CSCI E-235 Assignment 1
Static and Dynamic Terrain Simplification

Due Date: Wednesday, March 13, 2002, noon
Note the new deadline!

1. Introduction

In this assignment you will implement static and dynamic terrain simplification algorithms. The terrain is specified using a simple grayscale image map, where pixel values correspond to height. Your program reads grayscale PPM, RAW or “whatever format you like” images and internally builds an appropriate mesh data structure. You will implement static (also called discrete) simplification using edge merges and QEM, and dynamic (also called continuous) simplification using triangle bintrees and a view-dependent error metric. Your program displays either the statically simplified terrain with a user specified number of triangles, or displays the terrain using ROAM with dynamic simplification.

The goal of the homework is to compare the approximations made by static simplification and QEM, with the view-dependent simplification made by ROAM. A secondary goal is to compare the relative speed of these algorithms, both for pre-processing and during runtime. However, we do not grade your homework based on absolute speed or efficiency. It is much more important to write clean, easy to understand code than to worry about performance.

2. Requirements

Your final program needs to have the following features:
- It reads in a terrain height field as grayscale image. The program searches for all files with appropriate extension in the current directory and displays the first filename at the bottom of the display window. To change files, the user presses the space bar. The text display in the render window shows the next filename. The file is read in when the user presses the enter key. (Note: Alternatively, you can use GLUI or the Microsoft Win32/MFC classes to implement a simple file selection user interface.)
- The terrain is displayed as a triangle mesh in the display window. The user should be able to rotate and zoom the terrain with mouse control as follows:
  - Press Left MB, move up/down: rotate around x (horizontal) axis
  - Press Left MB, move right/left: rotate around y (vertical) axis
  - Press Right MB, move up/down: zoom in/out
  You may use GLUI to implement this interaction.
- Pressing the ‘a’ key moves the camera in a smooth trajectory (e.g., a circle or B-spline) around the terrain. The look-at position is the center of the terrain.
- The default is to display the terrain using diffuse gray color for all triangles. Pressing the ‘s’ key switches between smooth and faceted (flat) shading. Pressing the ‘c’ key switches to color mode, where each triangle vertex is colored according to its height (described below).
- By pressing the following keys the user can choose between the different simplification strategies (described in detail below):
- Press ‘s’: Static simplification. The mesh was preprocessed using Heckbert and Garland’s view-independent QEM.
- Press ‘d’: Dynamic simplification using ROAM and Lindstrom’s view-dependent delta metric. The mesh is preprocessed into a triangle bintree and dynamically simplified during rendering.
- The level of static or dynamic simplification is selected by pressing the number keys:
  - Press 1: Original, high resolution mesh
  - Press 2: Mesh with $\frac{1}{2}$ the number of triangles.
  - Press 3: Mesh with $\frac{1}{3}$ the number of triangles.
  - Press 4: Mesh with $\frac{1}{4}$ the number of triangles.
  - Press 5: Mesh with $\frac{1}{5}$ the number of triangles.
For dynamic simplification this is roughly the number of triangles drawn per frame.
- The current simplification strategy (‘static’ or ‘dynamic’), the number of triangles in the current frame, and performance in frames per second are displayed as text in the display window.

3. Grading

The assignment will count for a total of 100 points.
- 20 points - Read the terrain file and display it in the interactive viewer using the mouse for navigation as specified above.
- 25 points – Implement static simplification and QEM.
- 35 points – Implement triangle bintrees and ROAM split queues.
- 20 points – Implement the view-dependent delta metric.
For extra credit you can implement the following extensions:
  - 20 points – Implement the ROAM merge queue.

After the assignment has been submitted, please send an email to jeroen@merl.com. Assignments submitted after the due date will not be graded.

4. Height Field Data

Figure 1 A height map and corresponding adaptively sampled terrain mesh
The height field is given as a grayscale image, where pixel values between 0 and 255 correspond to height. Figure 1 shows an example of the height field and the mesh rendered from a particular viewpoint. You need to read in the grayscale image file and create a triangle mesh according to the two strategies outlined in sections 5 and 6.

To color the terrain mesh according to the height field, you can use a mapping of vertex height to color for each triangle vertex. If you make the lowest elevations green, medium elevations rock-colored, and the highest elevations snow white, the result corresponds to realistic terrain features.

5. Static Simplification and QEM

To simplify the height field using QEM you should implement a generic mesh data structure, such as winged-edge or directed edges. Fortunately, the triangle mesh of a height field is very regular. To make comparison with ROAM and dynamic simplification fair, you should use a triangulation of the mesh that is the same as the highest resolution of a triangle bintree (Figure 2).

![Triangle bintree mesh](image)

**Figure 2 Triangle bintree mesh**

It’s up to you if already decide to implement the triangle bintree now or if you do that later. Given the regular structure of the mesh you can quite easily store it directly in, for example, the winged-edge data structure. A winged edge or directed edge mesh data structure makes it easier for you to compute QEM, where you need to find neighboring vertices and faces.

Given the high resolution mesh, you can generate the simplified mesh using edge collapses together with the QEM cost metric. See the QEM paper for a detailed description of edge collapses and QEM. The basic algorithm works as follows. We start with a highest resolution mesh and for each vertex we determine the distance to all the planar triangles adjacent to that vertex. Garland shows that we can represent the squared distance of a vertex to a plane with a fundamental error quadric matrix $K_p$. The error quadric $Q$ at the vertex is the sum of all the $K_p$’s from its adjacent triangles. Initially this error is zero, since for the highest resolution mesh the distance of the vertex to each of its adjacent triangles is zero (the vertex lies on the triangle!).

Next we need to determine the edge with the smallest error. If we were to collapse this edge, it would result in the smallest error between the simplified model and the original.
For each edge, we compute a new optimal vertex $V_{\text{bar}}$ by solving equation (1) in the paper. This involves a matrix inversion. Then we compute the sum of the quadrics of the two vertices of the edge, $Q = Q1 + Q2$. The cost of collapsing this edge is then $e = v_{\text{bar}}^T Q v_{\text{bar}}$. The edge with the minimum error $e$ is contracted first. After every edge collapse we update the quadric of the new vertex $v_{\text{bar}}$ with $Q1 + Q2$. Garland explains that we can approximate the error by adding the matrices representing the quadrics for the two vertices involved in the edge collapse. We continue this process until we have reached our specified number of triangles. For example if we want (approximately) a mesh with only ½ the number of triangles than the original, we can stop as soon as we are equal or smaller than that number.

You do not have to implement the edge split, the inverse of an edge collapse, which would take a mesh back to a higher resolution model. Instead, you may start from the highest resolution mesh every time a number key is pressed. The simplified mesh should then be displayed in the window. You may want to update the display after every edge collapse to visualize the simplification.

6. Dynamic multi-resolution terrains and ROAM

To do dynamic multi-resolution terrains, hence forward simply referred to as dynamic terrains, you need to implement triangle bintrees as described in the ROAM paper. We start with a base diamond (two triangles), as described in the paper, and recursively apply the binary triangle splits as described in section 4.1 of the paper. We continue this until we arrive at some maximum level for the bintree. This really is a pre-processing step to the dynamic algorithm in which we fully populate the triangle bintree. For example, suppose our highest resolution mesh should contain 16 triangles. The bintree and corresponding meshes are depicted in the following figure.

![Figure 3 The triangle bintree and corresponding meshes](image)

We call the mesh in level 4 a 4x4 mesh. Figure 2 shows an 8x8 mesh. As described in the paper, we have created vertices $(u,v)$ in the domain space for the mesh. In other words we need to assign $(x,y,z)$ values to each vertex in the mesh, and we do this by taking $(x=u,y=v,z=w(v))$. Here $w(v)$ means that we assign the height based on the value we find in the height map.
The elegant property of the triangle bintree is that we also have coordinates for all the coarser level meshes, because the vertices of these coarser levels are all part of the highest level. Figure 6 of the paper shows this nicely in 2D.

From the highest resolution mesh we calculate an error delta for each triangle. The delta essentially represents how much the height field changes between the different resolution levels. For the highest resolution mesh these deltas are set to zero (much like the initial error in QEM is zero for the highest resolution mesh). See Section 6.1 below for a detailed explanation of how to calculate the deltas for the triangles of consecutive coarser levels.

Once the deltas have been calculated, we can use them to determine the priority for the triangles. This is the point where we begin the runtime part of the ROAM algorithm. The triangle with highest delta has highest priority. Careful: this is contrary to the QEM error! Remember that delta measures how bad the triangle represent the height field. If it is high, we want to split and refine it. For each frame, we start with the base diamond and add both triangles to a so-called split queue. We determine the highest priority triangle from this queue and we perform a forced split on this triangle (see Section 4.2 of the paper). The newly created triangles are added to the split queue, while the split triangle is removed from it. We update the priorities (deltas) and determine the highest priority for the new set of triangles.

We continue this process until we satisfy our limit and arrive at a triangulation that is optimal for that given limit. The limit in our case is the maximum number of triangles for each frame. For the next frame in the animation, or for a new viewpoint, we start again with the base diamond and execute the algorithm as described previously.

For this part of the assignment you will only implement the split queue of the ROAM algorithm. The merge queue is for extra credit, see section 7.

### 6.1. Lindstrom’s View-Dependent Error Metric

You need a view-dependent error metric to decide when to split a triangle. The triangle wedgies of the ROAM paper would work, but are somewhat complicated to implement. Instead, we want you to use the “delta” metric described in the following paper:


(see [http://www.cc.gatech.edu/gvu/people/peter.lindstrom/](http://www.cc.gatech.edu/gvu/people/peter.lindstrom/))

The basic idea is simple, and then there are some efficient ways to compute it. Again, do not worry too much about efficiency.

V1 and V2 are two vertices along the base edge of a triangle in the bintree. You first compute the midpoint Vm of that edge as V1 – V2 / 2. Delta is the vector between the midpoint Vm and the value of the height field at that location (W(Vm)). Note that delta corresponds to the line segment VmW(Vm). The error of the approximation is then defined as the magnitude of the projection of the delta vector into screen space.
Lindstrom’s paper describes how you can efficiently compute this projection by considering the worst case or maximum delta projection for all vertices of a particular recursive level. Instead, you can use the OpenGL modelview and projection matrices (use glGet(GL_PROJECTION_MATRIX) and glGet(GL_MODELVIEW_MATRIX)) and project each delta vector to screen space. Make sure the projections are not outside of the viewing frustum. (What error do you assign in that case?)

7. Extra Credit

For the extra credit part of this homework we ask you to implement the merge queue of the ROAM algorithm. ROAM’s efficiency largely depends on what is called frame-to-frame coherency. In general the visual information in the next frame of an animation is largely the same as in the current frame. Consider for example a camera pan which is not too fast. As the camera pans the larger part of the image will show the same content, except at the edges of the screen things will disappear on one side, and appear on the other side.

ROAM exploits frame-to-frame coherency by taking the triangulation of the previous frame as a starting point for the triangulation of the current frame instead of building a triangulation starting from the base diamond for every frame. For this, you need to implement a merge queue in addition to the split queue. Given a certain limit, for example triangle count or accuracy, for one frame, this limit may be violated for the next frame and we might have to back up to a coarser terrain mesh for that frame. We would have to merge triangles that were previously split. Section 5.2 of the ROAM paper describes the algorithm when both a split and merge queue are used in the triangulation.

We do have to warn you that this extra credit section is a challenging exercise!