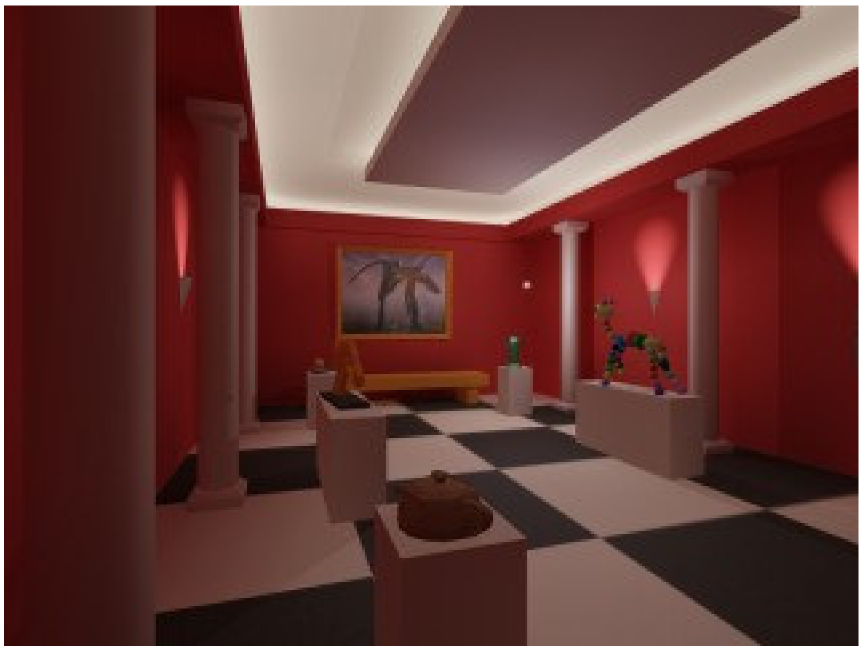
Global Illumination

This sample implements a method for rendering global illumination in real-time.



Overview

To achieve realistic lighting of a scene we have to take more than just the direct lighting into account. Global illumination where the light doesn't come just from the light source directly but also indirectly by bouncing off other surfaces in the scene can really make rendering look a lot more lifelike. Traditionally indirect lighting has often been approximated with ambient lighting. While this simple approximation generally improves appearance compared to having shadowed parts of the scene go totally black it still leaves a lot to be desired. This sample uses no ambient lighting but you can still see details in the shadowed parts of the scene. The position of the light as well as the surroundings clearly affects the light in shadowed parts of the scene.

Controls

Arrows/WASD	Move around
Mouse	Look around
Space	Toggle framerate display
F1	Toggle help
F9	Capture screenshot
-, +	Cycle wall color
1, 2	Control ceiling light 1 intensity
3, 4	Control ceiling light 2 intensity
5, 6	Control wall lights intensity
7, 8	Control dynamic light intensity
9	Toggle dynamic light animation
PageUp / PageDn	Increate exposure
P	Visualize probes

Details

The lighting in the scene consists of two components, direct lighting and indirect lighting. The direct lighting is regular Phong per-pixel lighting with cube shadow mapping for the dynamic light and a plain lightmap for the ceiling and wall lights. Indirect lighting is computed by sampling the direct lighting in the scene into cubemaps from various probe location spread out across the scene. This means that it doesn't matter how the direct lighting is rendered, it will automatically contribute properly to the indirect lighting. Just by rendering emissive materials in the scene you get proper contribution to the indirect lighting.

The probes are placed on a uniform grid across the scene. There are 308 probe locations (11x4x7) but in order to improve performance "empty" nodes (those that don't contribute to the lighting of any part of the scene) are skipped over. Actual number of active probes is 295. Each frame we update a number of probes. Given that the probe updates is the by far most expensive part of the rendering and the scene is updated iteratively anyway it doesn't make sense to try to update all probes before rendering to the framebuffer. Instead we try to maintain 60fps and update as many probes we can fit within that time slice. For the ATI Radeon™ HD 2900XT this amounts to about 40 probes.

The probes are conceptually represented by cubemaps, but instead of using actual cubemaps we use slices in a 3D texture. This way we can update more than one probe per pass, which is not possible with cubemaps since D3D10 does not support cubemap arrays. We don't need cubemap sampling anyway so using a 3D texture is not a problem. 40 probes means there are 240 slices. Rendering this in D3D9 style would require rendering the scene 240 times, which could easily turn into tens of thousands of draw calls even for relatively simple scenes. In order to minimize the number of passes required the geometry shader is used to output all geometry to multiple slices per pass. The geometry shader has an output limitation of maximum 1024 scalars. In order to maximize number of probes per pass we can update we minimized the data needed to be passed to the pixel shader. At 10 scalars we could fit 5 probes per pass (5 x 6 x 3 x 10 = 900 scalars). This means we can update all 40 probes in 8 passes, which is a bit more manageable than the 240 passes D3D9 style rendering would require.

Once the scene has been rendered into the cubemaps for all probes the probes are converted to Spherical Harmonics coefficients by integrating the light function stored in the cubemap over the basis functions. This is done by rendering single points into a volume texture where each point corresponds to a probe. The sample uses four SH bands, and thus we need 16 coefficients, so multiple render targets are used to store the result into 4 volume render targets.

Finally the scene is rendered to the framebuffer. For the indirect lighting a long dot-product is computed between the SH coefficients from the previous pass and the SH coefficients for the transfer function to evaluate the integral over the light environment. A simple diffuse light transfer function is used. The only input is the normal so the SH coefficients for the transfer function are precomputed on load time and stored in four cubemaps for fast access during rendering. The indirect lighting is also computed in the same way when rendering into the probes so that multiple light bounces can be achieved across a number of frames.

Check ati.amd.com/developer for the latest material.

Painting courtesy of San Base
Copyright © 2006 San Base
http://www.sanbase.com