Global Illumination with Path Tracing

SIDDHARTH SUNDAR, Indraprastha Institute of Information Technology
Y S RAMYA, Indraprastha Institute of Information Technology

Photorealistic techniques serve as the pinnacle of Computer Graphics techniques. One of the techniques that give the most realistic results is Path tracing - often used as a benchmark for images produced by other means. In this paper, we talk about our project implementing a path tracer.

Additional Key Words and Phrases: Computer Graphics

1 INTRODUCTION

Image or scene Rendering is the process of generating an image by means of computer programs. The computer program used to simulate the image contains complex data structures holding geometry, viewpoint, texture, lighting, and shading information as a description of the virtual scene. The data contained in these structures is then passed to a rendering program to be processed and to output the final raster graphics image file.

For photo-realistic rendering, the rendering software utilizes needs to solve the rendering equation, first introduced simultaneously in computer graphics by David Immel et al. and James Kajiya. The following sections outline one solution for the rendering equation, namely, Path Tracing[]

2 GLOBAL ILLUMINATION

2.1 Literature Survey

We see things because the light from our surroundings, enters our eyes where this illumination information is decoded by our brain to interpret the scene. Here, the light that enters our eyes after only one reflection is called direct illumination. Light that reaches us after repeated reflections from objects is called indirect illumination. Everyday, we observe a combination of these two phenomena. Thus, to simulate a photo-realistic scene, a combination of illuminations is required. Global illumination involves simulating both indirect and direct illumination.

Efficient and realistic Global Illumination is one of the premier problems of Computer Graphics. This is due to the fact that there is no one generic path taken by light rays. Light rays interact with different materials in different ways. A light ray may bounce off of an object, get refracted at the surface, get scattered partially, or be entirely absorbed into the the object. There are infinitely many combinations.

There are two methods possible to trace light rays in a virtual scene,

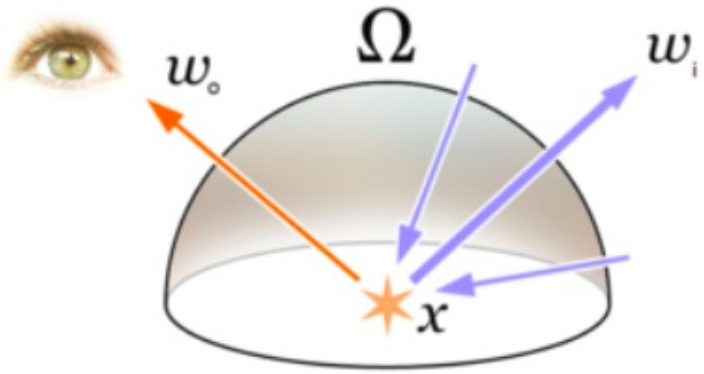
(i) start from the light source and trace the ray's route til it reaches the eye. This method is called Forward tracing

(ii) trace a ray entering your eye backwards,i.e, shoot a ray from the eye and follow it's path til it reaches the light source it originated from. This method is called Backward tracing

Forward tracing is not efficient, since not all rays originating from the light source enter the eye. Backward tracing is preferred. However, while backward tracing is an efficient way of computing direct lighting, it is not an efficient way of simulating indirect lighting. Simulating indirect lighting is simulating the path taken, by a light ray from the time it is emitted by the light source to the time it enters our eyes . Hence, the algorithms that simulates this path is called light transport algorithms.

Rendering Equation and Light Transport Algorithm

The rendering equation is based on the law of conservation of energy. It is an integral equation which states that the energy leaving a point is given by the sum of the energies emitted and reflected radiance from that point under geometric approximation.



$$L_o(p, \omega_o) = \int_\Omega f(p, \omega_o, \omega_i)\, L_i(p, \omega_i)\, \cos\theta_i \, \mathrm{d}\omega_i$$

This integral is hard to solve analytically. However, we can obtain a good approximation through the use of Monte Carlo methods. In Monte Carlo methods, we approximate the integral by sampling the integrand at many locations according to a distribution. Then, we add them up weighting them by the inverse of the probability of that location being chosen. More the number of samples, the more closer the sum is to the actual value of the integral. This method is also called unbiased because as the number of samples tends to infinity, we get the actual value of the integral. Biased methods sacrifice this property to achieve better results with fewer samples.

The integral as it is approximated is given by:

$$\langle L_o(p, \omega_o) \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{f(p, \omega_o, \omega_i)\, L_i(p, \omega_i)\, \cos\theta_i}{p(\omega_i)}$$

The final image quality of a render is determined by the Sample Per Pixel (SPP), which is an effect of the Monte Carlo approximation. Greater the samples per pixel, more refined the image. This means that more the light rays are shot out from that pixel, more color values to approximate the color of the object. The quality of the image can be improved until a point from where you get diminishing returns and can no longer tell the difference between an image with 5,000 SPP and 30,000 SPP.

Difference between Path and Ray Tracing

Ray tracing and Path tracing are two popular rendering algorithms. Both involve tracing the path of a light ray in the scene using backward tracing. These two algorithms, however, produce two very different renders.

In Ray tracing, the light rays are shot into the scene from the camera and wherever they intersect calculate the color of the pixel based on the material's properties. In case the material is transparent, they shoot off further based on the refractive angle or reflect from objects. However, This results in ray tracing only computing direct lighting. All other effects,i.e., its interaction with water(caustics), glass, soft shadows and other indirect lighting have to be calculated separately.

In Path tracing, light rays travel around the scene, splitting into multiple rays or bouncing off materials' or interacts with depending on their properties(water, diamond, glass, etc). When these rays finally reach the light source they get the important piece of the rendering equation, the initial energy of the light ray. The amount of light transferred to the pixel, is calculated. This information is obtained by many rays shooting from one pixel which is then averaged out to get the final pixel colour value. This is a brute force method for calculating the rendering equation and requires a lot of compute power as opposed to ray tracing.

Present Work

The basic Monte Carlo Path tracing has been modified to render sharper and more resolution images. Presently, methods such as Bi-directional path tracing and Metropolis Light transport are employed.

In Bidirectional path tracing, indirect light can be sampled not only directly from the eye to the light source but can also sample by connecting the eye subpath vertices to any light sub path vertices, even though it is not directly visible.

Metropolis light transport improves Bidirectional path tracing in that it can also render phenomena like caustics. Caustics can can only be rendered with either pure backward tracing(from eye) or pure forward tracing(from light source). Metropolis algorithm can render this effect by exploring similar paths once it finds the right path without any bias.

3 IMPLEMENTATION DETAILS

3.1 Path Tracing Algorithm: Diffuse and Reflecting Materials

The generalised pseudocode followed is given in Fig 1. The path tracer supports rendering for diffuse and reflective materials.

The rendering equation has been approximated to Light Emitted = sum(coefficient * incidentLight * brdf)

Diffuse Material: For diffuse lighting, random sampling of the ray is done uniformly from the hemisphere containing the normal. For perfectly diffuse materials (Lambertian), the BRDF is uniform, i.e the light is scattered in all directions equally. This is done by sampling uniformly along a hemisphere oriented with the normal at a point. The probability is constant for every sample and hence, we can ignore the division.

Reflecting Material: For reflecting material, the rays concentrated around the actual reflected ray are sampled. A roughness parameter can be changed to give the material a more matte look.

3.2 Texturing

If the texture of the material is specified, then the color is taken from the texture. For some objects, point to texture coordinate is defined. For the sphere it is given by $u = atan2(d_y, d_x), v = asin(d_z)$, where d is the vector from Point to Center of the sphere.

Algorithm 3 Path Tracing Main Loop

```
for each pixel (i,j) do
    Vec3 C = 0
    for (k=0; k < samplesPerPixel; k++) do
        Create random ray in pixel:
            Choose random point on lens P_lens
            Choose random point on image plane P_image
            D = normalize(P_image − P_lens)
            Ray ray = Ray(P_lens, D)
        castRay(ray, isect)
        if the ray hits something then
            C += radiance(ray, isect, 0)
        else
            C += backgroundColor(D)
        end if
    end for
    image(i,j) = C / samplesPerPixel
end for
```

Fig. 1. Path tracing Algorithm

4 RESULTS

4.1 Path tracing with primitives

The primitives implemented are spheres, triangles, and cylinders. Mesh uses triangles.

Out first attempt:

Uniform diffuse lighting.

4.2 Textures

We added support for textures for spheres.

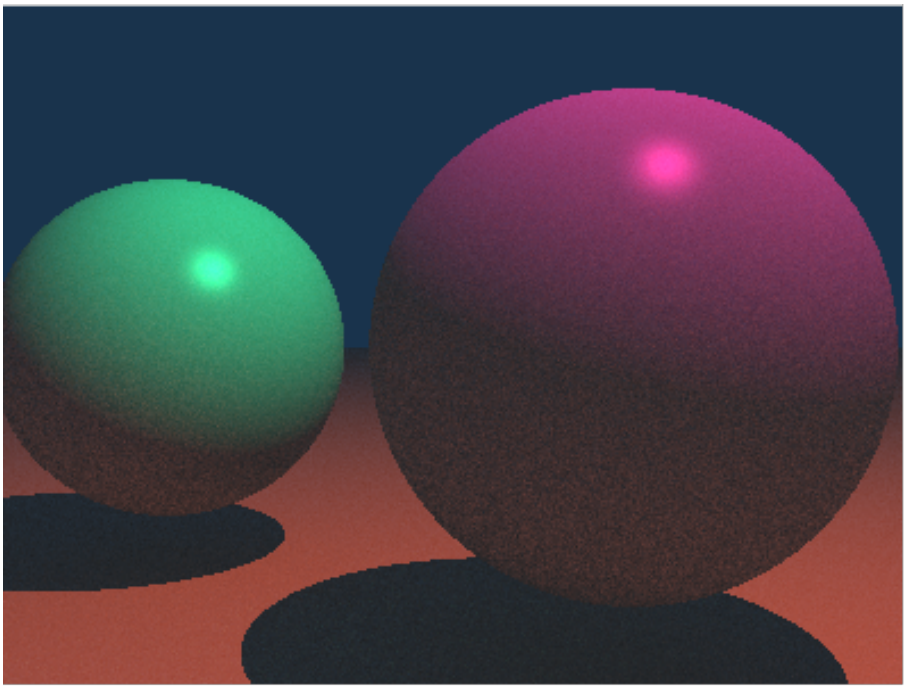


Fig. 2. Path tracing with direct and indirect diffuse lighting - Light bleed can be observed on the spheres. Direct lighting calculated with ray tracing

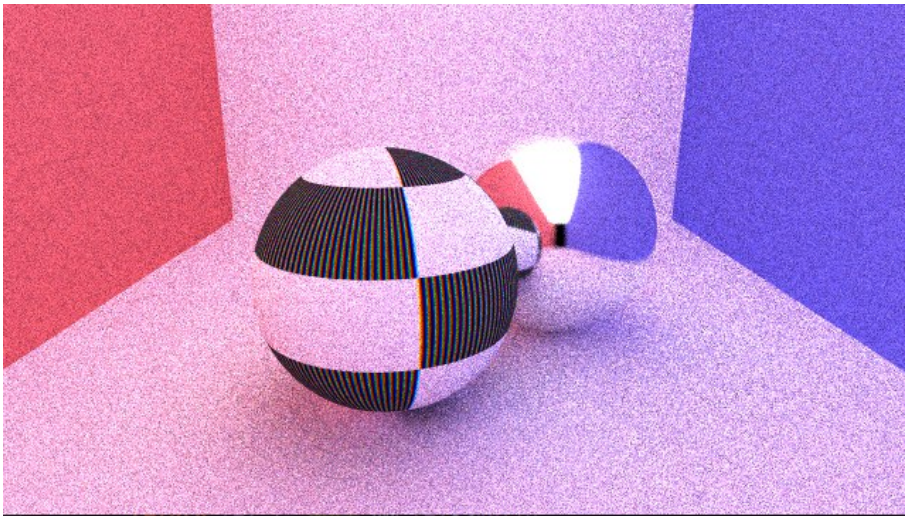


Fig. 4. Texture

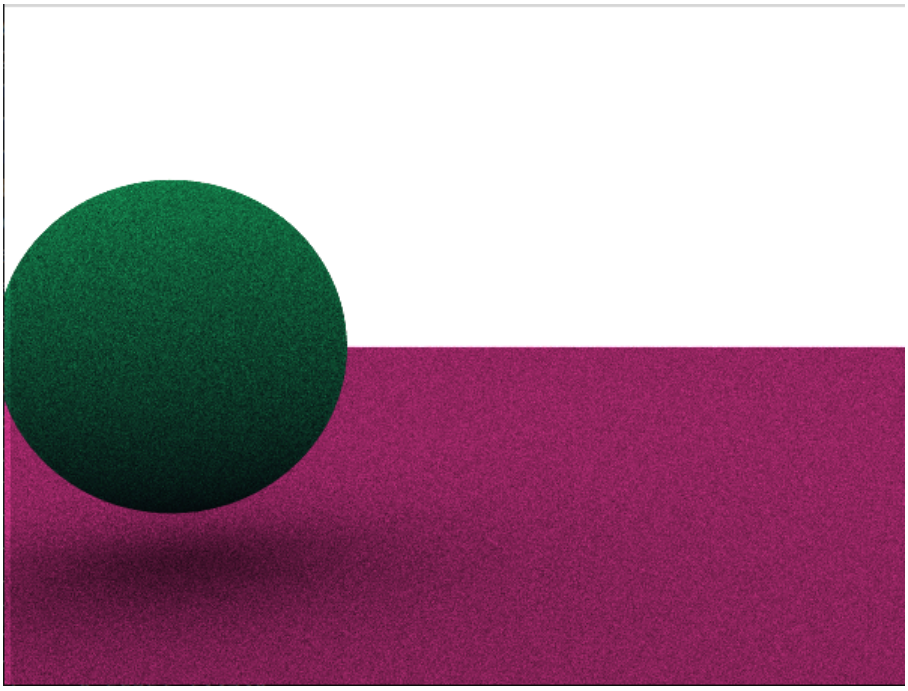


Fig. 3. Path tracing with Lambertian Surface

4.3 Meshes

Importing scenes from triangular meshes. The following is a teapot import from a .obj file. Each triangle of the teapot has been fashioned as a Triangle primitive. In the intersect function of the teapot, the ray checks all the triangles of the teapot for intersection and returns true if any one is true. It can be optimized in the future.

5 DISCUSSION

In the following pictures we can see the difference that sampling rate or the samples per pixel makes on the image quality. The sampling rate in Fig 5 is 1 SPP and in Fig 6 is 8 SPP. The SPP makes a huge difference to the quality of the image up to a certain extent (for the images we rendered, it was around 100SPP after which the improvement was negligible).

Teapot and other complex triangular meshes take a lot of time to render even with 1SPP because of the large number of triangles in the figure. Each ray checks if it has intersected any of the triangles for each bounce. There are 10 such bounces.

At present area light sources are being used. Point light sources or dimly lit scenes prove a huge problem for out path tracer. This is because the scattering is random, nearly all rays miss the light sources. This can be rectified with light tracing (which is a forward pass method) or with more sophisticated forms of path tracing like Metropolis Light Transport or Bidirectional Path Tracing.

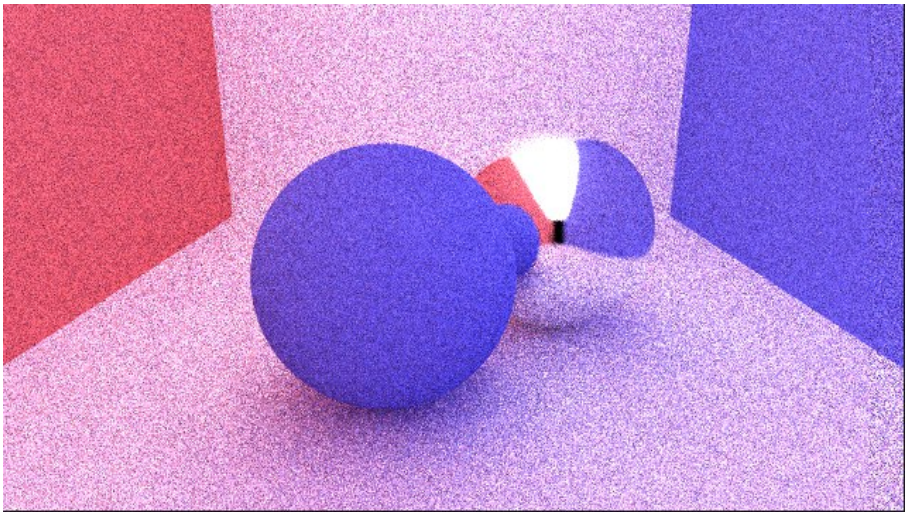


Fig. 5. Reflecting material and Diffuse material

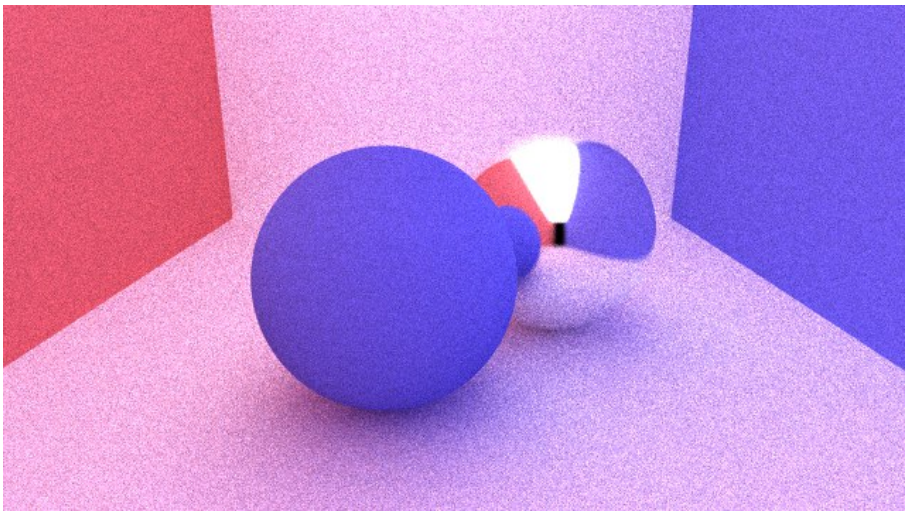


Fig. 6. More samples per pixel

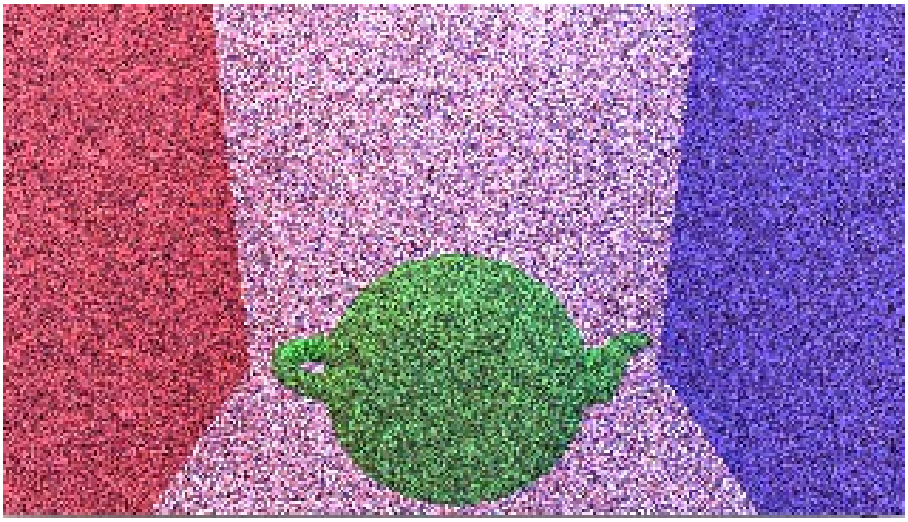


Fig. 7. Rendering a mesh

6 FUTURE WORK

A major factor with the path tracer is the time is takes to render the scene. This is partly due to the reason that the all rays check with each and every object if it intersects it. This becomes especially hard for meshes. Future work would include optimizing this by implementing kd trees to reduce the space where the rays check for intersection. After optimizing, we could try and implement caustics.

Our path tracer can also be made more parallel. Although using the GPU is a good option, this would require rewriting the recursive ray tracer to an iterative one. A lot of optimizations can be made to make the path tracer faster.

7 ACKNOWLEDGEMENTS

Used Assignment4 code provided by the TA as the basis for the path tracer. The object files for triangle mesh objects with its vertices and faces was taken from http://people.sc.fsu.edu/~jburkardt/data/obj/

REFERENCES

[1] https://www.wikiwand.com/en/Rendering_equation
[2] http://www.dusterwald.com/2016/07/path-tracing-vs-ray-tracing/
[3] https://www.wikiwand.com/en/Path_tracing
[4] https://www.scratchapixel.com/lessons/3d-basic-rendering/global-illumination-path-tracing
[5] http://www.dusterwald.com/2016/07/path-tracing-vs-ray-tracing/
[6] https://blog.demofox.org/2016/09/21/path-tracing-getting-started-with-diffuse-and-emissive/
[7] http://www.dusterwald.com/2016/07/path-tracing-vs-ray-tracing/