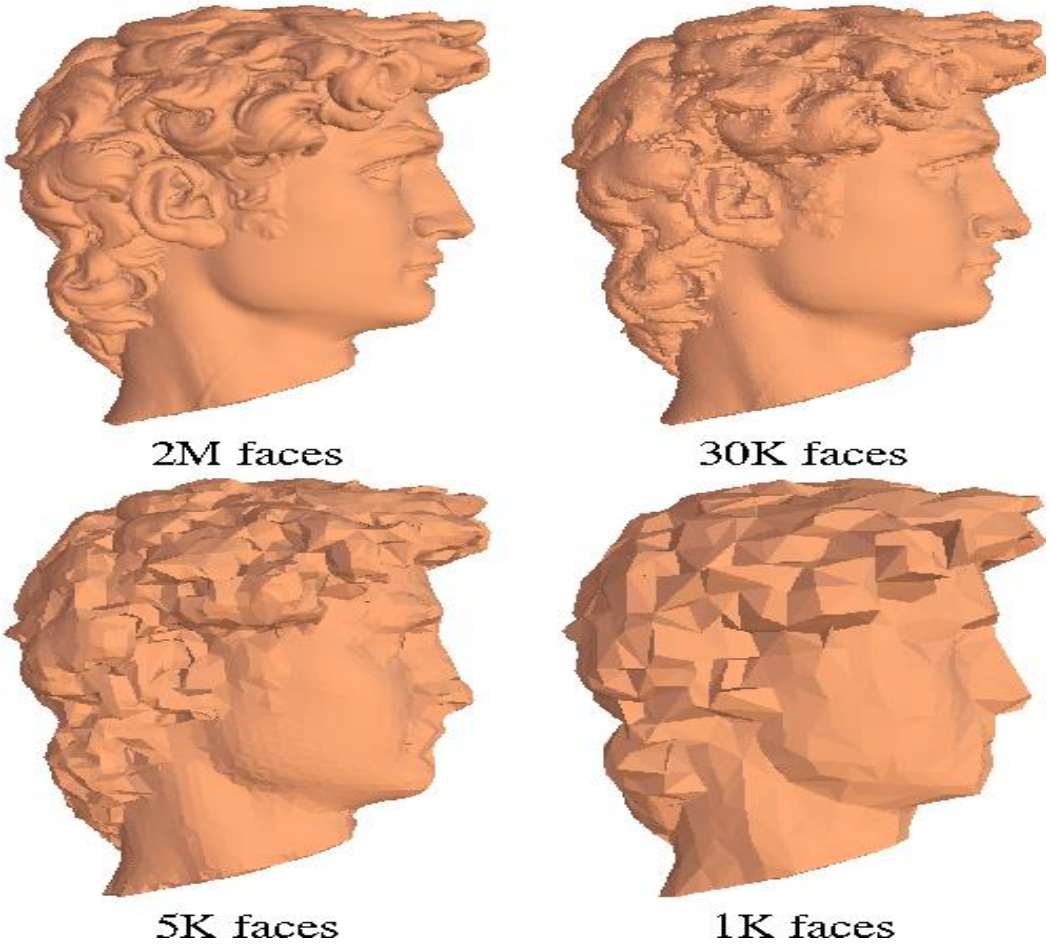
Assignment 1 – Mesh Simplification

You will implement the paper of Surface Simplification Using Quadric Error Metrics by Michael Garland and Paul S.heckbert (You can find a link in class material section on the site)



Set up

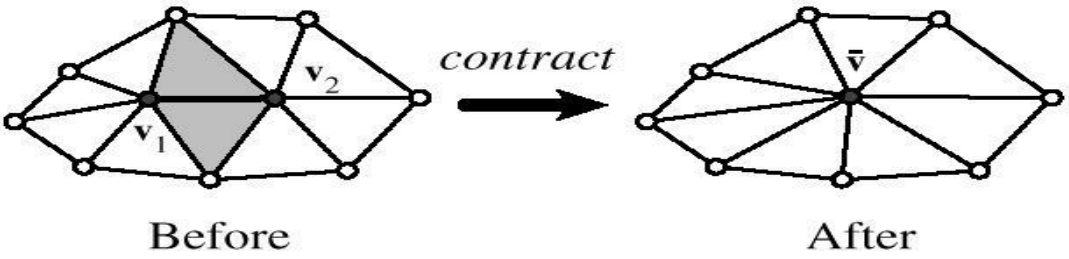
1. Download the project from useful links on the course site
2. Download GLFW version, suitable to your compiler and replace the glfw3.lib file on ..\ClassExample2\ClassExample2\res\libs
3. Download GLEW zip project and compile it with your VS to libs. Replace glew32s.lib on ..\ClassExample2\ClassExample2\res\libs
4. Open ClassExample2 project clean it and rebuild it.

Data base (recommended)

5. Add additional variable to shape class which contain the maximum number of faces (triangles) in shape.h.
6. Add Edge struct/class which contains two iterators to OBJIndex and error (double). The error will represent the error of a new vertex exactly in the middle of the edge $v_{new} = \frac{(v_1+v_2)}{2}$.
7. Add vector of edges to OBJModel class
8. Add multimap which contains mapping of every vertex to its neighbors.

Algorithm implementation (mandatory)

9. Your program will read mesh data from OBJ file, construct a simplify mesh and save it as IndexedModel in the Mesh class.
10. Calculate the error of merging two vertices to one vertex in the middle of every edge of the mesh.



11. Use STL heap algorithm to choose the minimum error edge.
12. Replace the two vertices on the merge vertex.
13. Delete the two faces corresponding to the edge you choose, from the OBJindices list.
14. Delete edge
15. Update new vertex edges errors ($Q = Q_1 + Q_2$). You may choose to update edges error when pop it (from heap). In this case you must check if the error after update is (still) lower from the heap top edge error.
16. Repeat 9-15 until you reach the required number of faces.
17. Render the simplify mesh beside the original mesh each one rotating slowly around its own y axis.

Additional topics

1. **Submission in pairs to the submission system. You can submit your whole project (recommended) or only the files you change or add.**
2. You can find explanation for Q calculation and edge (new vertex) error calculation in the slides and the paper about mesh simplification.
3. **Bonus (10 points)**: instead of taking the merge vertex in the middle of the edge choose the optimal (minimum error) position of the merge new vertex on the edge (see section 4 in the paper)
4. You can check your program using the OBJ files in the site.
5. Change the OBJModel::CalcNormal function to

```
void OBJModel::CalcNormals()
{
     float *count = new float[normals.size()];
     for (int i = 0; i < normals.size(); i++)
     {
            count[i] = 0 ;

     }
    for(std::list<OBJIndex>::iterator it = OBJIndices.begin(); it !=
OBJIndices.end(); it++)
     {
         int i0 = (*it).vertexIndex;
           (*it).normalIndex = i0;
           it++;
         int i1 = (*it).vertexIndex;
           (*it).normalIndex = i1;
           it++;
         int i2 = (*it).vertexIndex;
           (*it).normalIndex = i2;

         glm::vec3 v1 = vertices[i1] - vertices[i0];
         glm::vec3 v2 = vertices[i2] - vertices[i0];

         glm::vec3 normal = glm::normalize(glm::cross(v2, v1));

           if(count[i0]==0)
                  count[i0]=1.0f;
           else
                  count[i0] = count[i0]/(count[i0]+1);

           if(count[i1]==0)
                  count[i1]=1.0f;
           else
                  count[i1] = count[i1]/(count[i1]+1);

           if(count[i2]==0)
                  count[i2]=1.0f;
           else
                  count[i2] = count[i2]/(count[i2]+1);

           normals[i0] += normal;
         normals[i1] += normal;
         normals[i2] += normal;
     }
    for (int i = 0; i < normals.size(); i++)
    {
           normals[i]= normals[i]*count[i];
    }
    delete count;

}
```