Monte Carlo Ray-tracing

Advanced Global Illumination And Rendering (TNCG15)

Sathish Kottravel (satko730)

January 2014

Linköping University

Abstract

Photorealistic image generation is a paradigm that has various applications in the field of computer graphics such as film and visual effects, architecture, computer games etc.,. In computer graphics photorealistic image generation is achieved using various types of algorithms. These algorithms intend to solve rendering equation which derives its basis from light transport formulation. Global illumination has become key component of such algorithms as it considers both direct lighting (direct illumination) from the light sources and the contribution of light bounced from other surfaces lit in the scene (indirect illumination). Many methods have been developed to estimate the rendering equation such as Whitted ray tracing, Radiosity, and Monte Carlo path tracing. The aim of this report is to briefly discuss some those illumination techniques. Furthermore, this report describes the implementation of Monte Carlo method for rendering realistic images. In addition some CPU acceleration techniques are briefly outlined.

1 Introduction

Global illumination aims to solve rendering equation 1.0 [1] in order to produce photorealistic image of a scene. Consider a point p on a surface of an object, which reflects light in an outgoing direction ωo (possibly in the direction of camera), the exitance radiance $L(p, \omega_o)$ is given by:

$$L(p, \omega_o) = L_e(p, \omega_o) + \int_{s^2} f(p, \omega_o, \omega_i)\, L_i(p, \omega_i) \left| cos\ \theta_i \right| d\omega_i \qquad (1.0)$$

where $L_e(p, \omega_o)$ is the self emission, $f(p, \omega_o, \omega_i)$ is the BRDF of the surface point which depends on the material, $L_i(p, \omega_i)$ is the incident radiance arriving from all directions ω_i on the sphere S_2 around point p. Various algorithms have been developed in order to estimate the rendering equation. In this section basic idea of as Whitted ray tracing, radiosity, Monte Carlo ray tracing and other algorithms are discussed in detail.

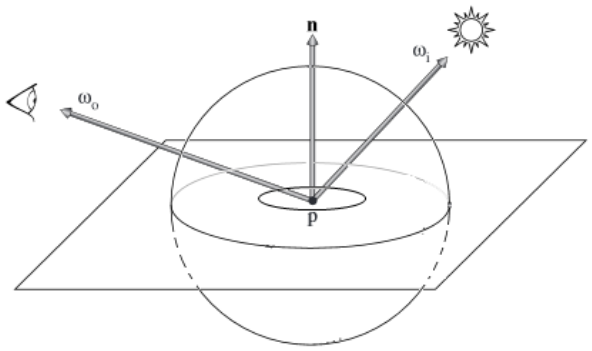

Figure 1: Exitance radiance from point p with surface normal n.

1.1 Whitted ray tracing

Whitted ray tracing [2] is a recursive method for evaluation radiance along reflected and refracted ray directions. It is one of the early methods proposed to solve rendering equation with higher emphasis on perfect reflection and refraction. This method is often called as backward ray-tracing because the rays are traced from camera to the light source. If a ray hits a reflective surface then it reflects in certain direction that makes this scheme recursive. Similarly for refractive surface. But if a ray hits diffuse surface then the radiance of the surface point point is determined by casting ray to the light sources in the scene (shadow rays) to determine if the point is in shadow. This kind of setup is known as local lighting model which takes only direct illumination into account and also often results in hard shadows. Also the recursive process is terminated as soon as the ray hits diffuse surface.

1.2 Radiosity

In Whitted ray tracing indirect illumination is not taken into account. Radiosity [3] method overcomes this problem by considering diffuse interreflections which results in more accurate illumination. This methods accounts for modeling of interaction of light bounced from surfaces that acts as diffuse reflector (Lambertian model). This method is based on heat transfer and hence it is suitable for rendering diffuse illumination. In this method, surface elements in a scene is divided into several tiny patches and light transport among those patches are calculated. For example consider a surface S_i with emissivity E_i, whose total of energy incident sums to B_i that can be written as follows:

$$B_i = E_i + \rho_i \sum B_j F_{ij} \qquad (1.1)$$

where ρ_i is reflectivity of surface i and B_j is the radiosity of another surface j and F_{ij} form factor of the surface j relative to surface i. The above equation (1.1) can be computed for every patch in the scene, in other words n simultaneous solution should be computed. The form factor usually considers geometric relations between two patches. It can be defined as fraction of energy that leaves from one surface to another surface with geometric realization. Such form factor is independent from changing viewpoint and other surface attributes.

1.2 Monte Carlo ray tracing

Monte Carlo ray tracing [4] gains its advantage from its generic approach to solve rendering equation regardless of type of a geometry with different kinds BRDF and its ability to be scaled to multiple dimensions. Monte Carlo ray tracing is often known as Stochastic ray tracing. This method computes an estimate of integral of in the rendering equation by exploiting the properties of randomness. This is done by drawing samples using a probability distribution function which accelerates the convergence rate. For example, if n random samples are used to estimate the integral of a function, this method converges at the rate of $O(n^{-1/2})$. In other words to reduce the error by half, its enough to evaluate with four times as many samples.

Consider one-dimensional integral $\int_a^b f(x)\,dx$ and uniform random variables $X_i \in [a, b]$, then expected value of the estimator is :

$$F_N = \frac{b-a}{N} \sum_{i=1}^{N} f(X_i) \qquad (1.2)$$

where N is the number of random samples, F_N is the estimated approximation of the given integral. In the above equation variance can be reduced by drawing random variables from some arbitrary PDF $p(x)$, then the estimator (1.2) can be written as

$$F_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)} \qquad (1.3)$$

Hence choosing appropriate sampling methods plays an important role in faster convergence. Also sampling methods affects the bias of the final result. To achieve unbiased results importance sampling can be used for sampling the BRDF of the objects. Importance sampling is discussed in later sections

1.3 Two-pass rendering method

Two pass rendering [5] is a technique in which a rough approximation to the global illumination of the scene computed in the first pass and computed result is first pass used in the second pass to render the final image. Radiosity or photon mapping can be used in the first pass. In the second pass the specular effects are captured using ray tracing which is view dependent technique such as distributed ray tracing. While radiosity is for diffuse effects which is independent of view. Hence the method to simulate global illumination in complex scenes with broader use of of light simulation often with progressive refinement.

1.3 Photon mapping

In Photon mapping [6] technique when a photon emitted from the light source, it is traced into the scene using photon tracing. When a photon hits an object it can be either reflected or absorbed. Such state of photon is determined by probabilistic methods based on the material of the surface. To determine such interaction type often Russian roulette is used. Initially Photon mapping was proposed for surfaces without participating media. Photon mapping is a two pass method. In the first pass, two photon maps are generated, they are a caustics photon map and the other one is global photon map. To generate caustics photon map, cast photons photons only towards the specular surfaces in the scene and they are stored as they hit diffuse surfaces. While constructing global photon map, photons are emitted towards every objects visible in the scene. When a photon hits a surface at first intersection point store illuminated photon and in the subsequent intersection point store shadow photon. In the second pass, the buffered preprocess information are used to render out images. Normally this pass is divided into four parts as shown in equation 1.4.

$$L_i(p,\omega_o) = L_{i,1}(p,\omega_o) + L_{i,d}(p,\omega_o) + L_{i,s}(p,\omega_o) + L_{i,c}(p,\omega_o) \qquad (1.4)$$

The first term in above equation 1.4 is direct illumination computation that can be estimated by tracing a ray from the point of intersection to each light source to check the visibility of the intersection point. If there is no intersection is detected with other objects, the light source is used to calculate the direct illumination. The precomputed information in global photon map in first pass can be used to reduce the shadow rays. Hence shadow rays are casted when the closest photons in global photon map contains direct illumination and shadow photons. The second term is diffuse indirect illumination that can be computed using Monte Carlo ray tracing. The third term is specular illumination. The last is caustic illumination. Last two terms are computed by estimating the radiance based on the photon maps. Often acceleration data structures such as octree is used for scene decomposition since photon mapping is computationally intensive.

1.3 Ray-tracing of Iso-surfaces

Iso-surface visualization has created a great impact on simulation applications. Ray tracing of iso-surfaces has made it possible to render and visualize a complex set of data with high image quality and with global effects. In direct ray tracing methods initially an iso-surface is extracted by computing the intersection of rays with the provided implicit function. That is, by computing the intersection of a ray with implicit function $f(x,y,z) = c$, where c is required iso value. Later every visible intersection point is subjected to illumination. Often Iso-surfaces are essential to understand the distribution of scalar or vector values, for example volumetric data. Also it is an alternative to expensive isosurface extraction. Direct illumination of iso surfaces computation is highly parallel in nature. Hence it is common practice to use GPU acceleration for ray tracing iso-surfaces. Often spatial data structures such as octree and kd-tree [7] are used to speed up iso surface rendering using techniques such as out-of core data caching methods.

In this section we discussed different methods of global illumination that can be used to generate realistic images has been discussed briefly. Rendering equation has been introduced and followed by Whitted ray tracing method, which is a common ray tracing method. Later radiosity has been introduced which is also one of early methods. Various recent developments in global illuminations such as Monte-carlo, Two-pass rendering, Photon mapping and Iso surface ray tracing has been discussed. In the following section 2, implementation of Monte Carlo ray tracing done for this project has been explained in detail. While the results obtained using this implementation has been presented in the section 3. The section 4 contains discussion about results obtained and future enhancements. The references used in this report can be found at the end of this report.

2 Background

In this section, my implementation of global illumination algorithm based on Monte Carlo ray tracing has been discussed. This project is implemented using C++ and simple Cornell box scene is used for demonstration. GLM external library is used for handling data structures such as 3d vectors and matrices. This library also provides some convenience functions such as dot product, cross product, transformations, vector normalization etc., Monte Carlo approximation is used for indirect lighting computation.

2.1 Ray-Tracing Setup

To render an image it is necessary to compute the radiance of each pixel in an image plane. In our ray tracing setup famous Cornell box setup has been used. Camera is placed inside the Cornell box. Camera has eye position and view direction. Image plane lies perpendicular to view direction and slightly in front of camera eye position. Care must be taken to place the camera completely inside the box. Failure to do so will often result in rendering of front face of cornell box. For convenience box is closed in all six faces so that rays does not escape out of the box otherwise it results in less light contribution from the scene.

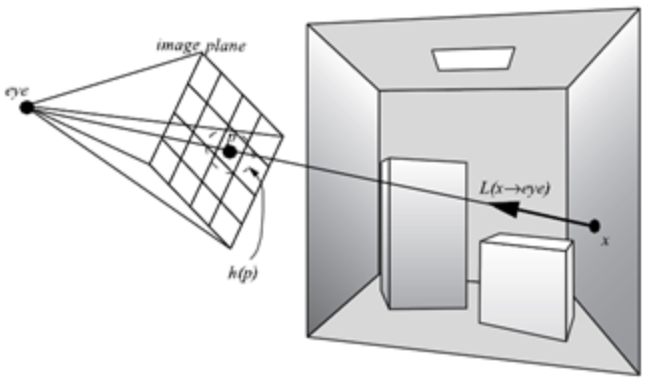

Figure 2: Scene setup with image plane. Image from Dutre, Philip, et al. Advanced global illumination.

A ray bounced from the scene object, passing through the image plane and reaching the eye can be represented as L_{pixel} :

$$L_{pixel} = \int_{imagePlane} L(p \rightarrow eye)\, h(p)\, dp \quad = \int_{imagePlane} L(x \rightarrow eye)\, h(p)\, dp \qquad (2.0)$$

where p is a point on the image plane, and $h(p)$ is filtering function [8]. In this implementation simple box filter is used which is computed by averaging all incident radiance value over the area of the pixel.

In image plane each pixel located in the middle of $n * n$ squares (or samples) where each square has dimension of *(1/n*pw, 1/n*ph)* where *pw* and *ph* are pixel width and height. The points have been chosen randomly in each squares which leads to stratified sampling with jittering. To evaluate the rendering equation a ray is cast from the camera eye that pass through jittered point sample *p* on the image plane as shown in Figure 2.

In this project each pixel has four ($2 * 2$) jittered sample points [9]. In addition scene can support multiple area lights. Each light is presented as circular disk for convenience. Four types of geometric primitives are support in this implementation, they are sphere, disk, cylinder and infinite plane. Infinite planes are used to construct walls of the cornell box.

2.2 Materials

The scene consists of objects with three types of materials such as purely diffuse, purely specular and refractive materials. In addition each objects can have color and emittance value. Light objects acts as luminaires object. Hence emittance value is set only for lights and color of the light objects are set as always zero. Non-luminaire objects always have emittance value zero. Refractive index has been set to 1.52 after experimenting with several values which is approximately equal to the refractive index of glass.

2.3 Intersections and Normals

All objects used in this project are represented in the form of equation. No mesh representation is used. Mesh geometry representation is left for future enhancements. Since all objects are represented in the form of equation, object-ray intersection can be done faster. Also it is necessary to know normal at intersection point to compute illuminations. Four types of object-ray intersections are implemented in this project. They are: Ray-Sphere, Ray-Cylinder, Ray-Circle and Ray-Infinite-Plane intersections.

The parametric form of a ray *R* with origin *o,* direction *d* and parameter *t* can be represented as:

$$R(t) = o + t\,d,\ t \geq 0 \qquad (2.1)$$

In equation 2.1 negative *t* represents intersection in the direction opposite to t which is often ignored. And in case of no intersection $|t| \leq tolerance$. $tolerance = 1e-4$ is a very small value used for numerical approximation of zero.

2.3.1 Ray-Sphere Intersection and intersection normals

Let us assume sphere have center $c = (c_x, c_y, c_z)$ and radius r, then the vector equation of the sphere can be represented as follows:

$$(p - c).(p - c) = r^2 \quad (2.2)$$

where p is a point on sphere. By substituting ray equation 2.1 in the sphere equation 2.2 will result in the following quadratic equation:

$$(d.d)t^2 + 2d.(o - c)t + (o - c).(o - c) - r^2 = 0 \quad (2.3)$$

Equation 2.3 is of the form $At^2 + Bt + C = 0$ .Solving the above quadratic equation results two values for t, which means ray intersection with sphere results in two intersection points. In such cases intersection point that is closest to the ray origin will be chosen. Also if ray is tangent or no intersection then $B - 4AC < 0$. In this implementation additional small offset is added to the t value in order to keep intersection points slightly outside the sphere in order to avoid ray being trapped inside specular objects due to small numerical precision errors. Such ray trapping can result in infinite loop. This implementation can detect such infinite recursive ray tracing and display warning messages.

Normal N_p at any intersection point p can be represented as vector normalized vector between center of the sphere and intersection point.

$$N_p = c - o + t\,d \quad (2.4)$$

2.3.2 Ray-Cylinder Intersection and intersection normals

Ray cylinder intersection is slightly complex than all other intersections implemented in this project. For every cylinder intersection, three intersection tests are performed. First intersection is performed on infinite cylinder which has axis extending along bottom center to top center of the cylinder. And if intersection is successful, further intersection test is performed on top and bottom cylinder cap (which are basically disks). Top and bottom cap can be considered as clamping plane for infinite cylinder. Ray-Cylinder intersection from [10] has been used in this project as reference.

This also make normal computation equally complex. Normal computation for infinite cylinder and cylinder caps are implemented specially for this project. In order to compute normal, three cases have to be detected:

case 1: Perform ray intersection on infinite cylinder
case 2: Perform Intersection on top cap of cylinder
case 3: Perform Intersection on bottom cap of cylinder

On case 1, the intersection point on the cylinder is determined. Then the intersection point is projected on to the cylinder axis. The distance between projected point and intersection point is always equal to radius. Also normalized vector between projected point and intersection point represents the normal to the cylinder surface. If there is no intersection detected on infinite cylinder, then it is not required to perform case 2 and case 3.

After performing intersection on infinite cylinder, if successful, the top and bottom cap intersection is performed. Always shortest intersection points from the ray origin should be chosen as intersection point. The bottom cap normal is nothing but is vector between bottom center and top cylinder and vice versa for top cap.

The special case is, sometimes ray enters the cylinder at top cap and exits at bottom cylinder cap or vice versa. This case is also handled.

2.3.2 Ray-Infinite-Plane Intersection and intersection normals

Let us assume that we have an infinite plane with normal vector n and a known point on plane p_0 the vector equation of the plane with intersection point p can be expressed as

$$(p - p_0).n = 0 \quad (2.5)$$

Upon ray-plane intersection, we can substitute ray equation 2.1 in plane equation 2.5 which will result in the following equation:

$$(td + o - p_0).n = 0 \quad (2.6)$$

Equation 2.6 can be further simplified as :

$$t = \frac{(p_0 - o).n}{d.n} \qquad (2.7)$$

In order to avoid division by zero $d.n$ is computed first. In other words it checks for condition if plane is almost parallel to ray by finding dot product between d and n. Thus we can safely determine division by zero cases. Normal is specified during creation of plane and it is uniform all over the plane at any intersection point.

2.3.2 Ray-Disk Intersection and intersection normals

To simplify circle intersection, Ray-Plane intersection can be reused. That is , after plane intersection, compute the distance between resulting plane intersection point and center of the disk. If the distance between them is less than or equal to radius of the circle, then the ray intersects with the circle. For this reason, *Disk* class inherits from *InfinitePlane* class.

Similarly normal at every intersection point on a disk is uniform over the disk plane. Normal is specified during creation of disk.

2.4. Radiance computation

The radiance estimate from rendering equation can be divided into two parts: direct illumination and indirect illumination. Direct illumination considers contribution of light from light sources. While indirect illumination considers contribution of light bounced from the other surfaces in the scene. The following equation represents the reflected radiance as in [8]:

$$\begin{aligned} L_r(p,\omega_o) &= \int_{s^2} L_e(r(p,\Psi) \rightarrow -\Psi) f_r(p,\omega_o \leftrightarrow \Psi) \cos(\Psi, N_x)\, d\omega_\Psi \\ &\quad + \int_{s^2} L_r(r(p,\Psi) \rightarrow -\Psi) f_r(p,\omega_o \leftrightarrow \Psi) \cos(\Psi, N_x)\, d\omega_\Psi \\ &= L_{direct\,(p \rightarrow \omega_o)} + L_{indirect\,(p \rightarrow \omega_o)} \qquad (2.8) \end{aligned}$$

where integral is done over the hemisphere S^2 . The ray tracing starts by shooting the rays towards the scene from the camera eye position. When nearest intersection point of the ray in the scene is found sum the contribution of the direct and indirect illumination at that point and recursively continue to shoot another ray until all sample points of the image plane have been visited. The direct and indirect light computation is explained in the following sections.

2.4. Direct Illumination

Let x be the closest intersection point of any ray projected into the scene. The direct illumination in equation (2.8) considers only the direct contribution of light sources to point p and also the term $L_e(r(x, \Psi) \rightarrow -\Psi)$ will be non-zero at the light source, hence we can transform the hemispherical integral into an integral over the area of the all light sources as shown figure 3 (only difference in our scene setup is disk light sources are used). This is represented in following equation 2.9

$$L_r(x, \omega_o) = \int_{A_{sources}} L_e(y \rightarrow \overline{yx}) f_r(x, \omega_o \leftrightarrow \overline{xy}) G(x,y) V(x,y) dA_y \quad (2.9)$$

where x is the point of intersection of ray from camera, y is random sample on light source surface. N_x *and* N_x are surface normals at point x and y. $G(x,y)$ is the geometric coupling factor that represents relation between the intersection points. $V(x,y)$ is the visibility of the points x and y . V = 0 if the ray between point x and y is occluded by other other objects, otherwise it is 1.

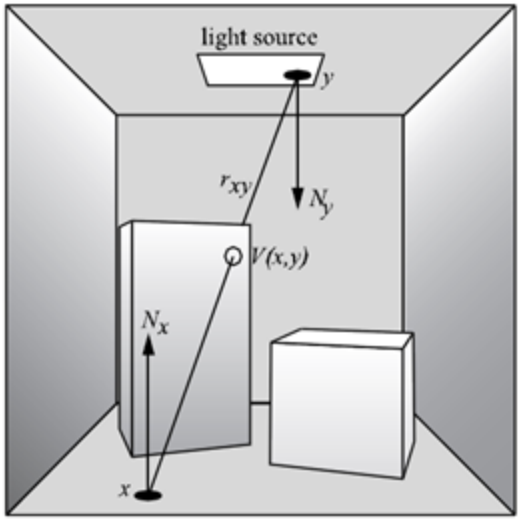

Figure 3: Direct illumination at point x. Image from Dutre, Philip, et al. Advanced global illumination

While computing direct illumination, the radiance at point x, we cast shadow rays from the intersection point to the light sources. Hence we draw random sample points from the surface area of all the light sources using uniform distribution function. Applying Monte Carlo integration to approximate equation 2.9 with shadow rays leads to the following estimator in equation 2.10

$$L_{direct}(x,\omega_o) = \frac{N_L}{N} \sum_{i=1}^{NS} A_{Lk}\, L_e(y_i \rightarrow \overline{y_i x})\, f_r(x,\omega_o \leftrightarrow \overline{xy_i})\, G(x,y_i)\, V(x,y_i)\, dA_{y_i} \quad (2.10)$$

where

$$G(x,y_i) = \frac{(N_x, \Psi)\ (N_{yi}, -\Psi)}{r_{xyi}^2} \quad (2.11)$$

where NS is the shadow ray count, N_L is the number of light sources, A_{Lk} i s the area of the k light sources in the scene, L_e is the emittance value from the light source, f_r is the BRDF of the surface, G is the geometry term as in equation 2.11 and V is the visibility term. Note that integral is performed over surface area of all the light sources. And finally Ψ represents all vectors between our intersection point x and sampled points on the lights y_i. Having known all the light object information in the scene, we can easily compute all the required terms and there thus we evaluated the Monte Carlo estimator for indirect illumination.

2.5. Indirect Illumination

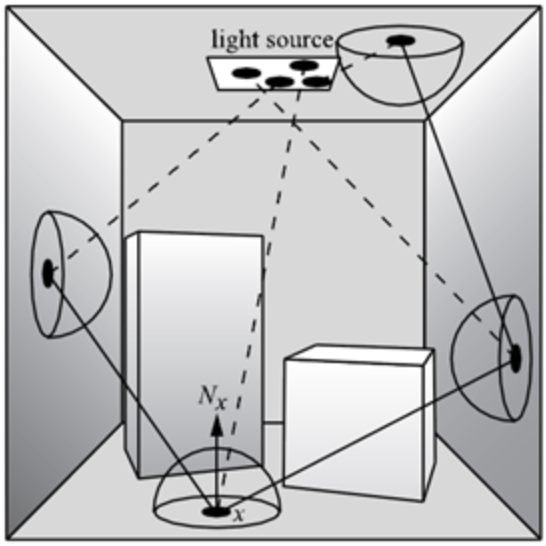

Figure 4: Indirect illumination at point x. This image represents recursive nature of indirect illumination. Dotted line represents shadow rays. Image from Dutre, Philip, et al. Advanced global illumination

Indirect illumination at an intersection point x is computed after at least one reflection at an intermediate surface between the light sources and x. The indirect illumination term in the equation 2.9 consists the radiance bounced from other surface points of the scene. The bounced radiance contains both direct and indirect illumination. Hence this indirect illumination computation is often done recursively. Applying

Monte Carlo integration to indirect part of equation 2.9 leads to the following estimate for indirect illumination computation:

$$L_{indirect}(x, \omega_o) = \frac{1}{N} \sum_{i=1}^{N} \frac{L_r(r(x,\Psi_i)\rightarrow -\Psi_i)\, f_r(x,\omega_o \leftrightarrow \Psi_i)\, cos(\Psi_i, N_x)}{p(\Psi_i)} \quad (2.12)$$

Sample direction Ψ_i is generated by using importance sampling of the Phong Reflectance Model. Importance sampling has been used to reduce the variance of the random samples drawn to get a better estimate. After tracing sample direction from point x, radiance is evaluated as $L_r(r(x, \Psi_i) \rightarrow -\Psi_i)$ at the nearest intersection point $r(x, \Psi_i)$. This shows the recursive nature of indirect illumination as you can see in figure 4.

2.6. Importance Sampling

Any objects surface can be modeled as material mixture of diffuse and specular reflection. This model is know as Phone Reflectance model. In this project BRDF based on the Phong shading model [11] has been used. Hence the reflectance distribution of a is divided into a diffuse and specular component:

where α is the angle formed between specular direction and the outgoing ray direction. k_d is the diffuse reflectivity, k_s is the specular reflectivity and n is the specular exponent commonly used in Phong models. Russian roulette [8] to determine whether to compute diffuse or specular component. This is done by following steps:

1) Generate a random variable with uniform distribution function $r \in [0, 1]$.
2) If r is smaller than k_d we take a diffuse sample and compute its contribution.
3) If r is bigger than k_d and smaller than $k_s + k_d$ then a specular reflection or refraction is computed.
4) If r is bigger than $k_s + k_d$, the ray is absorbed and the recursion is terminated.

The diffuse component of Phong brdf scaled by the cosine of incoming angle gives the pdf as

$$pdf(\theta_i) = \frac{1}{\pi} cos(\theta_i) \quad (2.13)$$

This pdf can be sampled by selecting two uniform random variable r_1 and r_2 over the interval $[0, 1]$. This pdf can be represented in spherical coordinate system as in equation 2.14.

$$(\theta, \Phi) = (arccos(\sqrt{r_1}),\ 2\pi r_2) \quad (2.14)$$

θ *and* Φ are Similarly specular component of Phong brdf has following distribution as in equation 2.15 and corresponding spherical coordinate transformation as in equation 2.16 , where *r1* and *r2* are two uniform random variables over the interval $[0, 1]$.

$$pdf(\theta_i) = \frac{n+1}{2\pi} cos^n(\alpha) \quad (2.15)$$

$$(\theta, \Phi) = (arccos(r_1^{\frac{1}{n+1}}), 2\pi r_2) \quad (2.16)$$

The diffuse and specular samples in equation 2.14 and 2.16 are in hemispherical coordinates, while the cartesian representation of the same is as follows:

$$(x, y, z) = (sin(\theta)\, cos(\Phi),\ sin(\theta)\, sin(\Phi),\ cos(\theta)) \quad (2.17)$$

Necessary co-ordinate transformations are performed on the cartesian representation in equation 2.17 to make the sample points align with basis vectors generated from normal direction of the intersection point. The refraction rays are computed using Snell's law. If ray enters from higher refractive index to lower refractive index it should account for total internal reflections. Thus generating two random variables for each method, we can obtain the random direction vector over the hemisphere.

3 Results and Benchmark

The scene is completely created using code in C++. No external mesh geometry has been used for this demonstration. Different scene settings has been tested in this project such as diffuse only, indirect illumination only, multiple lights, region rendering and colored lights. Also CPU parallelization has been done per pixel using OpenMP which does not take load balancing into account. In addition, easy inspection of specific region on image plane has been implemented and supports rectangular region rendering as shown in the Figure 10. All images are rendered at 512x384 resolution.

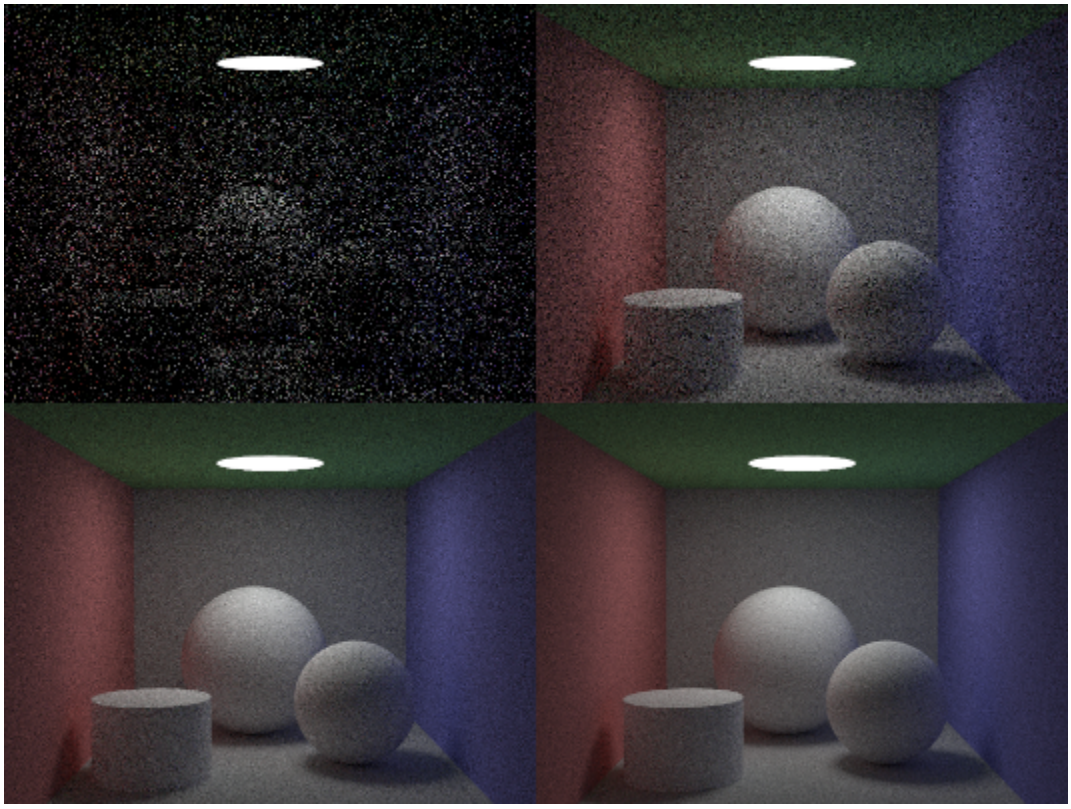

Figure 5: Computation of indirect illumination using 1 ray per pixel-sample (top-left) , 32 rays per pixel-sample (top-right), 128 rays per pixel-sample (bottom-left) and 256 rays per pixel-sample (bottom-right).

Figure 5 illustrates a scene containing diffuse objects only with indirect illumination. This test is performed to get understanding of rate of reduction in noise level in the images as rays per pixel-sample increased and also it can be observed that color bleeding is becoming more visible. As the ray per pixel rate is increased the noise level in the image started decreasing, thereby reaching convergence. Light is a luminaire disk object which is the brightest of all objects due to self emittance component. Also some soft shadow effects can be observed at this stage. The computation time of diffuse only indirect illumination is represented in the Figure 9.

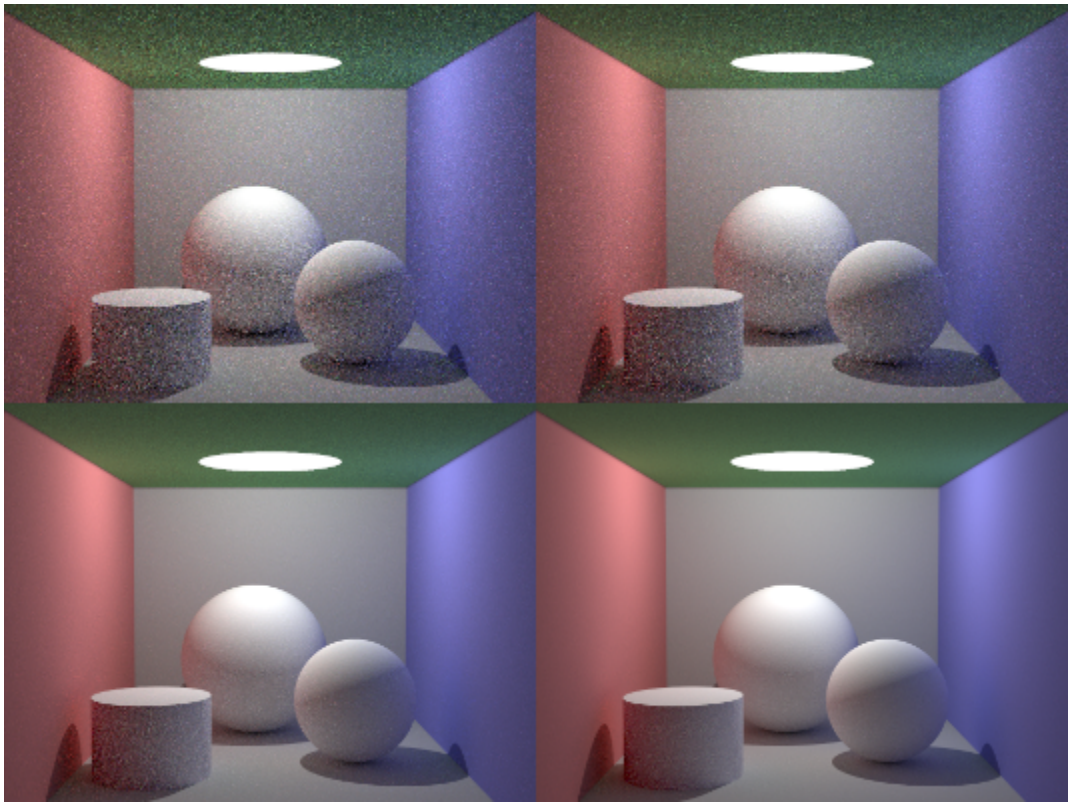

Figure 6: Computation of direct illumination using 1 ray per pixel-sample (top-left) , 32 rays per pixel-sample (top-right), 128 rays per pixel-sample (bottom-left) and 256 rays per pixel-sample (bottom-right).

Figure 6 illustrates similar settings to the previous figure 5. But only modification is, it involves computation of direct illumination only with two random shadow rays per light object. By using more shadow rays, the hard shadows can be shifted to soft shadows during direct illumination. In other words image can get better shadows if more shadow rays are generated towards light sources. The variable number of shadow rays are tested in later scene settings. Also time measure has been presented in graph of Figure 8. The computation becomes expensive especially when it is combined with indirect illumination. Figure 7 represents scene with combined direct and indirect illumination with multiple lights. Better color bleeding effects and soft shadows are traits of indirect illumination which is still preserved and blended nicely with the direct illumination.Finally Figure 9 illustrates full scene with all three types of materials in effect. Because of Russian Roulette method an unbiased image is resulted. Pure specular object placed in the scene produced nice reflection of the scene. Hence it is placed at the end of the room. Thus all required results generated in this project are presented.

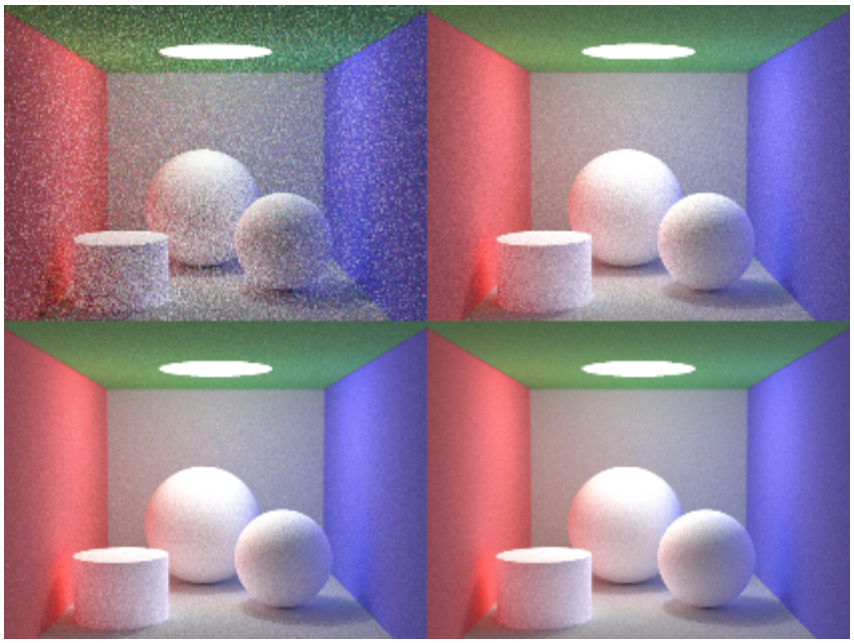

Figure 7: Computation of direct and indirect illumination using 1 ray per pixel-sample (top-left) , 32 rays per pixel-sample (top-right), 128 rays per pixel-sample (bottom-left) and 256 rays per pixel-sample (bottom-right).

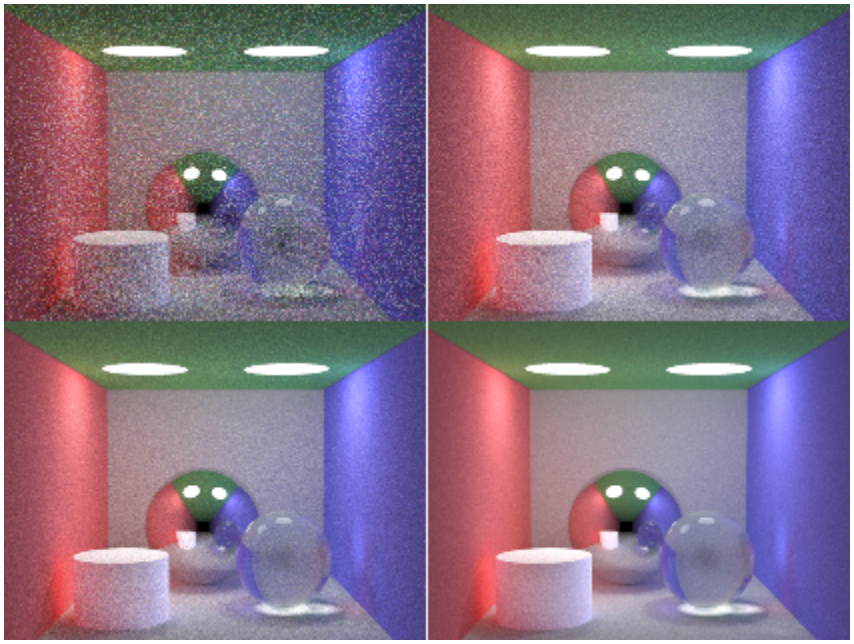

Figure 8: Computation of direct and indirect illumination with multiple lights, all three types of material in effect and using 1 ray per pixel-sample (top-left) , 32 rays per pixel-sample (top-right), 128 rays per pixel-sample (bottom-left) and 256 rays per pixel-sample (bottom-right).

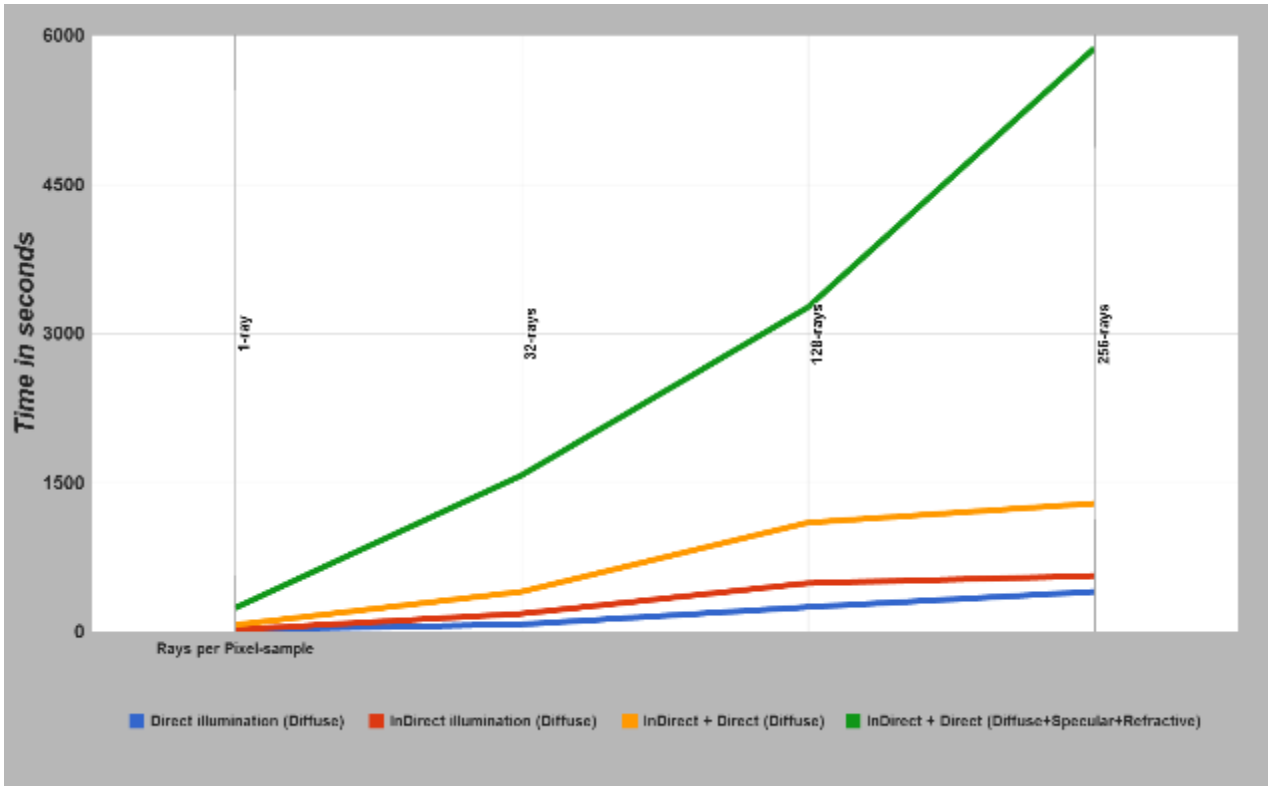

Figure 9: Time measure for scene settings in Figure 5, 6, 7 and 8. Images are rendered in 512x384 resolution. All these settings are tested with 1,32,128 and 256 rays per pixel-sample of the image.

Figure 10: Example region rendering (left). Simple scene with colored lights (right)

4 Discussions

In this project various rendering techniques have been introduced and discussed. In the later section implementation of Monte Carlo renderer have been explained in detail. In the result section, images generated using this implementation have been furnished. Many intermediate results are generated with different settings and furnished with a notion to analyze the effects added at important stages of this Monte Carlo of method. Important and final result is the Figure 7, that combines the direct and indirect illumination. This step is necessary so that rate of convergence can be increased. Figure 8 demonstrates the capability of the implemented renderer to support all three types of material objects such as diffuse, specular and refraction.

In order to extend this project, bidirectional path tracing can be adopted easily with less changes to source code. One of the main disadvantages of Monte Carlo renderer is that it cannot simulate caustics. Also more effective methods can be adopted to reduce the noise. For instance adaptive sampling can result in better image. Also techniques such as final gathering can reduce the noise by blurring the image. Various material models have to be tested. And further, this system can be adopted to import geometry meshes (which would require data acceleration structures such as octree and kd-tree for scene decomposition) and can be subjected to more rigorous benchmark performance analysis.

Some implementations often enable caching to avoid duplications in random number generation. But for simplicity such methods are not used in this implementation. Also during recursive tracing, often due to some numerical errors or improper scene setting, infinite looping occurs. This can be avoided by checking current depth of recursion with allowed maximum depth. This check was very useful to catch potential bugs. Though the current implementation uses CPU parallelization, there is plenty of room to optimize the implementation further both in CPU and GPU. But it would require additional changes in data structure organization. Often hybrid CPU and GPU acceleration are suitable for algorithms such as raytracing. Such adaptations will be the immediate step for future enhancement. Also recursive nature of algorithm will be the first barrier that one should overcome in order to adopt parallel optimizations. Current project supports region rendering but only one region at a time. Adding multiple region rendering can be useful feature for debugging.

The overall result of the project is satisfying because the core rendering algorithm is stable and have produced expected results. And also the project is implemented in C++ with some sense of objected oriented programming exploiting polymorphism. Hence the project can be extended further easily.

5 References

1. James T. Kajiya. 1986. The rendering equation. In Proceedings of the 13th annual conference on Computer graphics and interactive techniques (SIGGRAPH '86), David C. Evans and Russell J. Athay (Eds.). ACM, New York, NY, USA, 143-150.
2. Turner Whitted. 1980. An improved illumination model for shaded display. Commun. ACM 23, 6 (June 1980), 343-349.
3. Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. 1984. Modeling the interaction of light between diffuse surfaces. In Proceedings of the 11th annual conference on Computer graphics and interactive techniques (SIGGRAPH '84), Hank Christiansen (Ed.). ACM, New York, NY, USA, 213-222.
4. Philip Dutre, Henrik Wann Jensen, Jim Arvo, Kavita Bala, Philippe Bekaert, Steve Marschner, and Matt Pharr. 2004. State of the art in Monte Carlo global illumination. In ACM SIGGRAPH 2004 Course Notes (SIGGRAPH '04). ACM, New York, NY, USA, Article 5.
5. F. Sillion and C. Puech. 1989. A general two-pass method integrating specular and diffuse reflection. In Proceedings of the 16th annual conference on Computer graphics and interactive techniques (SIGGRAPH '89). ACM, New York, NY, USA, 335-344.
6. Henrik Wann Jensen, Global illumination using photon maps, Proceedings of the eurographics workshop on Rendering techniques '96, p.21-30, December 1996, Porto, Portugal.
7. Wald, I.; Friedrich, H.; Marmitt, G.; Slusallek, P.; Seidel, H.-P.; , "Faster isosurface ray tracing using implicit KD-trees," Visualization and Computer Graphics, IEEE Transactions on , vol.11, no.5, pp.562-572, Sept.-Oct. 2005.
8. Philippe Bekaert Philip Dutre, Kavita Bala. Advanced Global Illumination 2nd edition. AK Peters, 2006.
9. Peter Shirley, R. Keith Morley. Realistic Ray Tracing 2nd edition. AK Peters, Natick Massachusetts, 2003.
10. Arvo, James, ed. Graphics Gems Two. Vol. 2. Morgan Kaufmann, 1991.
11. E. Lafortune and Y. Willems. Using the modified Phong reflectance model for physically based rendering. Technical Report CW197, Dept. Comp. Sci., K.U. Leuven, 1994.