Global Illumination using Photon Mapping

Pedro Tanaka (cs184-bd), Mengyi Wo (cs184-fs)

I. Introduction:

Ray tracing trace rays from the viewer and compute the local shading of the intersection points on the objects by using techniques such as the Phong shading model. Because shadow rays are casted to decide whether the color of a certain pixel is computed, areas that are in shadow will appear as black if using ray tracing. Thus, caustics and indirect lighting cannot be calculated only using ray tracing. Photon mapping is a technique to achieve global illumination by emitting photons from the light sources into the environment. When the photon intersects a surface, Russian roulette is used to determine whether the photon is reflected, transmitted or absorbed. Photon mapping is a two-pass algorithm. In the first pass, we construct the photon maps by tracing the photons into the environment and storing them in photon maps using a K-d tree structure for faster photon access. A global photon map is constructed for indirect lighting and a caustic photon map with high density is constructed for caustics. In the second pass, we compute the color of a pixel in four parts. We use regular ray tracing to compute the direct illumination and specular reflection of intersection point. Global photon map is used to estimate indirect lighting. Caustic photon map is used to render caustics.

II. Implementation:

We started from the ray tracer we implemented for assignment 3. We extended the program to support OBJ files so we can have more complicated models. Support for MTL files is also added so the OBJ files can have material definitions.

We further improved the ray tracer to support refraction. Upon intersecting with a transparent surface, the new direction of the ray is calculated using Snell's law based on ratio of the two materials' refractive indices. If the calculation results in total internal reflection, we ignore the ray and move on to another ray.

Some other refinements to the ray tracer include super-sampling and BSP-tree using Axis-aligned bounding boxes. When compute the color of a pixel, we shoot several rays from the same pixel instead of one and average the colors returned to get better image quality. When the OBJ files are parsed and primitives like sphere are added to the scene, a BSP-tree is constructed, which will give us better speed in ray-object intersection.

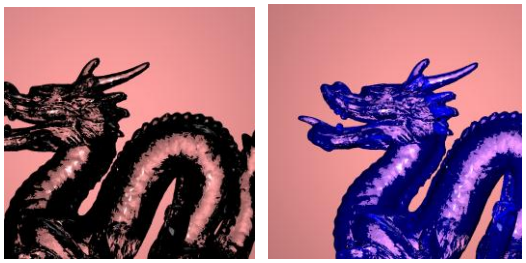
We then adapted Henrik Wann Jenson's implementation of photon map to our program. We wrote a function to trace photon and a Russian roulette function to determine the result of a photon hitting a surface. If specularly reflected, the photon is traced again in the reflected direction. If diffusely reflected, a new direction will be randomly chosen in the hemisphere. If transmitted, refraction direction is calculated similar to that of ray tracing. If the photon is not reflected or transmitted, it is stored in the photon maps. Because we are calculating direct illumination using Phong shading model, we set the Russian roulette such that a photon hitting a diffuse surface for the first time will always be reflected. For the caustic photon

map, we emit photons towards the caustic photon for better estimate of caustics.

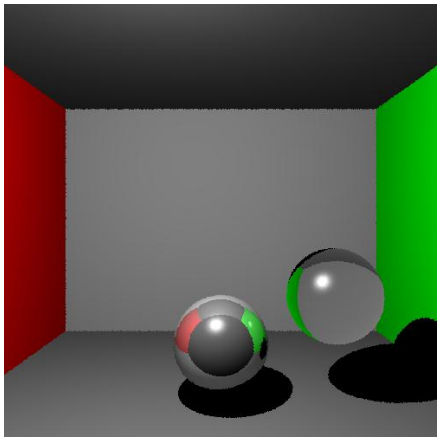
For the rendering, we used Phong shading model to calculate the direct illumination and specular reflection. Then we do a final gather to calculate the indirect lighting. We cast several random rays from the intersection into the environment and estimate the radiance by gathering photons at the new intersection points. This radiance multiplied by diffuse reflectance will be added to the final picture as indirect illumination. For caustics, we directly gather photons from the caustic photon map and add this contribution to the final picture.

To make our program faster, we also tried to parallelize the computation using OpenMP.

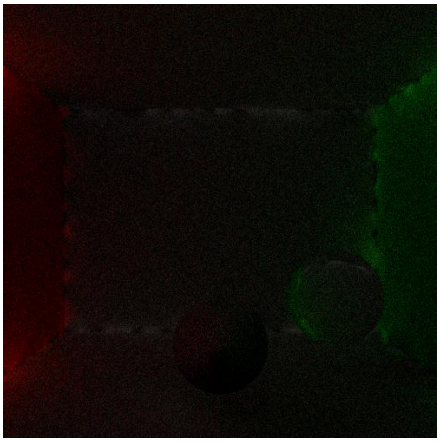
III. Results



Refraction (clear glass and colored glass)

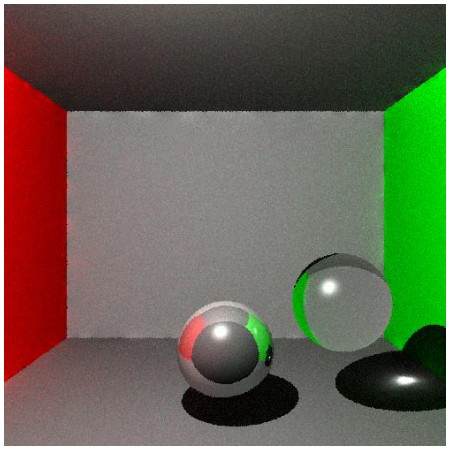


A cornell box using only ray tracing



Indirect illumination using global photon map

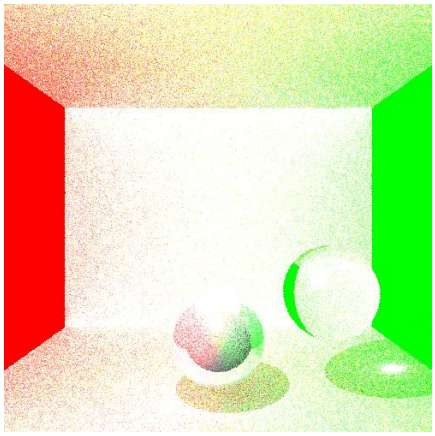
Caustics using caustic photon map



The final picture

For this set of pictures, we emitted 50000 photons into the scene. When 500000 photons are used, we

will have a very bright picture with lots of color bleeding as in the picture below.



IV.Discussion

By using photon map, we were able to produce caustics effect and indirect diffuse lighting. The shadows under the spheres now have some diffuse color instead of entirely black, although still not very natural looking and correct. We also get some color bleeding on the gray walls.

The challenges of this project include: 1.Trace photons and store them correctly and preserve the light energy. 2.Correctly estimate the radiance using the photon maps. Our earlier implementation resulted in very little and nearly none color bleeding effect on the wall, and very little color in the shadow because we were not tracing and storing the photons correctly so the light energy fell off very quickly.

We also ran into problems such as self-shadowing and truncation of floating numbers to integers. Those careless mistakes took us a long time to debug and taught us a lesson to write code more carefully.

Further improvement for this project include using projection maps to effectively emitting photons towards caustic objects when constructing the caustic photon map. In order to ensure we have a high density caustic photon map, we are emitting photons to all directions and only counting the photons that are hitting caustic surfaces. This technique is clearly inefficient and could be improved. In addition, we could run this program on a more powerful multi-core processor desktop to generate better images by using more rays in final gather. Our laptops limited us to use only few rays in the final gathering.

V.Reference

Realistic Image Synthesis Using Photon Mapping, *Henrik Wann Jensen*