CMSC 23700 — Introduction to Computer Graphics — Project 2(a)
Fall 2005 — November 15

Terrain rendering

Due: Wednesday, November 30

1 The problem

For this project your task is to implement a simulator for the next generation of off-road vehicles: the *sports utility hovercraft* (SUH). This program will read in terrain height and texture information and render the view from a simulated vehicle. The terrain height data is given as a square array of 16-bit height samples, which define a grid (or heightfield). This project is divided into two stages. In the first, you will implement a terrain renderer and vehicle simulator. The terrain renderer uses a level-of-detail optimization for the heightfield based on the ROAM algorithm. In the second stage, you will add texture mapping and two or more special effects.

In this document, we give an overview of the problem and describe the first part of the assignment.

2 Heightfields

Heightfields are a special case of mesh data structure, in which only one number, the height, is stored per vertex. The other two coordinates are implicit in the grid position. If s_h is the horizontal scale, s_v the vertical scale, and $\mathbf{H}$ a height field, then the 3D coordinate of the vertex in row i and column j is $\langle s_h j, s_v \mathbf{H}_{i,j}, s_h i \rangle$ (assuming that the upper-left corner of the heightfield has X and Y coordinates of 0). By convention, the top of the heightfield is north; thus, assuming a right-handed coordinate system, the positive X-axis points east, the positive Y axis points up, and the positive Z-axis points south. The heightfield is typically represented as a linear array of height samples, with the i, j element at index $iw + j$, where w is the width of the heightfield. Because of their regular structure, heightfields are trivial to triangulate; for example, Figure 1 gives two possible triangulations of a 5×5 grid. For this project, we will use the ROAM algorithm, which uses the triangulation shown on the left of Figure 1.

3 The vehicle

The user interface provides simple controls to navigate in the simulated SUH. This section describes the controls and the simulated physics of the vehicle.

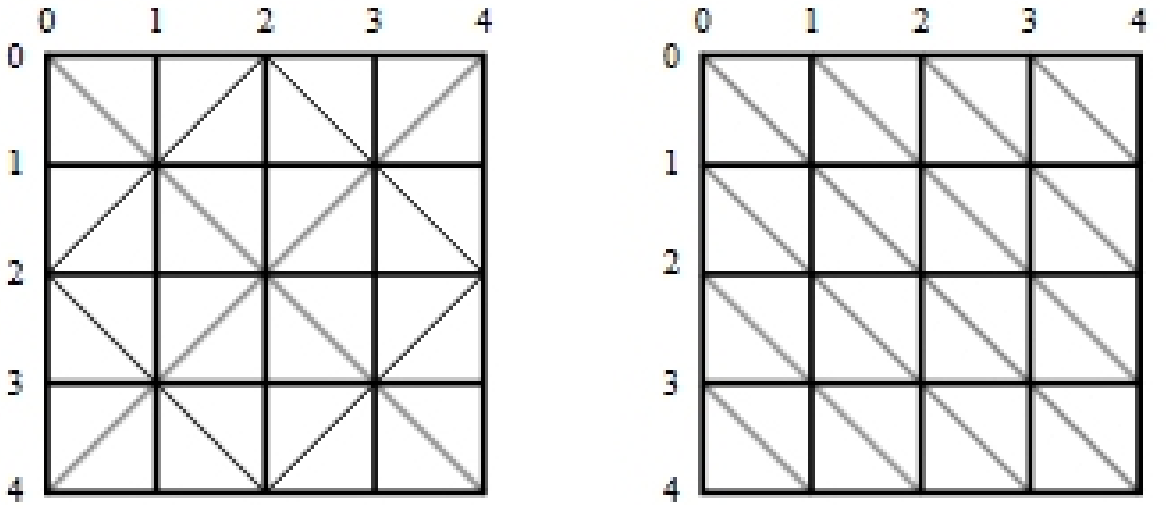

Figure 1: Heightfield triangulations

3.1 The controls

Navigation is controlled using the arrow keys. In addition, your implementation should support controls for the view as described in the following table:

UP ARROW	accelerate
DOWN ARROW	brake
LEFT ARROW	turn left
RIGHT ARROW	turn right
`f`	toggle fog
`l`	toggle lighting
`w`	toggle wireframe mode
`+`	increase level of detail (by $\sqrt{2}$)
`-`	decrease level of detail (by $\sqrt{2}$)
`q`	quit the viewer

For the navigation controls, you will need to sample the state of the arrow keys (instead of just reacting to keyboard events). GLUT provides a function for registering a callback that gets called when a special key is released:

```
void glutSpecialUpFunc (void (*func)(unsigned int key, int x, int y));
```

Thus you will need two callbacks to keep track of the state of the arrow keys.[1] Since holding down a key generated a repeated sequence of key events, which take time to service, you can disable key repeats using the following GLUT call:

```
glutIgnoreKeyRepeat (1);
```

3.2 The physics model

We simulate the SUH with a very simple physics model based on discreet sampling. The state of the vehicle at a step i is given as a triple $(\mathbf{p}_i, v_i, h_i)$, where $\mathbf{p}_i$ is the vehicle's position in the $X-Z$ plane,[2] v_i is its velocity in $\frac{m}{s}$, and h_i is its heading in degrees (with north being 180 and south being

[1] Note that you should guarantee that any transient keystoke gets sampled at least once in the physics model.

[2] Note that the position $\mathbf{p}$ of the vehicle is given in X-Z coordinates; the altitude of the vehicle (the Y coordinate) will always be 2 meters above the terrain at the vehicle's position.

0). We recompute the state of the vehicle one hundred times per second (*i.e.*, every 10 milliseconds). Given the vehicle's state at step i, we can compute its new velocity at at step $i+1$ as follows:

$$
\begin{aligned}
a &= \max\left(0.0, 8.0 - \frac{v_i^2}{16.0}\right) \\
f &= f_0 + f_1 v_i + f_2(v_i^2) \\
g &= \mathbf{g} \cdot \mathbf{s}(\mathbf{p}_i, h_i) \\
v &= v_i - 0.01 \times (f + g) \\
v_{i+1} &= \begin{cases} \max(0, v + 0.01 \times a) & \text{if accelerating} \\ \max(0, v - 0.01 \times b) & \text{if breaking} \\ \max(0, v) & \text{otherwise} \end{cases}
\end{aligned}
$$

In these equations, a is the acceleration, which is speed limited, f is the force of friction, and g is the force of gravity. The new heading of the vehicle is determined by the following equation:

$$
h_{i+1} = \begin{cases} h_i + \frac{t}{(v_{i+1}^2)+l} & \text{if turning left} \\ h_i - \frac{t}{(v_{i+1}^2)+l} & \text{if turning right} \\ h_i & \text{otherwise} \end{cases}
$$

and the new position is given by

$$
p_{i+1} = p_i + 0.01 \times v_{i+1} \langle \sin(h_{i+1}), \cos(h_{i+1}) \rangle
$$

This computation depends on a number of factors, which are defined as follows:

b	$=$	0.6	braking factor
f_0	$=$	6×10^{-4}	friction coefficient
f_1	$=$	2×10^{-4}	friction coefficient
f_2	$=$	4×10^{-4}	friction coefficient
$\mathbf{g}$	$=$	$\langle 0, 9.8, 0 \rangle$	gravity
l	$=$	32	turn limit
t	$=$	24	turning factor
		$\mathbf{s}(\mathbf{p}, h)$	unit slope vector with direction h at position $\mathbf{p}$

If the vehicle is traveling at velocity v_i, we first compute a new velocity v that represents the effects of friction and gravity. We then apply acceleration and/or breaking to compute v_{i+1} (note that we do not let the velocity fall below zero). We use the new velocity in computing the new heading. Lastly, we compute the new position.

Computing the slope function $\mathbf{s}(\mathbf{p}, h)$ can be done in one of a couple ways.

- Project a 2D unit vector $\mathbf{d}$ in the direction h; *i.e.*, $\mathbf{d} = \langle \sin(h), \cos(h) \rangle$ and let $\mathbf{p}' = \mathbf{p} + \mathbf{d}$. Then let $H(\mathbf{p})$ be the height at position $\mathbf{p}$. The unnormalized slope vector is $\langle \mathbf{d}_x, H(\mathbf{p}') - H(\mathbf{p}), \mathbf{d}_z \rangle$. Divide this vector by its length to get $\mathbf{s}(\mathbf{p}, h)$.
- The other approach is to let $\mathbf{n}$ be the normal vector of the triangle containing $\mathbf{p}$ and let $\mathbf{d} = \langle \sin(h), y, \cos(h) \rangle$, for some unknown y. Then solve $\mathbf{n} \cdot \mathbf{d} = 0$ for y and set $\mathbf{s}(\mathbf{p}, h) = \frac{\mathbf{d}}{\|\mathbf{d}\|}$.