Computer Graphics: Realtime Shadow Mapping with GLSL

Brady Ledger
Brady.Ledger.6525@student.uu.se
Uppsala Universitet

Figure 1: *Shadow mapping application, depth buffer image texture, shadow mapping problems.*

Abstract

Shadow mapping is a rendering process whereby shadows are added to 3D objects. Its use is widespread within prerendered scenes and realtime scenes in applications (i.e. video games).

Shadows are generated by checking whether a pixel is visible from the source of light. This is performed by comparison to the *z-buffer* image of the view from the light source, stored in texture format.

A common issue of realtime shadow mapping is aliasing or shadow continuity glitches. A simple way to overcome this limitation is to increase the shadow map size, but due to memory, computational or hardware constraints, it is not always possible. [Isidoro 2006]

1 Implementation

The shadow mapping algorithm uses two passes. Initially, the scene is rendered from the light's point of view offscreen via a *Framebuffer Object* (FBO). Only the depth values of each fragment are saved during rendition; no color texture is bound to the FBO and color writes have been disabled.

The scene is then rendered from the center of projection, but with an additional test to see if the current fragment is in shadow: if the current sample is further from the light than the depth buffer image at the same point, this indicates that the current fragment is in shadow – as it must be behind an object. [Wik]

The `shadow` variable within the *fragment shader* holds the shadowed test result. The method compares the z value of the vertex rendered from the light's point of view, with the z values rendered to the shadowmap.

2 Tweaks

The application presented allows for several interactions. A number of tweaks that affect the quality of finish for the shadow mapping can be accessed via interaction with the provided AntTweakBar.

The camera can be moved around the origin using `WASD` keys, and can zoom in with the mouse scroll wheel (or `[` and `]` keys).

2.1 Self-shadowing

Self-shadowing, or z-fighting, happens due to limited precision in the depth buffer. This affects fragments facing the light, as these are the values written to the shadow map. Removing front faces when rendering the shadow map, then switching to the back on scene rendering only applies shadowing to fragments not in the light.

`glCullFace` switching is enabled by default. To show self-shadowing, uncheck the *'refine: cull faces'* option.

2.2 Moiré aliasing

Moiré aliasing is an effect brought about by limited depth map precision and difference in sampling rates between the shadow map and the scene, causing inexact depth values.

Bias is introduced to reduce the inclusiveness of the shadow test, removing the Moiré surface acne. This can be adjusted via the *'refine: moire bias'* tweak. Too little bias causes Moiré surface acne to appear, too much leads to incorrect shadows (peter-panning). [Isidoro 2006]

2.3 Other tweaks

light rotation: radius Change the radius of the circling light. Causes shadow map resolution issues at large values.

refine: sampling tolerance Turn on sampling of depth values behind the light's view frustrum, by reducing the limit from `0` to `-40`. Causes artifacts.

display: depth buffer Show the depth buffer image.

display: zoom Move centre of projection with the camera. Shadows mapped incorrectly upon close proximity to objects.

display: background Change the color of the background.

References

ISIDORO, J. R. 2006. Shadow mapping: Gpu-based tips and techniques. In *Proceedings of Game Developers Conference 2006*, ATI 3D Applications Research Group.

Shadow mapping. `http://en.wikipedia.org/wiki/Shadow_mapping`. Accessed: 2014-06-04.