Topics in Graphics and Visual Computing

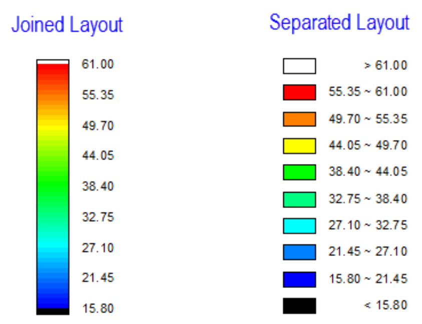
Yahav Levy 207654765, Tzvi Greenfeld 209300581

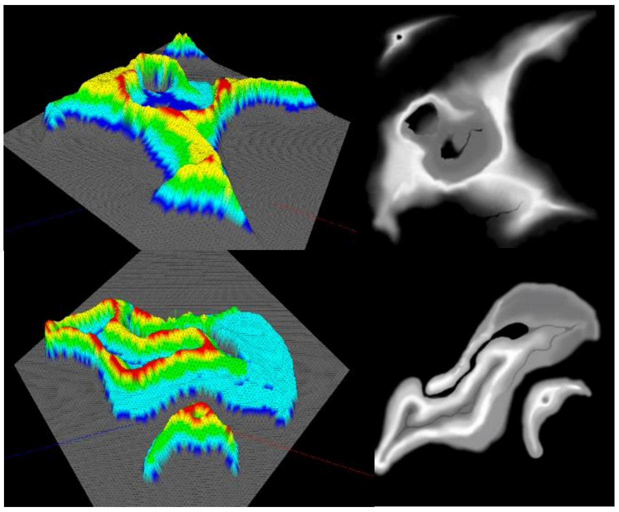
3D mesh by altitude interpolation

Triangle:

An object representation for each of the triangles in grid. Including rendering in different ways and positioning, the position of each triangle is derived from the matrix object of the input image, and step size. The height of each of the triangle's vertices is determined by the color intensity in that point in the image.

The number of triangles in the map is determined by sampling the image at intervals of \STEP_SIZE\ pixels and creating two triangles for each sample point. The intensity of the color at each corner of sample point is mapped to a value in the range [0, 61] and used to color the vertices of the triangles using a color map.





Picking:

This function is a color picking implementation. By calling each triangle's "drawIdColor" method, which is the implementation from openGL documentation. After getting the picked triangle id, its being pushed into a vector and if the size of this vector is 2 we run Dijkstra's algorithm to find lightest path from the first picked triangle to the second.

Keyboard input:

The function "handleInput" checks if the user pressed any key and perform transformations and activate effects accordingly.

'A', 'D': rotation around the y axis (left-right)

'W', 'S': transformation along the y axis (up-down)

'Q', 'E': rotation around the vector (-1, 0, 1)

Up and Down arrows / mouse wheel: scaling

'F': increase fog density

'G': remove fog

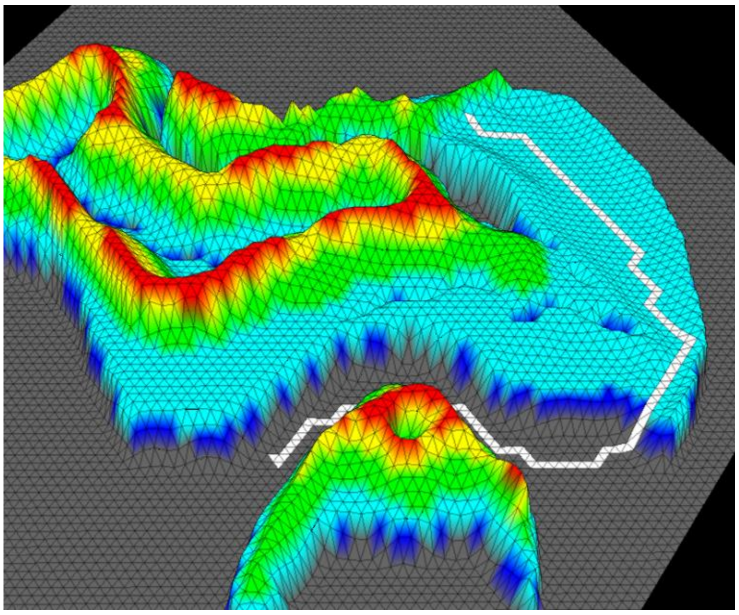
Path navigation:

In our project, we developed an algorithm that determines the easiest route between two triangles selected by the user on a map. Instead of finding the shortest path, we prioritized ease of travel by considering the terrain, including mountains.

To optimize the calculation, we used our conversion algorithm (mentioned below) that maps triangles in the high-resolution map to their corresponding triangles in the low-resolution map, and vice versa. After the user selects triangles in the high-resolution map, the algorithm converts these to the corresponding triangles in the low-resolution map.

Next, the algorithm applies the Dijkstra algorithm to these converted triangle picks and determines the easiest route between them based on the height gap between neighbouring triangles. The resulting path on the low-resolution map is then converted back to the high-resolution map using the conversion algorithm.

Finally, to connect the triangles in the determined path, we run a breadth-first search (BFS) algorithm on every pair of triangles. This allows us to effectively fill in the gaps between the triangles and complete the easiest route on the high-resolution map.



Graph:

This is a mathematical undirected weighted graph implementation of our triangles grid.

The graph represented with adjacency list, and the weights are also nested vectors of the same structure as the adjacency list.

Each triangle is a vertex in the graph and there's an edge between every pair of triangles that shares a side. The weights are Euclidian distance between the center of the triangles.

Graph::BFS(int I ,int j):

An implementation of BFS algorithm on the graph which returns a path (non-weighted) between triangle with id 'i' and triangle with id 'j'. we use HashMap to keep track of the path (represented by indices of triangles)

Graph::dijkstra(int start ,int end):

An implementation of Dijkstra algorithm. Using "Vertex" and "VertexComapre" struct we use priority queue based on distance from "start" node until the "end" node found. Keeping track of the path in the same way we did in BFS.

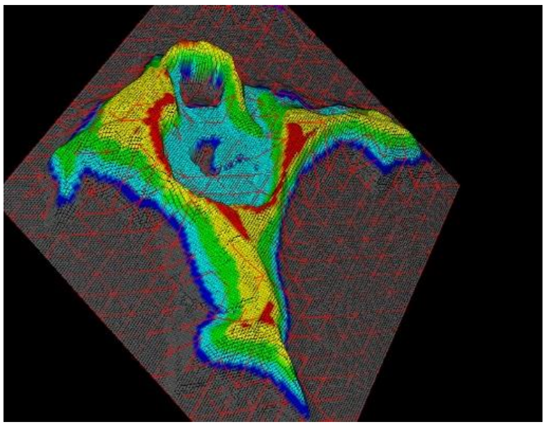
ResMapper:

To improve performance, we implemented a multi-resolution feature that uses two separate sets of triangles: one with a small step size (fine grid) and one with a larger step size. The screen is rendered using the higher resolution grid, but when calculations such as path finding are needed, we map triangles in the fine grid to their corresponding triangles in the large grid by finding the triangle that contains the middle point of the fine triangle.

This object is used to map triangles from low resolution to high resolution and vice versa.

When translating from low (large triangles) to high (small triangles) we take the middle point of the input triangle and loop over small triangle until we find one that contains the middle point of the input triangle.

Translation from high to low is almost identical, except for looping over large triangles instead of small.


Outline of low-resolution grid on top of high-resolution grid

Environmental Simulation:

Weather: Controller object for an environmental simulations implementation of fog.

In the fog implementation we defined a function called initFog that initializes the fog effect in an OpenGL environment by setting the fog color, density, and start and end distances. The addFog function increases the density of the fog, and the removeFog function sets the fog density to 0. These functions are called in the main program to control the visibility of the fog