Deferred Rendered Radiosity from First Person Perspective

This article present a new global illumination algorithm design for real-time. It handles diffuse lighting. The related research that has most similarity with this work is a paper named *Splatting Indirect Illumination*. If the hardware conditions are optimal the whole process could be implemented on the GPU with just a few draw calls.

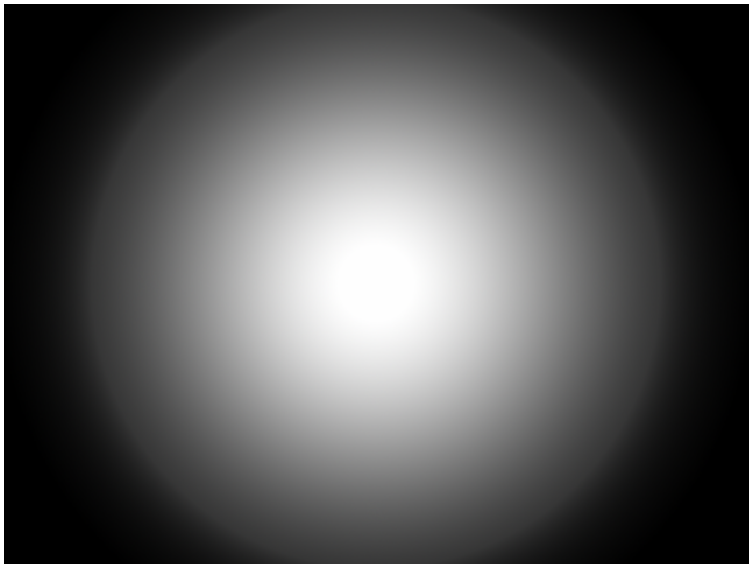
The heart of the algorithm is to treat the indirect light as many individual direct *reflection lights*. These reflection lights are constructed from the underlying geometry. Surfaces with similar properties build up a reflection light. By setting up rules for how to constitute a light the visual result could be controlled. In a way this process could be viewed as simplified radiosity. The algorithm was aimed for a *Laparoscopic Simulator* where the light source is mounted to the camera. Other applications where the algorithm could prove useful are first person shooters where the avatar wears a headlamp. From a first person perspective the light source and camera "sees" the same things and this brought certain optimization properties. By projecting all necessary data to the screen and perform the operations in screen space the problem is transformed from 3D to 2D, which generally has much less complexity. In fact this makes the algorithm nearly independent of geometric complexity. Since the aim for the algorithm is real-time applications I have restricted the light to perform 1-bounce. In most cases the 1-bounce simplification will give a decent interpretation of indirect lighting. There are examples of animated movies rendered with radiosity which only allowed between 1 and 2-bounces that has received *Oscar Academy Awards*[1]. If desirable the algorithm has built-in properties to generate more than one bounce. The light can also be detached from camera with minor modifications and performance penalties.

To more easily understand the concept, consider the monitor screen as a measuring devise for a second, where every pixel is a sample spot that tells how much light energy that passing through. If an object occupies some amount of pixels on the screen that quantity is proportional to the amount of light energy the object receives and further reflects to its surroundings. Small object at right angles could receive as much energy as large tilted objects. The fact that irradiance is proportional to the inverse quadratic distance from the light source is implied because of the perspective projection. The nature of the light caster also has importance for how much energy a surface receives. To practically portrait this I incorporate what I call *intensity distribution texture* which is just a plain texture that depicts the amount and direction of the light sources radiation in screen space, see the picture below.

With a huge number of lights a fast way to shade is needed; therefore I incorporated a deferred renderer into the system. In short, this technique postpones traditionally 3D world operations to 2D image space, with high optimization as result. Here it's used in a way where it performs *phong* lighting with perfect depth complexity and effective clipping. The deferred rendering approach also fits well with the assumptions above.

---

[1] A film named *Bunny* won *Best Animated Short Film* 1999



Intensity Distribution Texture

The algorithm has several global variables that control the performance and appearance. One of the design strategies has been to hold the number of variables low; too many gives a cumbersome situation. Since the settings are adjustable the same implementation could fit many targets. By tweaking the setting right the visual result of this algorithm could be very similar to does of offline global illuminators, with only diffuse lighting and a single bounce there is. The main difference between the approaches is that my algorithm lump areas into unities where the others use many point samples. Many offline illuminators can handle area light; this can only be imitated with the intensity distribution texture to some extent.

Walkthrough

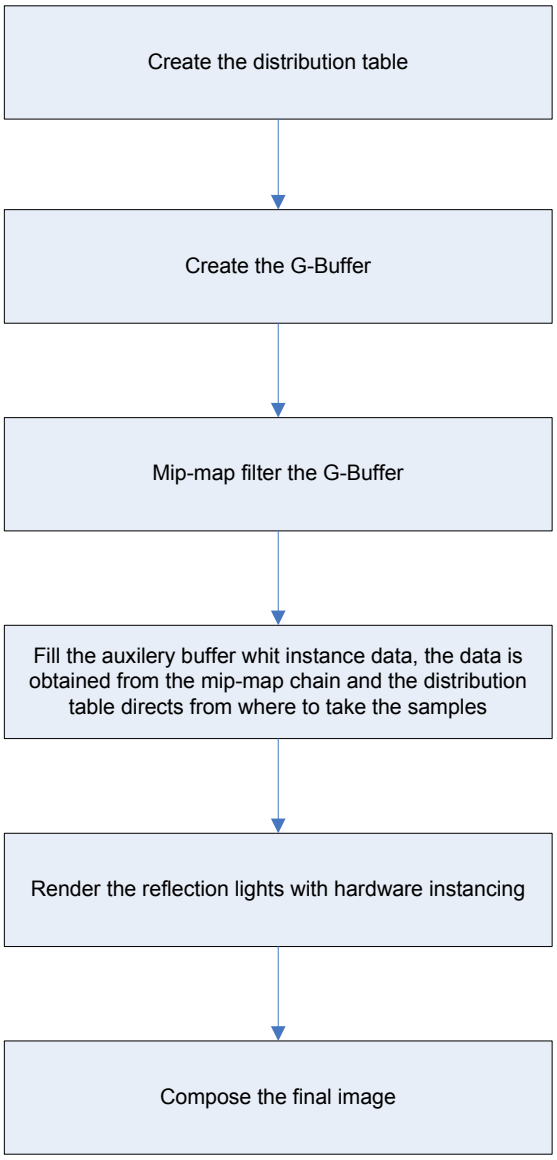
I will here give a short survey of how the algorithm works. First all data needed to perform the light operations is projected to textures, aka *G-Buffer*. The G-Buffer has four components:

- Surface normal in camera space
- Position in camera space
- Irradiant flux
- Diffuse color

The flux is obtained from material properties blended with the intensity distribution texture. Attenuation is accounted naturally with perspective projection. All data needed for rendering the scene is present in the G-Buffer. From its contents a distribution of reflection lights are created, these lights are scattered everywhere direct light hit a surface. Therefore the distribution could be viewed as a coarse generalization of the G-Buffer.

The G-Buffer is minimized several times and stored in a mip-map chain. A custom filter is used during minimization. The filter is design to merge texels that has similar properties according to some thresholds. If texels are discontinuous the most significant contributors is kept and the others discard. Samples are collected from the mip-map chain. From every sample a reflection light is rendered. In a preprocess a *distribution table* is created. This table is used as a guide when collect the samples. The higher level a sample is taken from the more texels it represents.

The shading process is carried out by a deferred renderer. The reflection lights are handled one at the time; there data and the contents of the G-Buffer produce the shaded result. The contribution from every light is blended into the frame buffer. To increase speed a mesh that depicts the lights active volume masks out an area in screen space where the shading is performed. There is no need to render outside this area since it couldn't have any effect, just waste instructions. The reflection lights are rendered with hardware instancing. If *render to vertex buffer* capability is available the auxiliary stream that feds the instancing could be created on the GPU. The direct light contribution and other effect is computed and blended in as layers. The final result is presented to the screen. See the chart below.


Create the distribution table
Create the G-Buffer
Mip-map filter the G-Buffer
Fill the auxilery buffer whit instance data, the data is obtained from the mip-map chain and the distribution table directs from where to take the samples
Render the reflection lights with hardware instancing
Compose the final image

The Process