Final Project Report
Srinath Ravichandran & Yuan Tian

Introduction:

For the final project, we've implemented a variety of complementary features to our ray tracing application framework. The details of the implemented features are as follows.

1.Soft Shadows

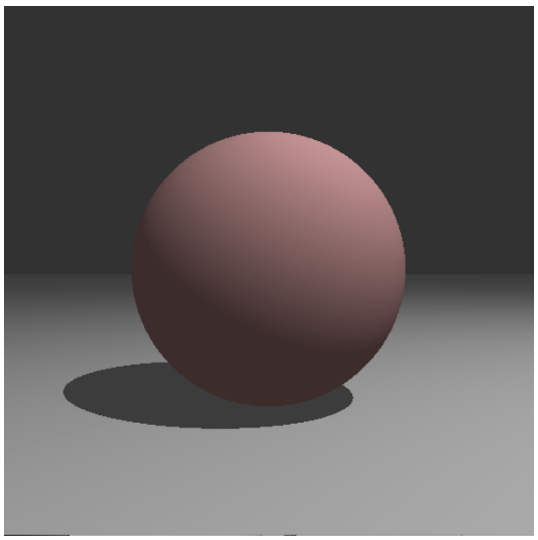

Figure 2a: Hard shadow due to a point light

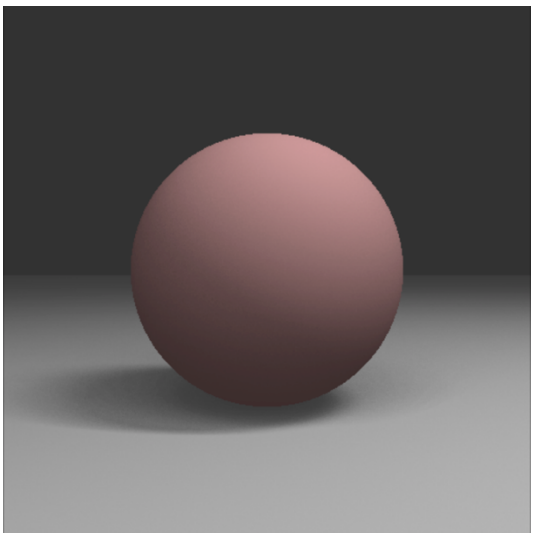

Figure 1b: Soft shadows (500 samples)

We've implemented soft shadows in the framework and the figure above shows the difference between employing hard shadows and soft shadows. Soft shadows are present only in the presence of area lights. We implement soft shadows by randomly selecting points on the area light for a given shade point. The lighting equations are evaluated along with shadow ray evaluation. Since shade points can see a fraction of the random points in the light, they can be in the umbra (no points visible), penumbra (partial points visible) and fully lit (all points visible) regions.

2.Texture Tiling

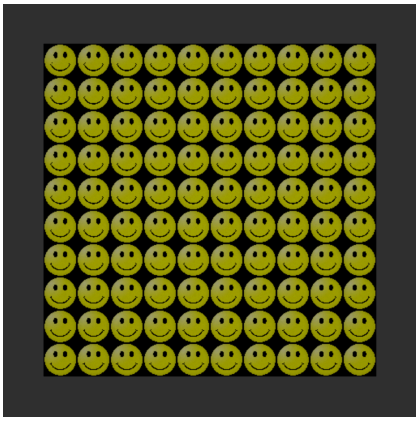

Figure 2a: 10 x 10 tiling

Figure 3: 4.3 x 4.3 tiling

Figure 4: 5.7 x 5.7 tiling

We've implemented texture tiling that allows mesh vertices to have arbitrary texture coordinates other than the range of [0,1]. This feature is implemented by using the mod function to wrap around the texture coordinates for the final texture coordinate computation in the shade function.

3.Projective Texture Mapping

We have implemented projective texture mapping of spheres with spherical, cylindrical and cubical mapping. The uv coordinates are computed based on the functions for computing the respective projective functions. The final uv coordinates are then tiled to have seamless textures. They can be clamped also.

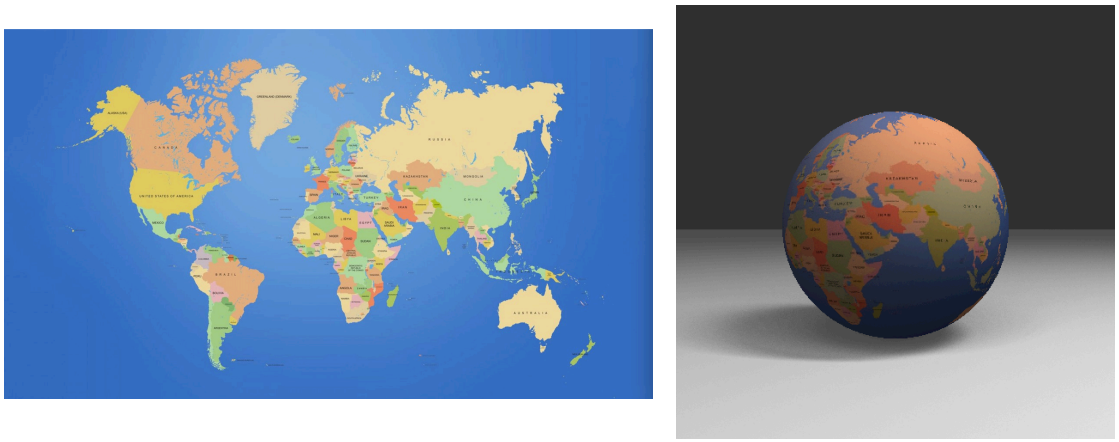

Figure 5: (Left) World map texture. (Right) Texture applied to the a sphere.

4. Mesh Intersection

We've implemented mesh intersection that allows the application employ meshes that basically contain a list of vertices and triangle indices. We've also implemented a ray-triangle intersection routine to enable the application to support triangle primitives. For each mesh, the naïve linear intersection code iterates through the entire triangle list and computes the nearest intersection of any intersection depending upon the intersection routine. The naïve method can be very slow for large meshes and an acceleration structure like BVH can be used to substantially reduce the intersection times.

5. Environment Illumination

We've implemented environment illumination that allows us to light scenes using an illumination map rather than having to employ point lights or area lights. For each shade point in the scene, depending upon the material properties, a light evaluation ray is computed which is then used to compute the spherical coordinates of the ray direction which are then converted to uv coordinates. These uv coordinates are used to fetch lighting value from the environment map. Using environment maps, we can simulate lighting of a virtual scene from the original environment the map was captured. A large number of samples might be required to compute the lighting without any noise since rays can be distributed in random directions in case of diffuse surfaces.

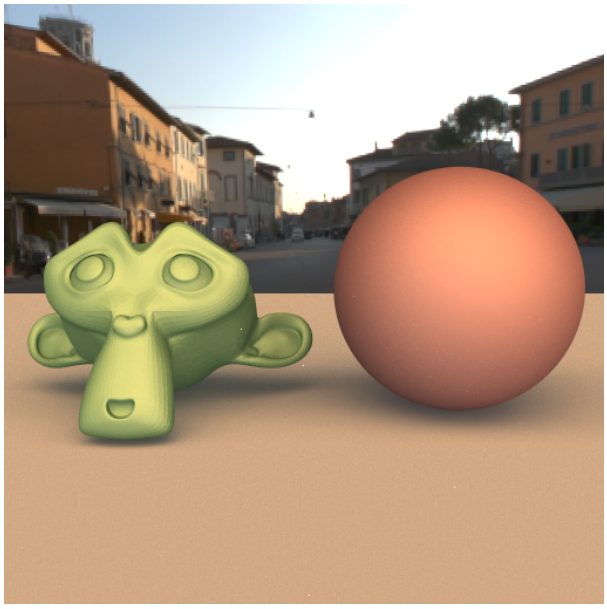

Figure 4: Environment illumination

6. Indirect illumination

We've implemented indirect illumination to show global illumination effects within the scene. Direct lighting just captures one bounce of light, which will make the scenes less realistic. Capturing multiple bounces of light gives the scene a very realistic look. Global illumination effects such as color bleeding are visible only when multiple bounces of light are considered. In order to implement indirect illumination, in addition to computing the direct lighting at each bounce, we spawn subsequence rays based on material parameters at the current shade point. The reflected rays are then used to compute the incoming bounce light by recursively evaluating the ray tracing function. A maximum bounce limit is set so as to stop recursion once the fixed depth is reached. The factors during each bounce are accumulated together to compute the final color for each pixel.

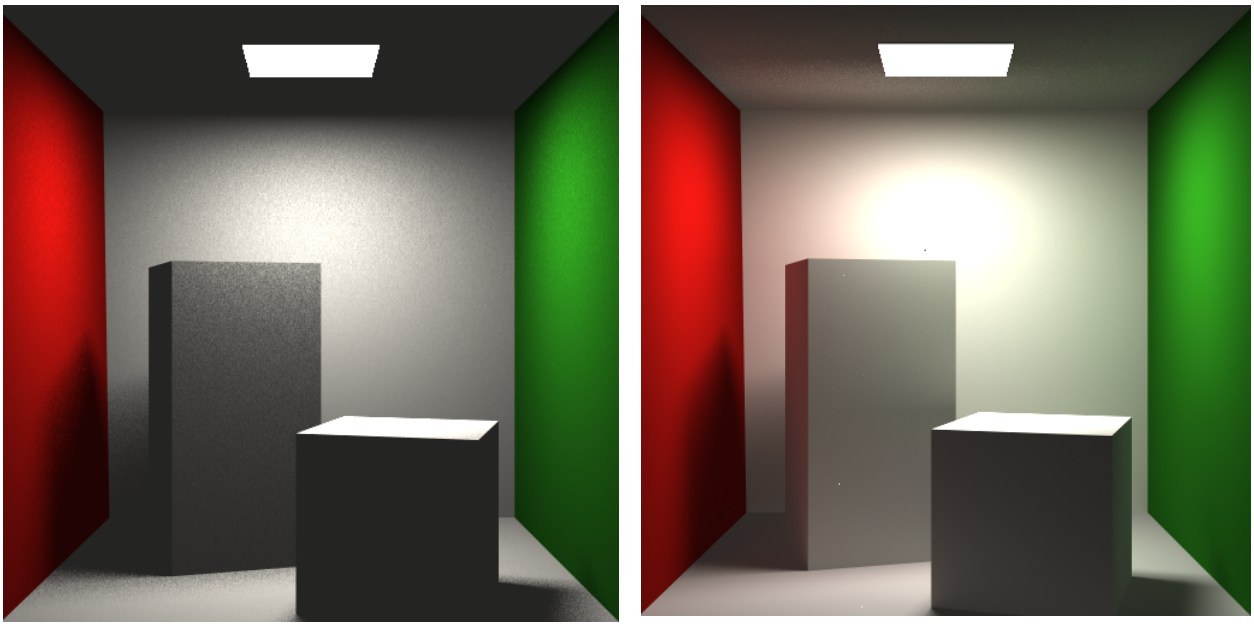

Figure 5: Left Image - Direct Illumination only. Right Image shows global illumination effects to multiple bounces (2)