of all VPLs need to be updated each frame if there are dynamic objects (moving and/or animated). Rendering only static geometry when generating the shadow maps might be a feasible approximation if the dynamic objects are small occluders. Otherwise, at least *recalcMin* would need to be increased to update more VPLs per frame.

4.4. Shadow Maps

Rendering the shadow maps for the VPLs is computationally intensive. So the question arises how the shadow maps should be generated. Due to the assumption that all surfaces are diffuse, the VPLs are uniform hemispherical point lights, i.e. a shadow map needs to cover the whole hemisphere above the surface at which the VPL is located. Although a cube map (5 faces each) could be used as shadow map, a parabolic shadow map [BAS02] seems to be a more adequate choice to us. It is naturally suited for a hemisphere and should provide better rendering times and a better quality-to-memory-requirements-ratio than a cube map. The resolution of the shadow maps is a key parameter because it controls both quality and rendering time as well as the required graphics memory (e.g. the shadow maps of 256 VPLs in a resolution of 256×256 pixels and containing 24-bit depth values already consume about 48 MB). For accurate precision, the scene should be sufficiently tesselated, otherwise the parabolic map may exhibit strong artifacts (e.g. large bounding geometry such as floors and walls should be subdivided into smaller triangles). This behaviour is due to the fact that only the vertices are parabolically transformed in the vertex shader, but the triangles are still linearly rasterized.

Exploiting the new geometry shaders and combining them with multiple render-targets, it is possible to compute multiple shadow maps in a single rendering pass, preventing the need to issue a lot of draw calls for the geometry. A triangle of the geometry is replaced in the geometry shader by a separate corresponding triangle for each render-target (shadow map). Please note that this approach is not compatible to traditional light-frustum culling because multiple light-frustums are used in a single pass.

Another point worth mentioning regards shadow map aliasing. Due to the usually low intensity of a single VPL, high-quality filtering (e.g. using Variance Shadow Maps [DL06]) may be an overkill and simple nearest-neighbour or hardware PCF (percentage closer filtering) may already deliver satisfying results.

4.5. Bidirectional Instant Radiosity

This extension, published in 2006 by Segovia et al. [SIMP06], does not target primarily the performance of IR (Instant Radiosity) but tries to optimize the locations of the VPLs. In essence, it extends Instant Radiosity by bidirectional path tracing to place the VPLs at locations where they illuminate the part of the scene seen by the camera, i.e. to select VPL locations which have influence on the illumination of the scene rendered from the camera.

To accomplish this, Segovia et al. use a combination of the standard IR and their so-called reverse IR algorithm to find VPL locations. In the standard algorithm, paths are traced from the light source and intersections with scene geometry

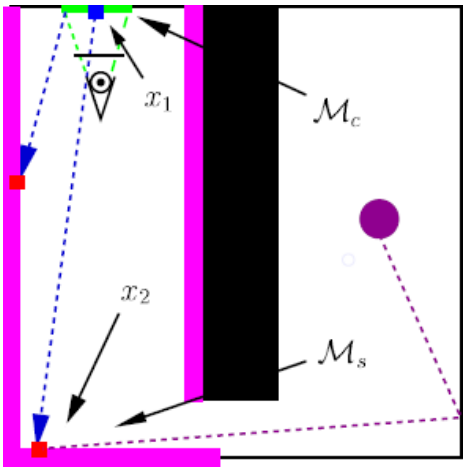

Figure 23: *Reverse Instant Radiosity. First, rays are cast from the camera to sample M_c (the green points visible from the camera) to obtain the point set x_1. Then random rays are cast from these points to sample M_s (magenta, points illuminating the visible scene indirectly) to obtain x_2, the set of reverse VPL candidates.*

are VPL candidates. In the reverse algorithm (see figure 23), rays are first cast from the camera to find points which can be seen from the camera. To find a VPL which illuminates one of these points, another random ray is cast from the point into the scene; the first intersection will be a VPL candidate. So we trace random paths of length 2 from the camera to find VPL candidates which illuminate an area of the scene seen by the camera.

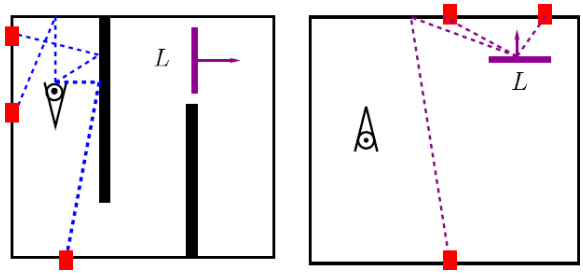

Figure 24: *Reverse vs. standard IR*
Left: *Reverse IR is better suited for this scenario because relevant VPLs would only be found after numerous bounces if applying the standard algorithm, i.e. tracing paths from the light source.*
Right: *The standard algorithm will find more relevant VPLs because the light source directly illuminates a small area of the scene. Finding reverse VPLs of comparable relevance is difficult.*

Figure 24 illustrates that one scene might be suited for the standard algorithm whereas another one for the reverse algorithm. So Segovia et al. decided to combine both approaches by first creating a set of $\frac{N}{2}$ standard and $\frac{N}{2}$ reverse VPL candidates. A subset of these are then resampled to form a family of final VPLs with equal estimated importance to the illumination of the scene seen by the camera. So depending upon the scene, either the standard or the reverse algorithm may deliver more relevant VPLs. The VPL importance is estimated by further ray-tracing. Please refer to their paper for more details.

So this extension of Instant Radiosity provides a way of finding relevant VPL locations, especially if the scene is rather closed and most of the visible illumination consists in indirect illumination.

5. Conclusions

Instant radiosity tries to simulate the real world illumination and the results look very much like plausible illumination conditions. The shadows that are created by combining the indirect illumination (created by this fairly new approach) and the direct illumination look soft as they appear in the real world. Also, the light gets reflected by the walls and other surfaces and therefore even objects, where no direct light shines at, get some lighting as would be expected in real world scenarios.

The approach as shown in [LSK*07] has still the following drawbacks:

- **Only static objects:** The current implementation of the incremental instant radiosity approach does only allow static objects as passive light sources. Dynamic objects can consume the passive lighting but they cannot be used as virtual point light (VPL) source. This limitation is in place because with dynamic objects the implied caching methods (the re-usage of VPLs for multiple frames) won't work and the algorithm could not stay in real-time anymore.
- **Only diffuse surfaces:** The approach also only works on diffuse surfaces. Glossy surfaces are not suitable because they would need more than one bounce to be lighten properly. More than one bounce would mean that more VPLs need to be created and tracked through the scene. Therefore, the algorithm could not stay in real-time anymore because of the additional calculations that need to be performed.
- **Ray tracing:** To make the VPLs work a ray tracer is needed. This might be a drawback since that might mean that a lot of work needs to be spend to create this new item for a current engine. But it is seen that some kind of ray tracer is required for a lot of the current effects and/or features that are already implemented in current engines; e.g. when a bullet is shot the path of it is traced until it hits a wall, body or any other object in the scene.

The first two drawbacks might be overcome when newer

and faster hardware is available, and the last one might also be solved when more and more engines implement some kind of ray tracer to make other effects work properly too.

5.1. Performance results

The performance results were gained in [LSK*07] by comparing the incremental instant radiosity (re-use of VPLs over more than one frame) approach with a radiosity approach where all the VPLs were recreated for each frame. Other than that the two algorithms were exactly the same. As testing machine the authors of the paper used a dual-core 2.2GHz AMD Athlon 64 with 1GB of memory and a NVIDIA GeForce 8800 GTX with 768 MB of memory.

The tests were applied to three different scenes. Cornell is the classic Cornell box, Sibenik is an architectural model, and Maze is a simple surrounding scene with a detailed statue. In each scene the time of the two approaches were measure. For the direct illumination one single light source was used. The number of VPLs was fixed to 256, which has been found as the best balance between visible artifacts and speed of rendering. The recomputation budget (for the instant radiosity approach) was set to 4-8 VPLs per frame for all the different test cases. The size of the shadow maps for the VPLs was set to 256x256 with 16 bit depth values. For the direct illumination a shadow cube map with the size of 1024x1024 pixel was computed.

The G-buffer was only split into 4x4 tiles, since that resulted in the best rendering speed with 256 VPLs. This limitation could eventually be walked-around by using the direct method that has been introduced in chapter 4 of this paper.

The tasks of the algorithm were executed in parallel: this means that the VPL management tasks, that need to be done on the CPU, are always executed in parallel with the GPU tasks by allowing only one-frame delay in the positions and intensity of the VPLs. The CPU is not the limiting factor for the algorithm and therefore the frame rates are exactly the same when using the spot light or the omnidirectional light, although the VPL management computations for the omni directional light is notable slower.

The results in [LSK*07] show that the incremental instant radiosity algorithm increases notable the speed of the rendering. That's especially seen for complexer scenes where the compared method had a hard time with recreating the VPLs for each frame. When comparing the two algorithms it can be seen that the speed up factor is between 1.4 and 6.8, which means that, for example, a scene that runs at 7.1 frames per seconds, with the compared algorithm, runs at 48.6 frames per second when using incremental instant radiosity. The comparison and results for the scenes and the two approaches is found in table 2.

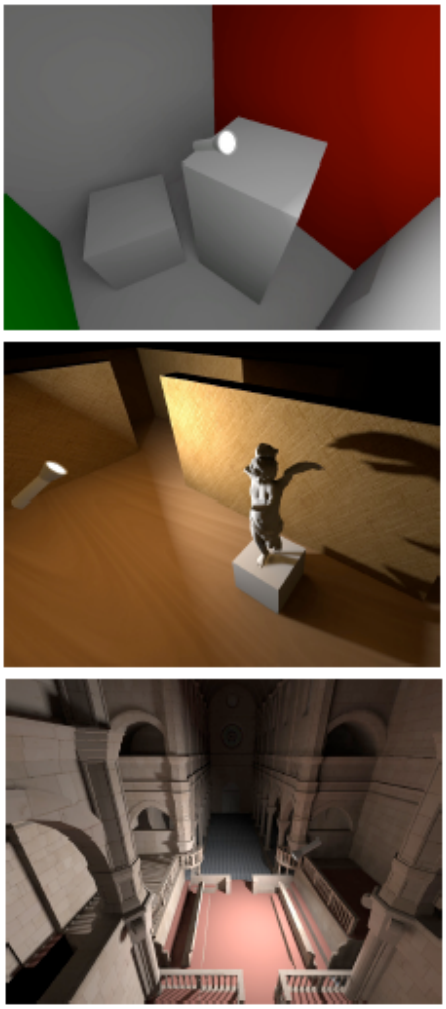

Figure 25: *The test scenes that have been used. The first image is the Cornell scene, the second the Maze scene, and the last image the Sibenik scene.*

References

[BAS02] BRABEC S., ANNEN T., SEIDEL H.-P.: Shadow mapping for hemispherical and omnidirectional light sources. In *Advances in Modelling, Animation and Rendering (Proceedings Computer Graphics International 2002)* (Bradford, UK, 2002), Vince J., Earnshaw R., (Eds.), Springer, pp. 397–408.

[Bea08] BEAM J.: Dynamic lightmaps in opengl, June 2008. `http://www.3ddrome.com/articles/dynamiclightmaps.php`.

[Bli77] BLINN J. F.: Models of light reflection for computer synthesized pictures. In *SIGGRAPH '77: Proceedings of the 4th annual conference on Computer graphics*

Scene	Resolution	FPS	FPS	Speedup Factor
Cornell	1024x768	35.1	65.1	1.9
Cornell	1600x1200	21.2	29.7	1.4
Maze	1024x768	12.5	49.2	3.9
Maze	1600x1200	10.2	28.5	2.8
Sibenik	1024x768	7.1	48.6	6.8
Sibenik	1600x1200	6.3	25.9	4.1

Table 2: *Timings from the test scenes. The columns from left to right: the name of the scene; the resolution; the frames per second for the comparison method; the frames per second for the incremental instant radiosity method; the speedup factor when comparing both methods.*

and interactive techniques (New York, NY, USA, 1977), ACM, pp. 192–198.

[Bun] BUNNELL M.: *Dynamic Ambient Occlusion and Indirect Lighting*. NVIDIA Corporation.

[CB04] CHRISTENSEN P. H., BATALI D.: An irradiance atlas for global illumination in complex production scenes. In *Eurographics Symposium on Rendering* (2004).

[DL06] DONNELLY W., LAURITZEN A.: Variance shadow maps. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2006), ACM, pp. 161–165.

[DS05] DACHSBACHER C., STAMMINGER M.: Reflective shadow maps. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2005), ACM, pp. 203–231.

[Dut03] DUTRÉ P.: *Global Illumination Compendium*. Katholieke Universiteit Leuven, 2003.

[is96] ID SOFTWARE: Quake, June 1996. `http://www.idsoftware.com/games/quake/quake/`.

[is08] ID SOFTWARE: id software, June 2008. `http://www.idsoftware.com`.

[Jen96] 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.

[Kel97] KELLER A.: Instant radiosity. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), ACM Press/Addison-Wesley Publishing Co., pp. 49–56.

[LB00] LANGER M. S., BÜLTHOFF H. H.: Depth discrimination from shading under diffuse lighting. *Perception* (2000), 649–660.

[LSK*07] LAINE S., SARANSAARI H., KONTKANEN J., LEHTINEN J., AILA T.: Incremental instant radiosity for real-time indirect illumination. In *Proceedings of Eurographics Symposium on Rendering 2007* (2007), Eurographics Association, pp. xx–yy.

[Mit07] MITTRING M.: Finding next gen Ű cryengine 2. In *SIGGRAPH 2007* (2007).

[SA07] SHANMUGAM P., ARIKAN O.: Hardware accelerated ambient occlusion techniques on gpus. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2007), ACM, pp. 73–80.

[SIMP06] SEGOVIA B., IEHL J.-C., MITANCHEY R., PÉROCHE B.: Bidirectional instant radiosity. In *Proceedings of the 17th Eurographics Workshop on Rendering, to appear* (2006).

[SKS02] SLOAN P.-P., KAUTZ J., SNYDER J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), ACM, pp. 527–536.

[ST90] SAITO T., TAKAHASHI T.: Comprehensible rendering of 3-d shapes. *SIGGRAPH Comput. Graph. 24*, 4 (1990), 197–206.

[TL04] TABELLION E., LAMORLETTE A.: An approximate global illumination system for computer generated films. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), ACM, pp. 469–476.

[VG97] VEACH E., GUIBAS L. J.: Metropolis light transport. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), ACM Press/Addison-Wesley Publishing Co., pp. 65–76.