CMSC 23700 **Introduction to Computer Graphics** **Project 2(b)**
Fall 2004 **November 16**

Terrain rendering (Part b)

Due: Monday, November 29

1 Overview

The goal of a terrain rendering algorithm is to render a triangle mesh that models a given heightfield. The naive approach of part-a is to draw the complete triangle mesh, which requires rendering very large numbers of triangles (*e.g.*, a 1024×1024 heightfield has 2 million triangles). Even with view-frustum culling, the number of triangles is substantial. Since rendering such large numbers of triangles in real-time is impractical, we will use a *continuous level-of-detail* (CLOD) scheme in Part-b to reduce the number of triangles rendered per frame. Many techniques have been developed for managing level-of-detail when rendering terrain. In the second stage of the project, you will replace the simple mesh renderer of part-a with the *split-only* variant of ROAM algorithm. You will also add lighting and fog to the renderer.

1.1 Lighting

For this part, we will add a single directional light (the sun) to the scene, which is specified in the map file. Adding lighting means that you will need to specify normals for your triangles as you render them.

1.2 Fog

Fog adds realism to outdoor scenes. It can also provide a way to reduce the amount of rendering by allowing the far plane to be set closer to the view. The map file format has been extended to include a specification of the fog density and color. We will use the `GL_EXP` fog mode for this project, but feel free to experiment with other settings

1.3 The controls

The controls are enhanced to support fog and dynamically changing the level of detail.

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

1.4 The input format

For this part, the input format has been changed to accommodate lighting and fog parameters. The following fields have been added to the `Map_t` structure to hold these values:

```
typedef struct {
  ...
    Vec3_t      sunDir;     /* direction of sun light */
    RGB_t       sunColor;   /* color of sun */
    float       fogDensity; /* fog density parameter */
    RGB_t       fogColor;   /* fog color */
  ...
} Map_t;
```

2 ROAM

The ROAM algorithm is organized around a dynamic representation of triangle meshes called *triangle binary trees* [DWS$^+$97]. Figure 1 gives an example of a tree and Figure 2 shows the corresponding levels of triangulation. In the split-only version of this algorithm, we compute a new tesselation

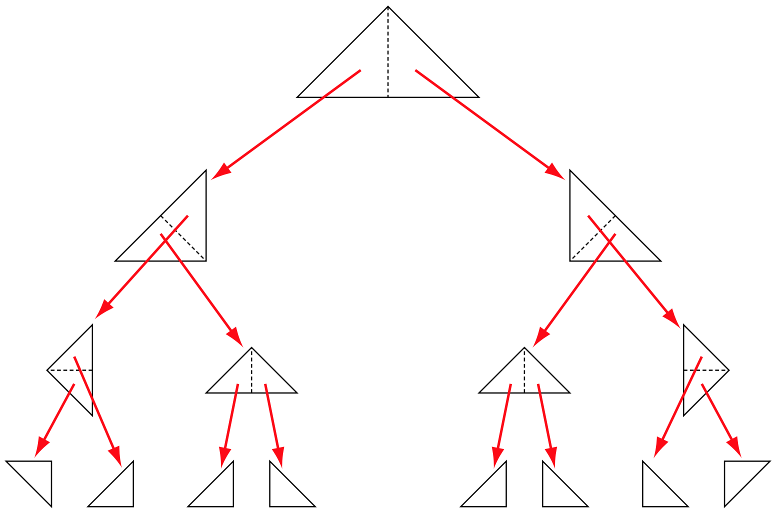

Figure 1: Binary triangle tree

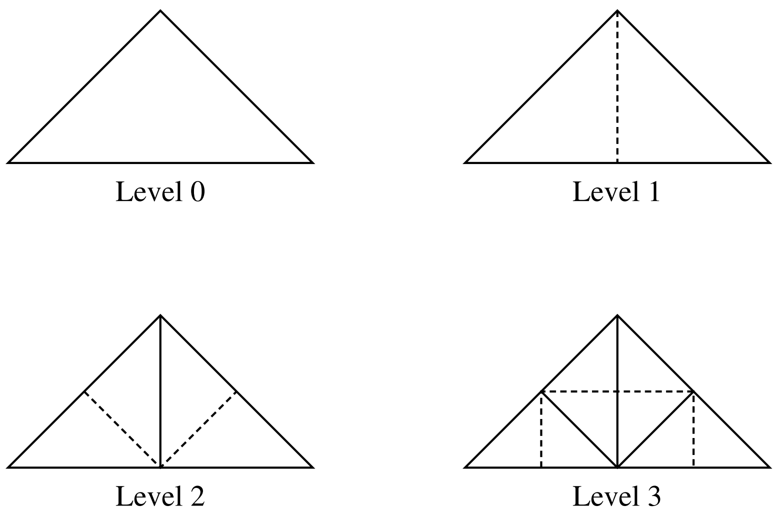

Figure 2: Binary triangle tree levels

of the heightfield each frame by starting with the two triangles that cover the whole heightfield and then refining the mesh. We assign triangles a priority based on the benefit of refining them (*e.g.*, error metrics). Each triangle in the mesh has three neighbors (except for those triangles on the border) as is show in Figure 3.

As can be seen from these figures, constructing a binary triangle tree can be done as a recursive splitting procedure. The trick is that we only want to split a triangle if the resulting mesh provides a visibly more accurate approximation of the height field. Thus, we modify the recursive splitting procedure to split the triangle with the highest priority, where priorities are a measure of the visual effect of not splitting. We use a limit of the number of triangles in the mesh to control the amount of rendering work we do. Thus, the psuedocode for the tesselation phase is

```
initialize the mesh to top two triangles
while (size of mesh < limit) {
  split highest priority triangle
}
```

Splitting a triangle requires splitting the triangle's base neighbor (otherwise a T-junction results), but it may also presplitting the neighbor, when it is at a higher-level in the binary triangle tree. Figure 4 shows this situation.

2.1 Hints

You can adjust the priority of triangles to eliminate detail where it is not needed and to enhance detail where it is needed. For example, triangles that lie wholly outside the view frustum should have minimum priority, while the triangle containing the vehicle should have maximum priority.

Your program will need several distinct, but related data structures. You start with the heightfield that is the input data. You will need to compute a *variance tree* that contains the world-space variance information, a representation of the triangle mesh, and a priority queue for ordering splits. A strict priority queue is both not necessary and not efficient enough. Instead, use some number of priority buckets (think radix sort) to get constant-time insertions and deletions. It is also useful to

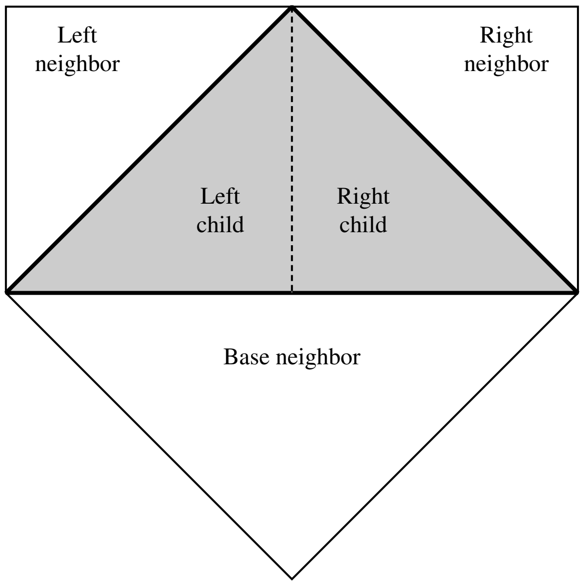

Figure 3: Triangle neighbors

have the bintree triangles down to the mesh level.

3 Requirements

Part-b of the project is due at 9pm on Monday, November 29. Your final version must be checked into your CVS repository at that time. You should set the vehicle's initial position as before and set the initial triangle budget to 10,000.

4 Document history

Nov. 16 Original version.

References

[DWS+97] Mark Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein. ROAMing terrain: Real-time optimally adapting meshes. In *IEEE Visualization*, pages 81–88, 1997.

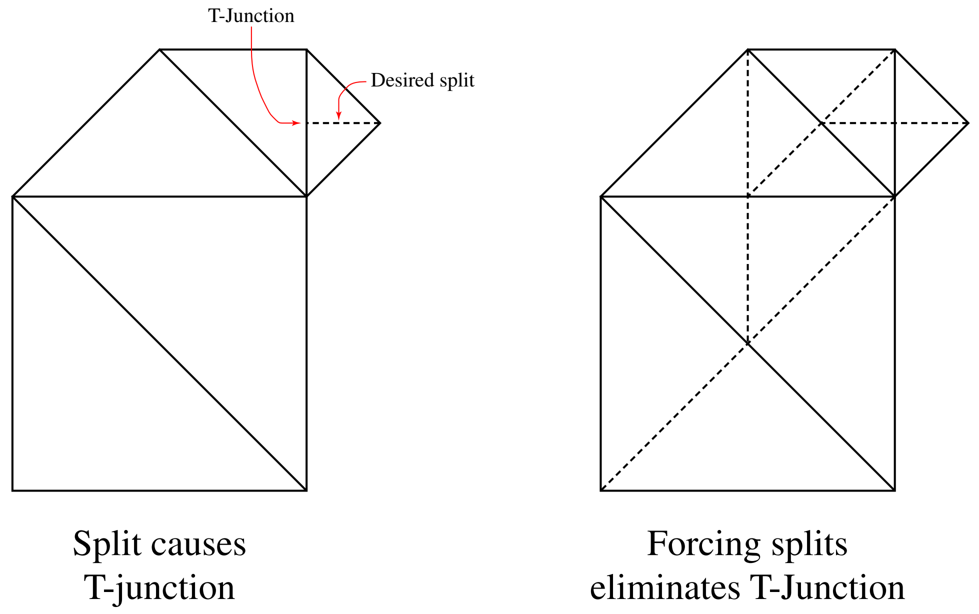

Figure 4: Forcing splits