Luke Plewa

3 graphics technologies: specialized graphics feature - map editor
keyframe animation
level of detail – mesh simplification

Mesh Simplification Algorithm:

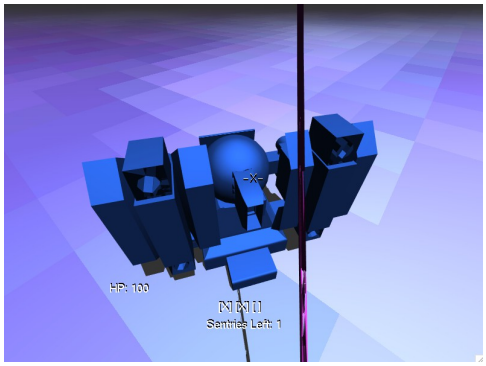
implementation contained in src/mesh.cpp: void myMesh::simplify()

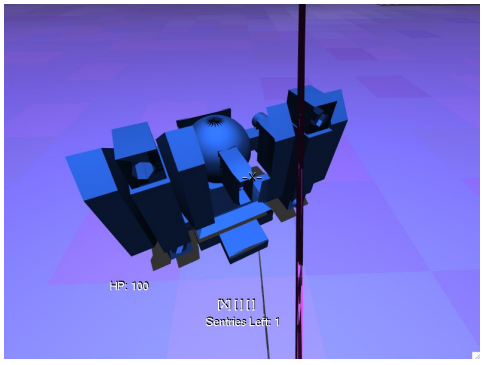
Simplify the mesh by removing the faces which share the smallest edges. I pick a good minimum size for an edge (after testing I chose 0.35f), and remove all faces who share edges which are smaller than that. The vertices which were part of those shared edges are then replaced by the midpoint of the edge. This implementation reduces the number of faces but doesn't reduce the number of vertices. The simplified model is swapping with the high resolution model when the player is far away from the position of the models.

```
for all faces i, j
    if i!=j && !marked[i] && !marked[j] // marked when already decimated faces
        for all vertices m, n on faces i, j
            if m == n count++ // the faces share a vertex
        if count > 2 // the faces share 2 vertices aka an edge
            if length of edge < min edge length // min edge length is a preset constant
                vertices[p1 of edge] = midpoint of p1,p2
                vertices[p2 of edge] = midpoint of p1,p2
                marked[i] = true
                marked[j] = true
for i in all marked
    if marked[i]
        faces[i].remove // remove faces whose edges have been collapsed
```

Results: these get printed on every run to show the difference in faces removed

Ninja Current size: 11594. New size: 6781
Territory Current size: 1504. New size: 1376
Tower Current size: 41688. New size: 21443
Sentry Current size: 4736. New size: 3646
Sword Current size: 1856. New size: 1081





The original mesh is on the left, the simplified mesh is on the right
These meshes have a difference of 1,090 faces!

Sources: http://www.cse.psu.edu/~shontz/imr_2012.pdf

CPU-GPU Algorithms for Triangular Surface Mesh Simplification (Shontz, Nistor 2012)

The Code:

```
void myMesh::simplify()
{
   printf("Current size: %d. ", mFaces.size());
   printf("Starting mesh simplification... ");
   int count;  // the num of shared vertices * 2
   int p1, p2; // the numbers of the vertices to be replaced
   bool *marked = (bool*)malloc(sizeof(bool)*mFaces.size()); // a face has been marked
   float diff = 0; // the length of the edge for comparison
   int count_min = 2; // the faces share 2 points each
   int temp_point = 0; // temporary point
   float min_edge_distance = Constants::getInstance().getValue("mesh_decimation_limit");
   glm::vec3 A, B, midpoint;

   for(int i=0; i<mModel->mVertices.size(); i++){
      mVertices.push_back(mModel->mVertices[i]);
   }
   for(int i=0; i<mFaces.size(); i++){
      marked[i] = false;
   }

   //iterate over all faces
   for(int j=0; j<mFaces.size(); j++){
      for(int i=0; i<mFaces.size(); i++){
         if(i != j && !marked[i] && !marked[j]){
            count = 0; // check if they share an edge (share two vertices)
            for(int v=0; v<3; v++){ // iterate through vertices
               for(int n=0; n<3; n++){
                  temp_point = mFaces[j].V[v].mVertex;
                  if(count < 2 && temp_point == mFaces[i].V[n].mVertex){
                     if(count == 0) p1 = temp_point;
                     else if(p1 != temp_point) p2 = temp_point;
                     else count--;
                     count++;
                  }
               }
            }
            if(count >= count_min){ //create midpoint of vertices
               A = mModel->mVertices[p1-1]; // shared point 1
               B = mModel->mVertices[p2-1]; // shared point 2
               midpoint = vec3(0.5f*(A.x+B.x),
                  0.5f*(A.y+B.y),
                  0.5f*(A.z+B.z));
               if(length(A - B) <= min_edge_distance){
                  mVertices[p1-1] = midpoint;
                  mVertices[p2-1] = midpoint;
                  marked[i] = true; //mark the faces for removal, because the 2 points have been collapsed
                  marked[j] = true;
               }
```

```
            }
        }
    }
}
// delete the marked faces
int x = 0; // the offset count because we are removing elements
for(int j=0; j<mFaces.size(); j++){
    if(marked[j]){
        mFaces.erase(mFaces.begin()+x); // remove marked faces
        x--;
    }
    x++;
}
printf("...finished mesh simplification. ");
printf("New size: %d\n", mFaces.size());
}
```