1. Motivation

As the computational power of both personal computers and game consoles continues to increase, it is becoming increasingly possible to offload complex tasks, such as terrain mesh generation, onto the end-users of these products. The increased power of modern GPUs allows for the easy rendering of crisper, more detailed three-dimensional objects, objects that are currently either pre-calculated and stored in formats that difficult to edit or stored in an intermediary form that is then computed into a final mesh at runtime.

Terrain is a prime example of a data set that is computed into a finalized model as the application is loaded. The most common model, however, tends to produce comparatively ugly and inaccurate representations of terrain, using more triangles than would otherwise be necessary to produce a representation that does not necessarily capture all the important features of the terrain. It is therefore the goal of this paper to take advantage of the computational power available today to generate a more realistic terrain representation from existing intermediary terrain data.

2. Prior Work

The most frequently used format for terrain mesh representation in video games is the digital elevation model, or DEM, which is also referred to as a height-map. These height-maps take the form of uncompressed images that map elevation values to grayscale pixels. In video games, flight simulators, and other real-time simulations that require representation of geographic data, these elevation values are directly mapped to a uniform grid of vertices, which produces a dense mesh of triangles that represents the terrain in three dimensions.

Another popular method of representing terrain data is the triangulated irregular mesh, or TIN, which is a mesh of irregularly spaced points that can more accurately represents terrain features. The points in these meshes are calculated through various means, depending on the implementation, but generally produce lower-density meshes with detailed clusters of points along important features while reducing triangle count in largely unvaried surfaces. Triangulation is typically performed by constructing a two-dimensional Delaunay triangulation of the generated points. (Peucker, et.al., 1978)

3. Algorithm

The algorithm I use to generate contour-driven terrain meshes consists of three main parts: Contour angle calculation, local contour derivative calculation, and weighted point distribution. Triangulation is performed using existing techniques and is, as such, not the focus of this paper.

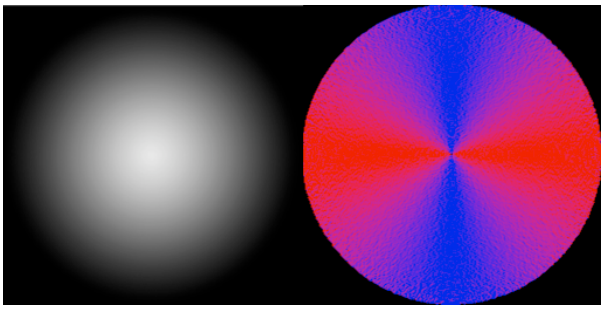

Fig. 3.1.2: Contour angle visualization for linearly distributed circle.

3.1 Contour Angle Calculation

The most important step of the algorithm is to calculate the angle of the contour at every point in the mesh. These values are calculated and stored on a per-pixel basis in an array that is the same size as the input data. The vertical slope between each pixel in the input data and its neighbors is calculated, then its inverse is used to create a normal vector for a hypothetical surface that bridges the two points. These vectors are first normalized and then added together to produce an average normal for the point. The vertical component of the vector is discarded and the vector re-normalized. By discarding the vertical component, we have eliminated the variance in the vector from the relative distance in height and left only the normal to the contour at the given point. The angle of this two-dimensional vector can then be trigonometrically calculated and then stored in an angle map. As this operation must be performed for each pixel in the input DEM, this step of the algorithm runs in O(n) time, where n is the number of pixels in the input data.

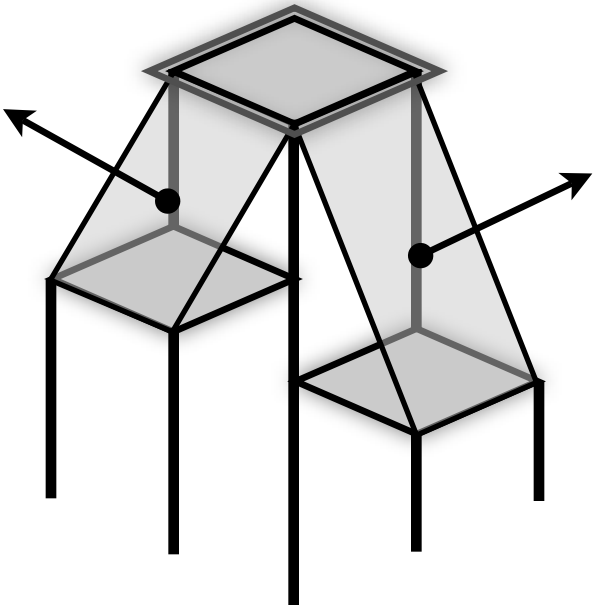

Fig. 3.1.1: Normal vector calculation from neighboring pixels.

Due to the limitations of storing DEM data in integer format, it is unfortunately frequent that a comparatively large segment of the terrain may be represented by a single value. This makes impossible the calculation of angle, which relies on a change in vertical height to detect slope. This results in large regions where there is no stored angle, which results in large discrepancies in final point weight. As a result, a pixel-filling algorithm is used. For each region of the map that contains a relative plateau and thus no angles, the

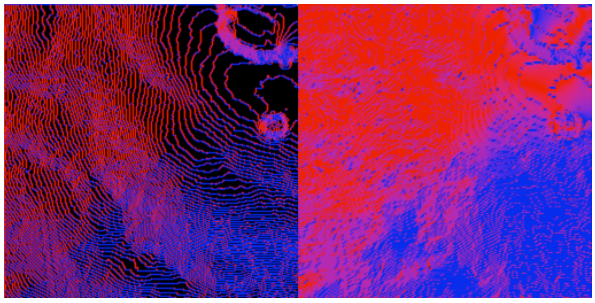

Fig. 3.1.3: Left: angle map before pixel fill; Right: angle map after pixel fill

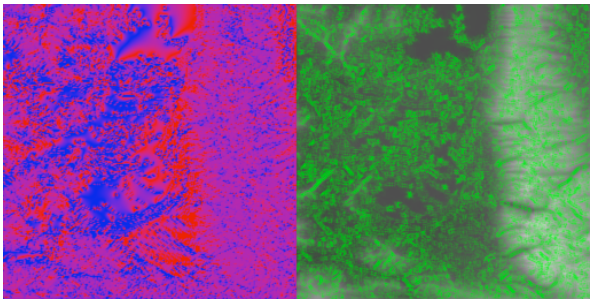

Fig. 3.2.2: Left: angle map of sample data; Right: derivative of contours. Shades of green indicate higher contour variation.

pixels immediately bordering the region are identified and marked. The angle of each "empty" pixel is then calculated as the average of all bordering pixels, as weighted by the inverse square of the distance. This tends to produce areas that have similar angle and, thus, minimal contour variation, throughout, except in the cases of very small regions bordered by steep changes.

3.2 Contour Derivative Calculation

A separate array is then created to store the local rate-of-change in the contours of the image. For each pixel in the angle map, we first take the height of the pixel. Then, for each pair of neighboring pixel, we test to determine if the height value of the current pixel lies in between the values of the two test pixels. If so, we linearly interpolate the angles of the test pixels between those two points to get an approximate angle of the contour at the given height. As this algorithm, again, only visits each pixel in the angle map a handful of times, it can be run in O(n) time.

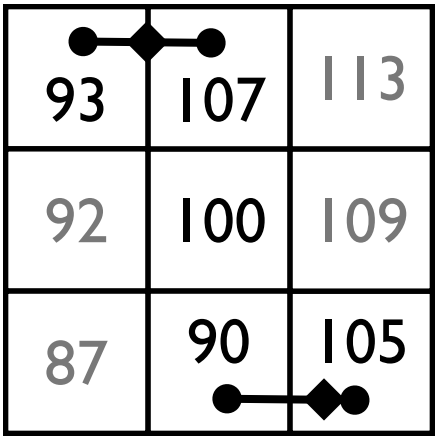

Fig. 3.2.1: Linear interpolation of pixel height

3.3 Mesh Point Spacing

Finally, the points in the mesh are spaced throughout the grid in a manner such that points are clustered around areas where the contours vary greatly while leaving relatively unchanging areas with very few points. This is done by sweeping through the grid, determining at each pixel if there is a point already placed within a certain radius, and placing a point if there is not. The minimum distance between points at any given place in the mesh is determined based on both a user-defined coarseness value and the rate of change of the contour at the given point.

4. Results

Unfortunately, the limited capabilities of our input data proved to be quite problematic for our results. The crucial issue that we discovered was the plateaus that were artifacts of the limited data range that bitmaps were able to provide us. The nature of these plateaus made it impossible to smoothly calculate the angle of contour, resulting in angles that widely varied from pixel to pixel in very small regions. As a result, the contour derivatives tended to show thin bands of very rapid change, which drastically increased the number of points placed in relatively unimportant areas. Areas with even small amounts of variation created extremely rough points of terrain, resulting in very rough, visually distorted areas in the generated mesh.

Apart from these errors, however, points were placed accurately on peaks and ridges, especially where the angle of the contour changed rapidly.

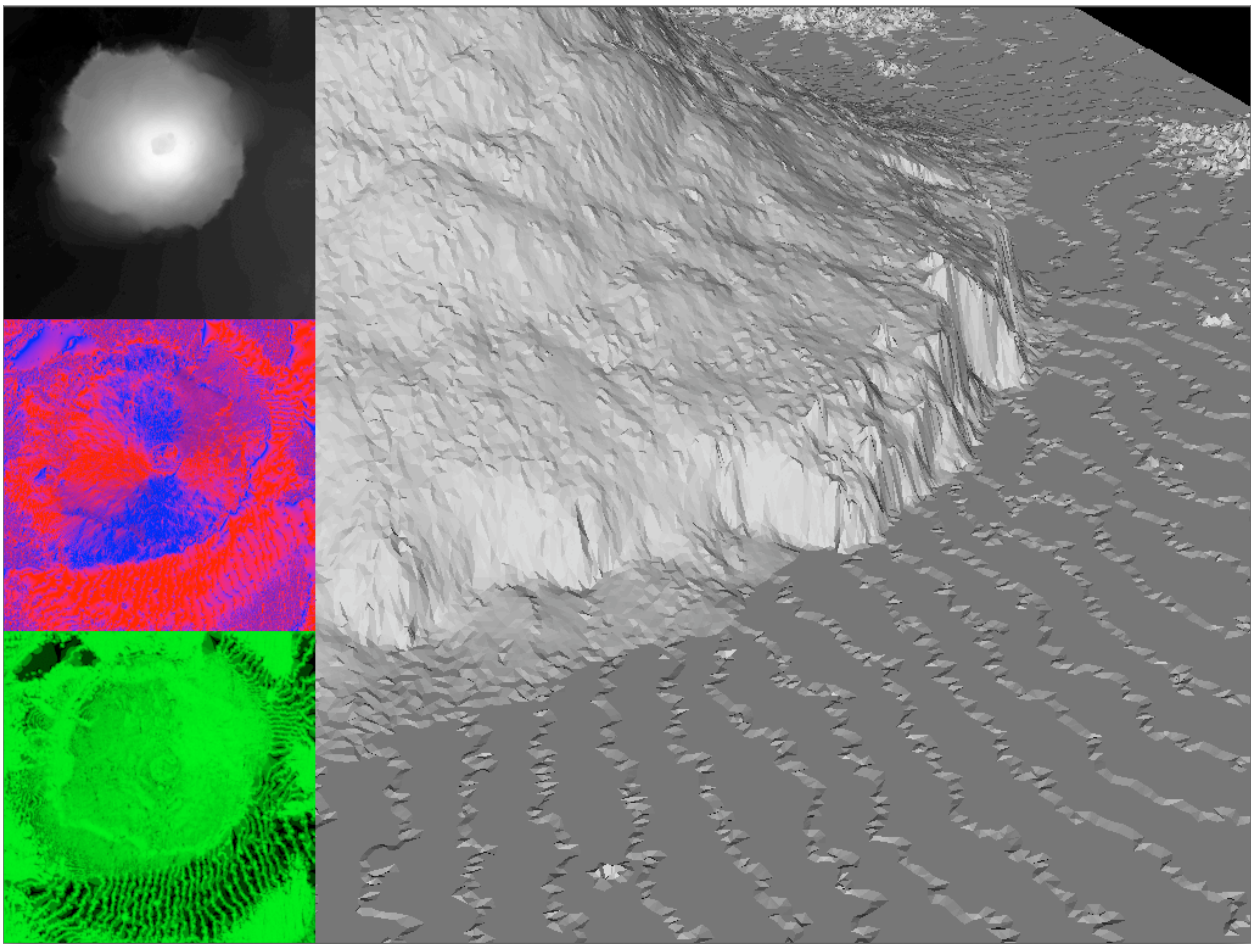

Fig. 4.1: Contour-driven representation of terrain. Note the triangle banding along the flat areas and the rough areas in the upper-right: artifacts produced by inaccuracies in the calculations brought on by integer clamping.

5. Conclusions

There are many improvements that can be made to this algorithm. Potentially the most helpful would be a plateau-detection algorithm that attempted to interpolate height values over flat surfaces to allow for the smoothness of ridges. In addition, the point weights generated by this technique may be supplemented by weights generated by studying the changes in vertical slope, which would decrease triangle count in areas where there are rough variations in contour but the height remains relatively constant.

6. Bibiliography

T.K. Peucker, R.J. Fowler, J.J. Little, D.M. Mark. "The Triangulate Irregular Network", American Society for Photogrammetry Proceedings, 1978.

R.J. Fowler, J.J. Little. "Automatic Extraction of Irregular Network Digital Terrain Models", Proceedings of the 6th annual conference on Computer graphics and interactive techniques, 1979.

D.T. Lee, B.J. Schachter. "Two algorithms for constructing a Delaunay triangulation". International Journal of Parallel Programming 9-3, 1980, pg. 219-242.

Delaunay triangulation library written by Sjaak Priester and distributed under GNU general public license. Accessed from http://www.codeguru.com/cpp/cpp/algorithms/general/article.php/c8901 on April 22, 2009.