Bryant Pham

0983058

Assignment 3

1. Below will detail an algorithm to compute 2D iso-contours on a triangle mesh

Let Tall be the set of all triangles with 3 edges that make up the triangular mesh.

Let S\* be the iso-contour scalar value.

For each T in Tall

For each edge E in T

Check if S\* intersects with edge E

Increment intersection counter

If there are 0 intersections:

Continue to next triangle

If there is 1 intersection and both vertices are equal to S\*:

Connect the two vertices with a line

If there are 2 intersections:

Calculate interpolated x\*, y\*, z\* values from the two edges

Connect the two edges with a line at the x\*, y\*, and z\* values

# Design and Implementation

The design and implementation of my algorithm consists of

1. Loading the grid data
2. Building the quads
3. Calculating the intersections and drawing contours

## Loading the grid data

Loading the grid data is done in the ***load\_data\_on\_uniformGrids*** function. From a specified DAT file, this function retrieves the number of nodes in each axis (NX, NY) and coordinates/scalar data for each node and stores them in an array. The x, y, z coordinates and scalar data is captured in the following *Node* object with associated attributes and data types:

*Node*

x: float

y: float

z: float

s: float

index: int

## Building the quads

Building each quad is done in the ***buildQuads*** function. Given an array of nodes in the uniform grid, iterate through each row and calculate indices for each corner of the quad respective to the node in the grid. Because the nodes are given in reverse order (from bottom to top), the corner indices are given by (assuming x represents the current iteration’s index):

bottomLeft: x

bottomRight: x + 1

topLeft: x + NX

topRight: x + 1 + NX

Each quartet of corner indices pair to form edges of a *Quad* in the following data structure:

*Quad*

edges: [

{v1: topLeftNode, v2: topRightNode},

{v1: topRightNode, v2: bottomRightNode},

{v1: bottomRightNode, v2: bottomLeftNode},

{v1: bottomLeftNode, v2: topLeftNode}

]

Create quads from nodes in each row and store them in an array.

## Calculating the intersections and drawing contours

Calculating intersections and drawing contours is done in the ***drawIsoContour*** function. Iterate through each quad and determine if the iso-contour scalar *S\** intersects with each edge. Intersections are stored in an *intersectingEdges* array:

*intersectingEdges*

[

{v1: edge23Node1, v2: edge23Node2},

{v1: edge13Node1, v2: edge13Node2},

{v1: edge01Node1, v2: edge01Node2},

{v1: edge02Node1, v2: edge02Node2}

]

If *S\** is intersecting only one edge and both vertices of the edge are equal to *S\**, draw a line through the vertices of that edge.

If *S\** intersects two edges, interpolate the x\* and y\* coordinates for both edges and draw a line through the interpolated coordinates.

If *S\** intersects four edges, then calculate the mean of all the vertices *M* and compare to *S\*.* Edges 01, 02, 13, 23 correspond respectively to element indices 2, 3, 1, 0 to the *intersectingEdges* array. If *S\** < *M*, connect edge 01 to edge 02 and connect edge 13 to edge 23. If *S\** > *M*, connect edge 01 to edge 13 and connect edge 02 to edge 23.

Line segments are immediately placed into the buffer for rendering and are not stored.

## Iso-Contour Implementation for Triangle Meshes

The implementation for triangle meshes is nearly identical to the implementation for quads. The difference is that you build triangles and determine if S\* intersects with any of the three edges of the triangle. Below is the data structure used to represent a triangle:

*Triangle*

edges: [

{v1: corner1, v2: corner2},

{v1: corner2, v2: corner3},

{v1: corner3, v2: corner1},

]

After building the full set of triangles, the ***drawIsoContour*** function can be reused to draw contours on triangle based meshes.