Thygrrr's Shadow Caster

OpenGL 1.2 and SGIX_shadow extension based Shadow simulator

(c) Moritz Voss, 2k+2

This program module serves the purpose of adding a fine structure or grain to an image, and then casting rays of light over its surface, thus creating a depth-of-field effect, which primarily manifests itself in various forms of shadows and specular reflections.

System Requirements

- **OpenGL 1.2 Platform**
- **SGIX_shadow** extensions family (e.g. Oxygen GVX od nVidia GeForce3 chipsets)
- Windows32, Macintosh, or Linux/Unix Operating Systems
- Two-button-mouse with mouse wheel.

Preamble

This is a small, quite unfinished piece of code published under the GPL. It was supposed to be an university semester work, but it hardly qualifies as such – but heck, who knows, I might as well submit it and hope I didn't waste a semester. I personally find the program might be worth quite a bit to newcomers who don't understand all the *nVidia* demos, as I had to work through tons of their source before finding a paw full of well-commented, useful lines.

Getting started

After loading the module, you will be prompted with the usual ‘Load Textures’ dialog. You can choose any number of textures – granted you have enough graphics memory – which will be treated either as a single texture or as an animation, which will cycle infinitely if you push the Play button. Textures should meet the usual OpenGL requirements – width and height must be powers of two.

Once you have loaded your textures (this may take a while), you’re ready to go! What you should see is the first image of those that you have chosen, mapped onto an appropriate polygon mesh, which, initially, is spread out in the XY plane. Somewhere on the screen, you will find a little white ball with a thin white line sticking out of it. This is the Light Source Visual Aid, and the Light Destination Vector Visual Aid. These help you imagine where your light is actually pointing while you play with the program.

By clicking your left mouse button, you place the light source somewhere else on the screen, and by clicking your right mouse button, you define the end point of the direction vector, i.e. the point the light is shining at. You should be able to see how the light already illuminates you image a bit. Since computer displays are two-dimensional on the one hand, the nature of OpenGL is 3D on the other, the mouse wheel will have to serve as the main control mechanism to influence the ‘in-screen depth’ of your mouse wheels. Scrolling upwards (i.e. away from you with most mice), you push the last moved object (light source or light destination) away from you, into the depths of virtual 3D. By turning the mouse wheel in the other direction, you pull the light components towards you. A small display of changing, specifically scaled (so they are easy to read) coordinates will give you a rough idea which way you are moving, and how far you will have to go on to reach the location you intended to occupy.

Moving the end-point of the light will result in it shining onto the surface at different angles, while moving the light source itself will bring it closer to the surface, changing its reflection’s focus, or move it away from the image, giving the light more room to spread out.

The Control Panel

To the right, you see a control panel, framed by the framework software’s control buttons (play, load, etc.), and the memory counter & virtual trackball. It’s split into two control pages, and one pseudo-apologetic ‘about’ dialog, which is more of a monologue, coming to think of it. The first page deals with general control of the lighting models, and of the visual aids, while the second solely serves the purpose of configuring the fractal terrain synthesizer.

Illumination Model

This ComboBox at the top of the panel lets you choose between “Normal Lighting” and “Normal Lighting + Shadow Map”, the two lighting models supported by this program. Normal Lighting is the default, while the computationally intense, but fancier Shadow Mapping may be activated at your discretion. This will not work if you don’t have the necessary platform support!

Light Source & Gloss Effects

Below the Illumination Model ComboBox, you find another pair, letting you choose between spotlight and directional light sources, as well as allowing you some freedom of choice what your image material should behave like. If you choose to use a directional light source, please note that the light direction vector no longer points to the exact ‘destination’ of the light, but is instead the direction of the parallel light source itself. The glossiness modes are no gloss, image gloss (bright areas in your image are reflective, dark ones aren’t), and metal foil (your entire image appears to be printed on some kind of metallic foil). The latter uses OpenGL’s **GL_SEPARATE_SPECULAR_COLOR** colour control mode.

Light Colour

The four sliders below simply regulate what colour the incoming light has. The alpha value primarily has only an effect on the visual aids, but may affect specularity in later versions of this software.

Visual Aids

- **Light Src**: Shows the light source as a gluSphere(...) in the light colour.
- **Light Dst**: Attaches the destination vector to the Sphere. Also works standalone.
- **Show normals**: This displays the surface normals of the mesh holding your image
- **Light/Depth View**: Displays a small window that shows you what the actual shadow map looks like. Only available when shadow mapping is active.
- **Clip Aids**: Perform GL_DEPTH_TEST on the aids, hiding them if they are logically behind the image surface.

Terrain Synthesis

The ‘Surface’ panel merely consists of a few sliders, and a button that allows you to re-seed the random number generator (otherwise, all changes you do will be based on the same random seed, thus letting you permutate your terrain non-destructively. You do this by selecting one of the three terrain types from the drop-down box on the page, and simply fiddling with the controls till you have a structure that suits your desires. You may influence the amplitude (maximal height of the ‘mountains’) as well as the coarseness (percentage of small/pointy hills), and the falloff, which essentially results in how finely split up your hills will be. As a post-synthesis step, you may apply a variable strength finite impulse response filter, letting you smooth out the terrain to any degree – even till it’s completely flat again. The two algorithms I have implemented are **Fault Formation**, which simulates tectonic faults in terrain, and **Midpoint Displacement**, also known as Diamond Step or plain Fractal Terrain.

About Dialog

This little text explains why this program is so sucky :)

Frequently Asked Questions:

Q: In the two Glossy Material modes, I get odd, dark, and strongly aliased patches in some of the valleys. Is that a bug?

A: Not really, it appears to be a quirk of OpenGL. It’s the last area that is illumined by the spot light on its outer fringes, but the way you are looking at it, the specular component gets multiplied onto it a hundredfold too strong. The best way to cope with this is to switch glossiness off, or move the light source to a steeper angle against the image surface.

Q: Whoa. I am getting really wacky artefacts in Shadow Mapping mode!

A: That is quite inevitable, as well. Shadow Mapping works through a means of projective texturing, and the projection is limited to a maximum angle of <180° . The program uses 175, just for your information. Because there is no real/rational 360° fish-eye-matrix, 180° view ports is the best you can achieve, and that takes a heavy toll on depth buffer precision, then. As for the shadow map, being a projective texture, it cannot ‘back project’, that means areas behind it twist into rather wacky forms of infinity (looks rather cool, doesn’t it?).

Q: I get small bright spots in my shadowed areas!

A: Okay. This one, for once, really is my fault. I can’t seem to properly set up depth-biasing for my current viewport configuration in the program. Sorry.

Q: Why is the program in such an immature state?

A: One word – procrastination.