Real-time Dynamic HDR Based Lighting in a Static Environment

Marcus Hennix  Daniel Johansson  Gunnar Johansson  Martin Wassborn
{marhe745, danjo008, gunjo829, marwa338}@student.liu.se
Linköping University, Campus Norrköping 2004

Abstract

A problem when using standard HDR light maps in a synthetic scene is that the objects have to stay stationary in order for the lighting to be correct. Our idea is that this can be solved using standard image morphing techniques to solve for occlusion in the scene. Using the morphed or interpolated HDR map, the synthetic object can be lit at different locations.

1. Related work

Other than the exampled mentioned in the Image Based Rendering course [1,2], our initial inspiration sprung from a real time HDR lighting application done by a Japanese named Masaki Kawase [3]. His demo displayed very good looking lighting and lens effects using HDR images in real-time. We wanted to combine these visual effects with morphing techniques to get a very good looking real-time application which made it possible to move the lit objects in space.

Several papers have been published covering the area of view morphing and view interpolation [5,6,7,8], so we felt that the morphing part of our project would become manageable.

Eventually, we found a paper [4] describing our problem of navigating in static environments that constituted the technical foundation on which our application was built. Our idea was to expand the way of looking at the navigation to contain the lighting information in the HDR maps as well.

2. Our approach

Unlike Kawase’s approach we wanted move in the static environment. In order to do so, we needed some kind of geometric information about the scene. We initially set out to capture real world geometry (depth maps) but this proved very time consuming and tedious. Instead we used depth maps generated by 3d studio max which were lit by HDR probes, captured at various locations.

From the generated images we produced geometry by triangulation which was rendered to an off-screen cube map. Using the cube map we lit a synthetic object.

To make the scene more vivid we added lens effects through Cg shaders and multiple render passes.

3. Creating the environment

To start with, we modelled a scene in 3d Studio Max to generate the images and their corresponding depth maps for use in our pipeline.

3-1. Lighting Acquisition

Real light probe images were taken at several locations across Norrköping and then edited and finished in Adobe Photoshop and Paul Debevec’s HDRShop application. These were then imported into 3dsmax and used as lighting in the scene.
3-2. Modelling
A simple scene of a room was modelled. We tried to emphasize changes in the lighting intensity across the modelled scene, so that we would have both very bright and very dark areas. The models were created with the standard tools in 3dsmax.

3-3. Generating images
To be able to use high dynamic range images as both input textures and output images from 3dsmax we needed to use a different renderer than the standard scanline renderer. We chose Brazil, a very good renderer that can use HDR images and calculate global illumination among many other things.

To reconstruct the scene in our real-time application we needed depth information from each image. After some experimenting we found that the best option to do this was to render a separate pass with no lighting, using white distance fog as depth information. A script was written in maxscript to render a cube map with depth information.

4. Application pipeline
4-1 Environment reconstruction

Loading HDR images
Our entire application pipeline is depicted in Appendix A.

In order to load HDR images, we had to write an HDR loader which handled RLE-encoded HDRI images [9] (which are what HDR-shop exports). When loading HDR images, Gamma correction is applied to give more life to the picture. This is to be replaced with proper camera curve functions if time permits.

Triangulating
From the information supplied in the depth images, it is possible to reproject the points from screen coordinates to view coordinates. To do this, you need to guess what parameters where used when rendering the image, for example field of view and near/far clipping planes. Making these guesses proved to be very hard since we didn’t have any in depth technical information about our renderer and thus couldn’t use the rendered depth buffer directly. Instead we used a rendered image with distance fog to give the depth to use when reprojecting our points.

4-2. Off-screen buffers
Using the OpenGL WGL_PBUFFER_ARB extension we set up off-screen pixel buffers with appropriate properties. Rendering to a pixel buffer is just like rendering to the frame buffer. The only difference is that you switch rendering targets.

4-3. Shaders
We used a variety of shaders to achieve the visual effects seen in the screenshots. The shaders are implemented in hardware through Nvidia’s Cg library. A list of relevant effects is listed below.

Bloom
This shader simulates the blurring effect that can occur on areas on an image with high intensity. This effect is implemented by first blurring the whole image, then blending this image over an identical but unblurred image using a smooth curve.

Exposure
The human eye lens can contract in order to compensate for different lighting conditions. In order to simulate this phenomenon we measure light in the scene by reading evenly sampled points and their light intensity. By changing exposure according to this mean, we can simulate the effect.

Reflect/Refract
These effects occur on very shiny objects and/or transparent objects. We calculate pixel intensities by using the eye position and the surface normal. In the case of the refract shader we also need the index of refraction of the transparent material.

5. Results
We are very satisfied with our results given the limited timeframe during which we completed the project. We achieved the desired visual look in real time, something that we first believed to be unlikely. While the geometry obviously isn’t correct, it is sufficient to solve for occlusion and supply us with a convincing overall impression.
6. Future work

The geometry of the scene could be further improved by using images rendered at different locations to compensate for missing information. We also want to improve our triangulation in order to decrease the polygon count in the geometry. This could be done by a smarter triangulation to start with, or by a decimation algorithm.

It would be really interesting to experiment with some real life photography and supplied depth information to see how well our method adapts to real environments.

References


Appendix A

3D STUDIO MAX

HDR PROBE PHOTOGRAPH

DEPTH MAPS

TRIANGULATION

TEXTURED GEOMETRY

VIEWPORT TRANSFORM

SYNTHETIC OBJECTS

DYNAMIC CUBEMAP

ENVIRONMENT SHADER

BUFFER 1

BLUR SHADER

BUFFER 2

COMPOSITING SHADER

BUFFER 3

FRAME BUFFER

LOOP n TIMES

RENDERED HDR IMAGES

DEPT骶 MAPS

HDR LOADER

COMPOSITING SHADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER

HDR LOADER