FaceR – Marker

Adnan Maruf

Florida International University

amaruf009@fiu.edu

***Abstract*— This goal of this project is to label feature points, curves, loops on human face (triangular mesh)and to save the feature markers to a separate file (self-defined format) using WebGL. Finally, the project needs to be integrated to GeomSE.**

***Keywords*— GeomSE, Face, Mesh, Feature points, WebGL**

I. Introduction

Though people are good at face identification, recognizing human face automatically by computer is very difficult. Face recognition has been widely applied in security system, credit-card verification, and criminal identifications, teleconference and so on. Face recognition is influenced by many complications, such as the differences of facial expression, the light directions of imaging, and the variety of posture, size and angle. Even to the same people, the images taken in different surroundings may be unlike. The problem is so complicated that the achievement in the field of automatic face recognition by computer is not as satisfied as the finger prints. Facial feature extraction has become an important issue in automatic recognition of human faces. Detecting the basic feature as eyes, nose and mouth exactly is necessary for most face recognition methods[1]. Fig 1 shows some feature points on a human face.

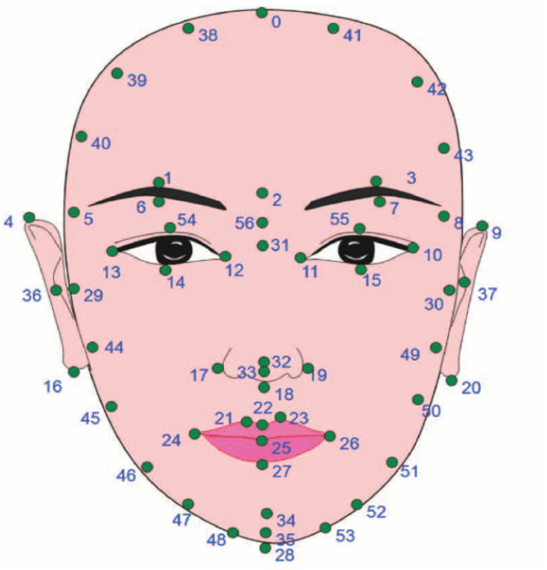


Fig 1. Feature points on face

To mark the feature points on the mesh file, the mesh file needs to be loaded first. This project mainly deals with the .obj file. The OBJ file format is a simple data-format that represents 3D geometry alone — namely, the position of each vertex, the UV position of each texture coordinate vertex, vertex normals, and the faces that make each polygon defined as a list of vertices, and texture vertices. Vertices are stored in a counter-clockwise order by default, making explicit declaration of face normals unnecessary.[2]

To mark a specific point on the mesh, ray casting technique has been used. Ray casting is the most basic of many computer graphics rendering algorithms that use the geometric algorithm of ray tracing. Ray tracing-based rendering algorithms operate in image order to render three-dimensional scenes to two-dimensional images. Geometric rays are traced from the eye of the observer to sample the light (radiance) travelling toward the observer from the ray direction. The speed and simplicity of ray casting comes from computing the color of the light without recursively tracing additional rays that sample the radiance incident on the point that the ray hit.[3]

II. Pipeline

The project has been divided into eight modules. The details of each module is discussed in this section.

A. *Load Object*

To load object file, the vertices and face information have been read directly from the mesh file. For simplicity, the texture information is omitted. There are different type of formats the .obj file stores face information. This project can read all the formats of the .obj file. A sample loaded 3D sphere object is shown in Fig 2.

B. *Interactive View*

In order to mark the points in the mesh file, user must need to rotate and scale the 3D object. Translate is not necessary here. Both rotation and scaling have been added with the mouse movement. Three.js library[4] is used for this interactive view.

C. *Adding spheres in vertex positions*

Each of the vertex point of the mesh has been replaced by a tiny sphere. First the sphere is created and then it's translated to the original vertex position. Each sphere represents a vertex.

D. *Replacing edges with cylinders*

First, the .obj file doesn't have any edge information. So, half edge data structure has been used to find the edges. Each edge is a pair of two vertices. Then using the Thee.js library each edge is replaced by a tiny cylinder.

Fig 3. displays the mesh file after replacing the vertices and edges with spheres and cylinders respectively.

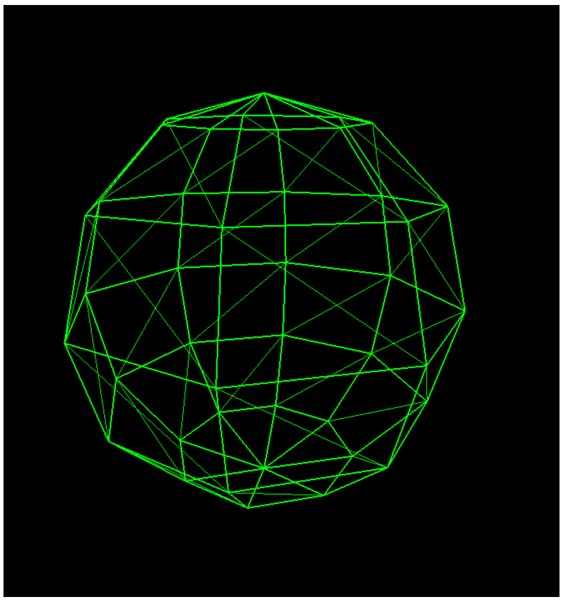


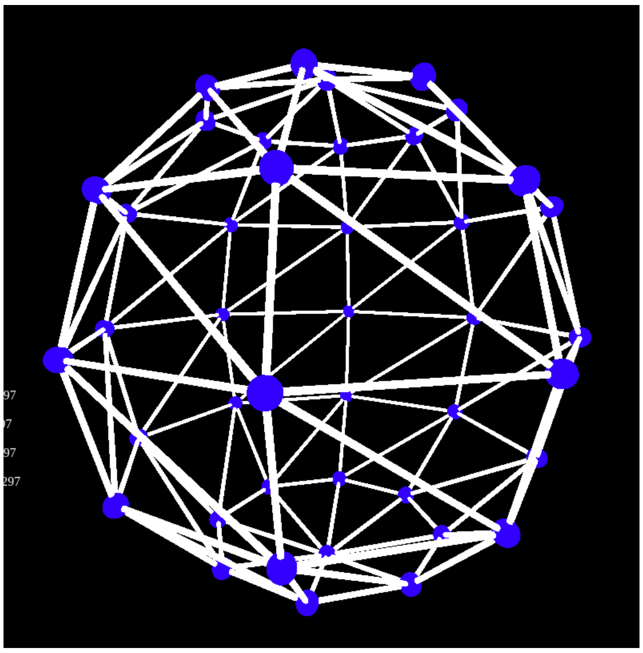
Fig 2. A sample 3D object loaded

E. Vertex and Edge selection

The ray casting technique is used to get the mouse clicked position's object id. Each sphere and edges has an unique object id. When clicked object is found, it's material property is update with a random color. So, whenever user clicks on a position on the canvas, using ray casting we get the object and the color property of the selected object is changed.

F. Save Object

Since the .obj file format doesn't give any edge information, the coloured information from the canvas can't be stored. So, the plan is to work on the .m file format which stores the edge information as well.



G. Integration to GeomSE

All the libraries used in this project have already been used in the main GeomSE[5] site. So, integration won't be an issue.

Upon getting the approval from TA, the project will be integrated to the GeomSE website.

III. Result

Although the vertices and faces can be marked in real time for a given mesh, it has some known issues with the large face mesh file. An example of current result on the 3D sphere is given in Fig 4.

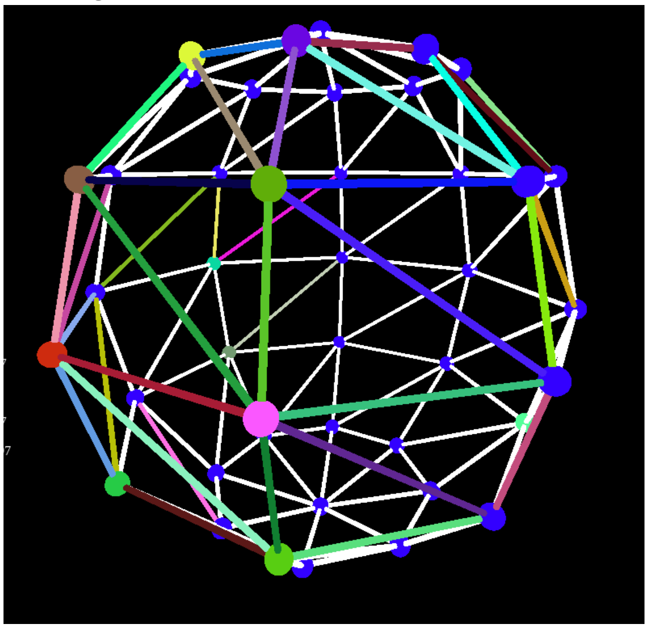


Fig 4. Selected vertices and edges

IV. Future Work

The certain work to be done is to fix the issue on face meshes. Also, the marked information needs to be saved in a separate file.

V. Conclusion

This project was so much helpful to get familiar with the WebGL and other latest libraries like the three.js library. The techniques to provide interactive features on meshes and the structure of the triangular mesh data structure have been well learned.

References

[1] H. Gu, Et. al, “Feature Points Extraction from Faces”, Springer.

[2] Wavefront .obj file, Wikipedia: https://en.wikipedia.org/wiki/Wavefront_.obj_file

[3] Ray casting, Wikipedia: https://en.wikipedia.org/wiki/Ray_casting

[4] three.js r85: https://threejs.org/

[5] GeomSE: http://geom.cs.fiu.edu/#/home