CMSC 23700 **Introduction to Computer Graphics** **Project 2(b)**
Fall 2004 **November 22**

Terrain rendering (Part b)

Due: Thursday, December 8

1 Overview

The second part of Project-2 is to add various visual effects to your terrain. Everyone should implement texture mapping and at least two other effects. Additional effects are worth extra credit.

2 Texture mapping

To make the surface of the terrain look more realistic, your implementation should blend in a *detail texture*. Each map has a `detail.ppm` file that contains the texture; you can use the function

```
RGB_t *LoadTexture (Map_t *map, const char *name, int *wid, int *ht);
```

to load this texture.

You use the detail texture, which is essentially a noise texture, to modulate the surface color close to the camera (say out to 80 to 100 meters). The texture is allocated as an RGB image, you may want to copy it to an RGBA representation with a 40-50% alpha channel, which will mute its effect somewhat. You can also use alpha blending to get a smooth transition from textured polygons to untextured ones. You should not app

3 Other extensions

In addition to texture mapping, you should implement one or more of the following features. Note that will most of these features, you will want to use techniques such as frustum culling to avoid unnecessary computation and rendering.

3.1 Trees

A barren landscape in uninteresting, so to make it more attractive we can add some vegetation. A map can contain a `tree` file, which is a text file containing the locations of trees in the terrain. Use the function

```
Tree_t *LoadTrees (Map_t *map);
```

to load the array of tree information.

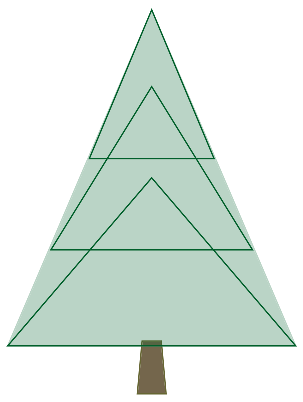

Figure 1: Tree constructed from tapered cylinders

For purposes of this project, we use a simple representation of trees based on four concentric tapered cylinders (see Figure 1). We can use `gluCylinder` to render each of the tree components; to keep the polygon count down, you should use multiple levels of detail in your rendering. You may also want to consider various culling techniques.

3.2 Geysers

Driving around a static wasteland is boring, so we populate the terrain with geysers. Information about the position and size of geyser's is stored in the `geyser` file. Use the function

```
Geyser_t *LoadGeysers (Map_t *map);
```

to load the array of geyser information.

The behavior of a geyser is controlled by three parameters: the frequency f, the duration d, and the average height h. A geyser erupts on average once every $\frac{1}{f}$ seconds with an average duration of d seconds. The average height of an eruption is s.

We model geysers using particle systems. The initial velocity vector of the particles should have a cone-shaped distribution. You will want to set the average velocity to get the particles to the average height. You should also taper-off the system (both in particle velocity and number of particles) to get a gradual ending of the eruption. For example, let d be the duration of the geyser's eruption, then we can define the "size" of the geyser as a function of time t (assuming $t = 0$ is the start of the eruption):

$$s(t) = \begin{cases} he^{-(\frac{2t}{d})^2} & 0 \leq t \leq d \\ 0 & t > d \end{cases}$$

For example, $s(d) \approx 0.02h$. You can use this size function to influence the initial particle velocity, number of particles, and particle lifetimes. Feel free to play with your parameters and other decay functions.

3.3 Water animation

The surface of the lakes is flat and static. We can make it more interesting by adding waves to the water surface. The `water.pgm` file contains a depth map that gives the water depth for each grid location (a value of zero means dry land). Use the function

```
   Elev_t *LoadWaterMap (Map_t *map);
```

to load the water map. This map can be used to construct meshes that represent the lake surfaces, which can then be animated.

3.4 Shadows

The map format has been extended to include the direction of the sun (specifically, the vector for a directional light). In this extension, you will precompute a shadowmap from the heightfield that can be blended with the surface texture to result in a shadowed landscape. You can determine if a vertex is in shadow by casting a ray from the vertex back towards the direction of the light (see Figure 2). Using this information, you precompute the lighting information and luminance map for the terrain.

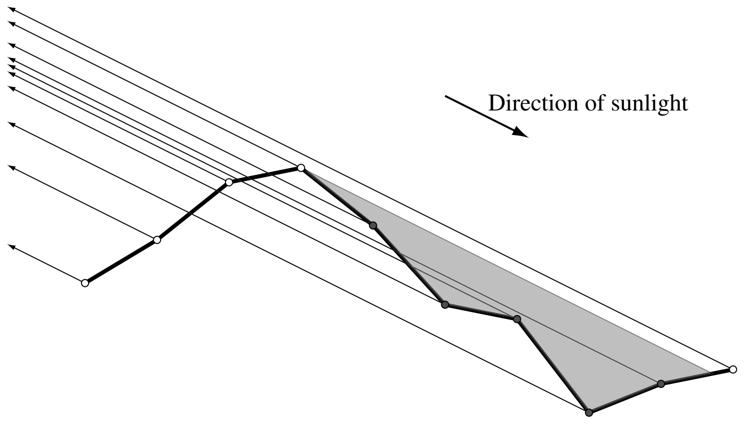

Figure 2: Computing shadows by ray casting

At rendering time, you turn off lighting and instead blend in the luminance texture.

4 Requirements

Part-b of the project is due at 12pm on Thursday, December 8. Your final version must be checked into your gforge repository at that time. Your project should include a README file that describes which extensions you implemented and gives a brief outline of the techniques that you used.

5 Document history

Nov. 22 Original version.