COMP3011 Computer Graphics

Assessment 3

Report Sheet (v6)

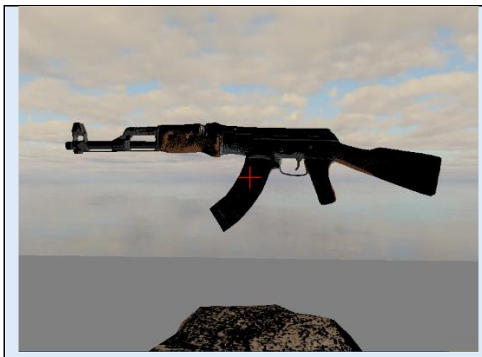
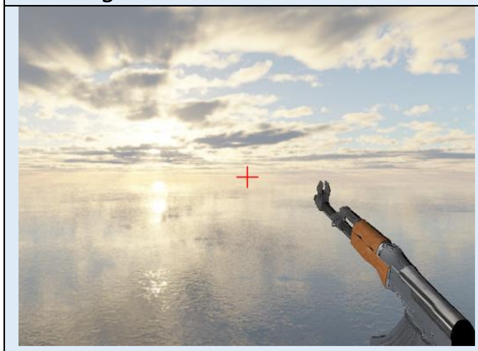
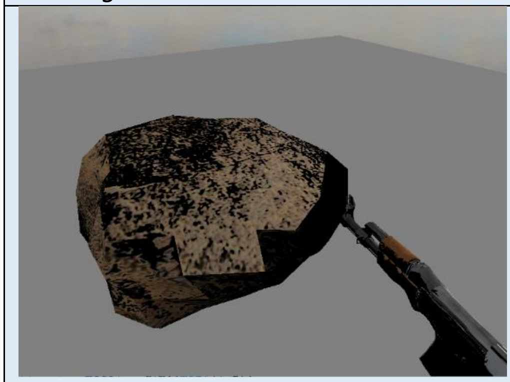
Use this table to help you prepare for your demo. You will also need to submit this report to Moodle. Page limit is 6 pages including all screenshots and text.

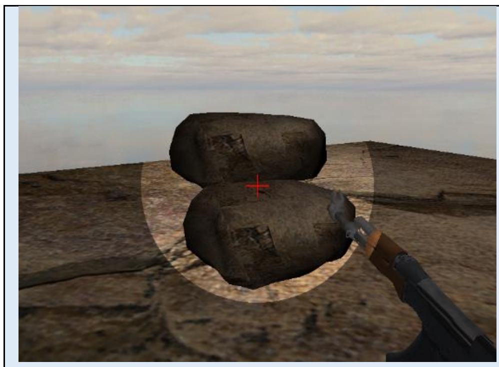
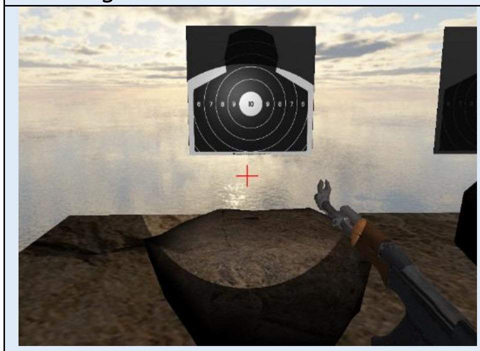
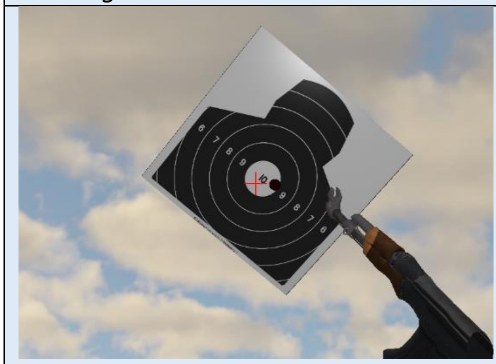
Student Name: Yutian Chen

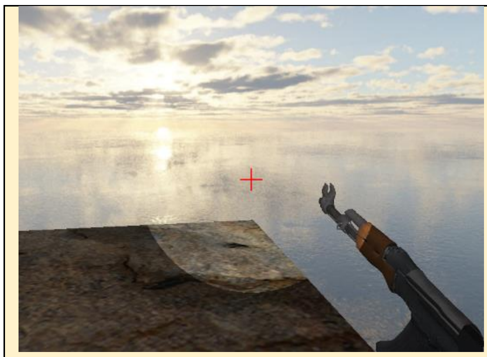
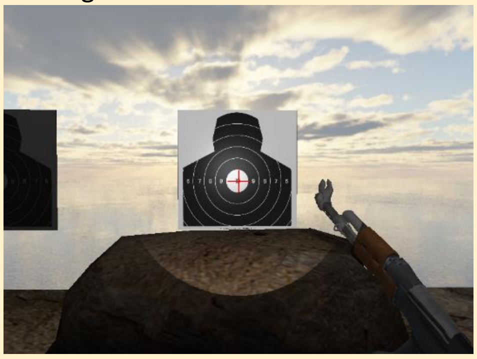
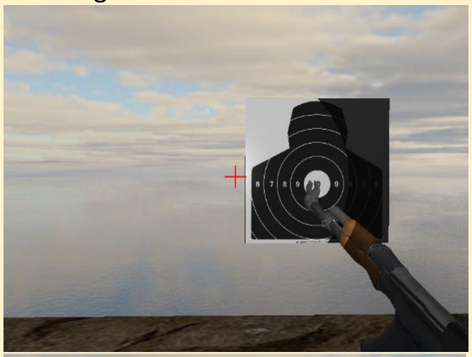
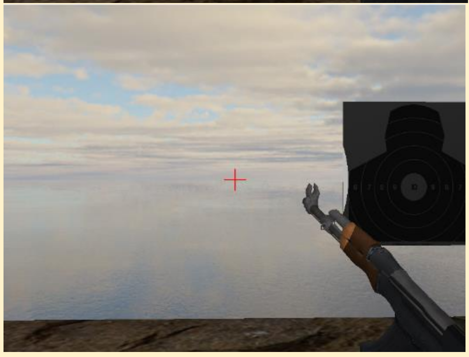
Student ID: 20028267

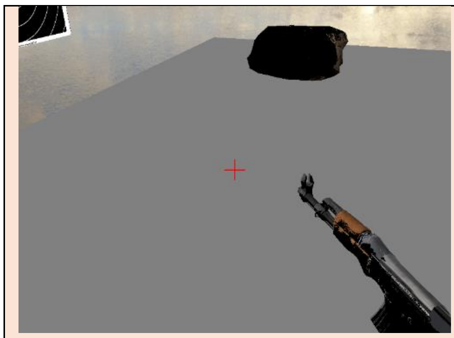
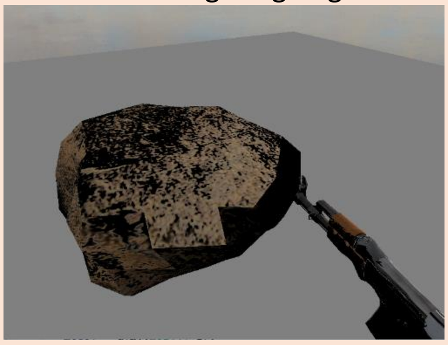
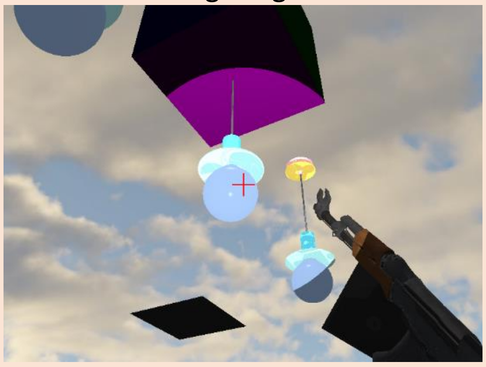
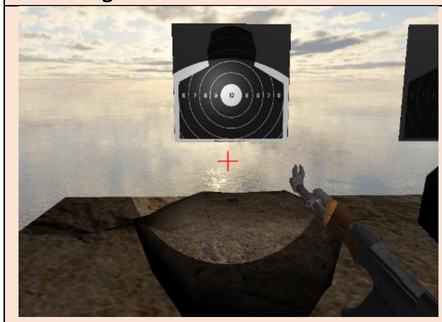
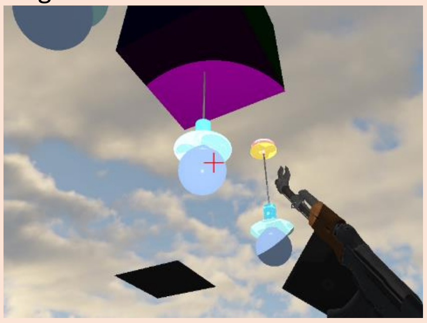
Username: scyyc1

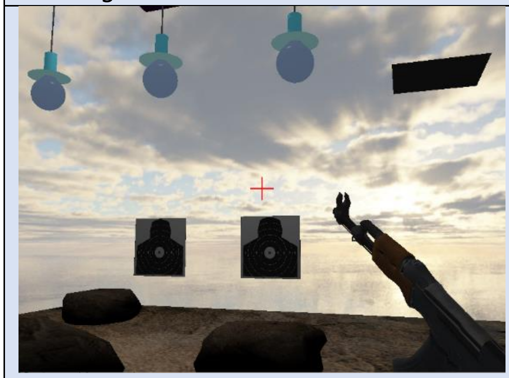
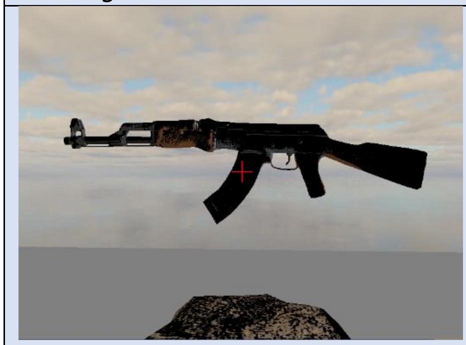
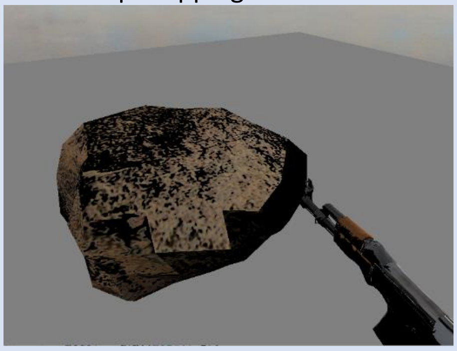
Introduction *Please explain the reason why you chose to implement this scene*	*Describe your inspirations*	*Provide a general description of the scene.*
The scene I implemented is a demo of a shooting games. As a game enthusiast, I have always been curious about technique behind the screen. The key mechanism of shooting game is simple and easy to implement, which makes a good accompaniment to the knowledge currently learned in the CG module.	My inspiration came from the famous shooting game: **Counter Strike: Global Offensive**. I have implemented the simplified key game play logic——shooting with 2D targets and render a scene to mimic a rifle range.	I have implemented the scene with serval targets, and a gun (ak47) loaded from .obj model that always follows the camera. When a player shoots the target using the mouse, the target will generate small circles to represent bullet holes and show where the player has shot. I have also loaded serval models to decorate the environment such as rocks and lamp. I have also implemented bump mapping for light part, which will increase the sense of reality of my scene.
TR 1 – 3D modelling & 3D Transformations *Please give details for up to 3 objects* *Note – if you have programmed an API for parsing OBJ data then provide additional details on the last page*		
Object 1 3D modelling *Please give a screenshot*	*reference specific code (filename and line)*	*Description of object*

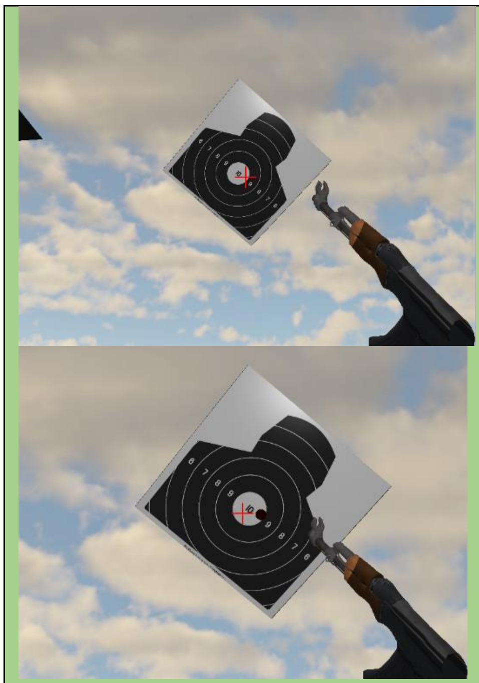
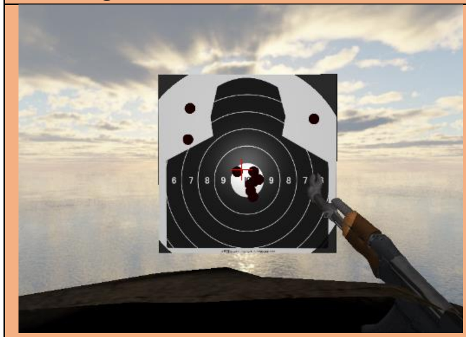
 Picture 1. The loaded weapon model with bump mapping	Model.h mesh.h (will explain in the last page) main.cpp: - Line 253: load the model	I have load an AK47 model to represent the weapon in the shooting game. I spent about a week to program the OBJ parser and set up the lighting corresponding to MTL file.
Object 1 3D Transformations *Please give a screenshot*	*reference specific code (filename and line)*	*Description of transformations*
 Picture 1. The loaded weapon model without bump mapping	main.cpp: - Line 401, 488 – 524: call the “displayGun” function to draw the gun. This function will set the MVP matrix for the weapon and fix it to the camera.	The weapon is fixed to the position in front of the camera by using the camera position, yaw and pitch. I also implement shooting animation. The gun will move slightly backward and rise in a small angle when shooting. It also shakes when player moving in the scene.
Object 2 3D modelling *Please give a screenshot*	*reference specific code (filename and line)*	*Description of object*
 Picture 2. The loaded rock model with bump mapping (eliminate other objects)	Model.h mesh.h (will explain in the last page) main.cpp: - Line 259: load the model	I have loaded a rock model to simulate the battle field of shooting game. The picture one show the
Object 2 3D Transformations *Please give a screenshot*	*reference specific code (filename and line)*	*Description of transformations*

 Picture 2. The loaded rock model without bump mapping	main.cpp: - Line 369 - 399: draw 4 rocks in the scene and pass their MVP matrix	I draw 4 rocks in the scene in different size and position. They all locate at -1 xz plane, which is on the ground.
Object 3 3D modelling *Please give a screenshot*	*reference specific code (filename and line)*	*Description of object*
	plane.h: define a plane target.h: - Line 15 – 24: define the vertices array of a 2D triangle box in xy plane. - Line 55 – 75: "setUp" function set up the buffer array for shading and load the texture. -	The target is an interactive object rendering with texture. The model is a 2D triangle box. The box is located at a plane which is defined in plane.h using a normal vector and a point on the plane ("position" variable). When player shoot the plane, detect the shooting point on the plane and generate bullet holes in black point.
Object 3 3D Transformations *Please give a screenshot*	*reference specific code (filename and line)*	*Description of transformations*
	target.h: - Line 77 – 92: "draw" function will draw the target with model matrix main.cpp: - Line 419 – 439: call the "draw" function to draw the target plane.h: - Line 46 – 55: calculate the model matrix with its position and normal.	The "target" object tracks the normal vector of the plane and use it to calculate the model matrix (transpose inverse of model matrix). I have set 5 targets in the scene. Target 2 is a moving object at the left of the scene. It moves back and forth around the z-axis. The bullet holes on the target will follow the movement of target. Target 5 will not generate bullet holes and the others show that the target work perfectly in any position and direction.
TR 2 - Animation *Please give a screenshot*	*reference specific code (filename and line)*	*Description of animation*

 Picture 1. The gun “shakes” when walking  Picture 2. The gun “recoils” when shooting   Picture 3&4. The moving target	main.cpp: - Line 331 – 333: generate my frequencies of animation with sin function - Line 25 – 26: defined the indicator “isFire” and “isWalking” - Line 335: 0.1s after one shot, close the shooting animate. - Line 506 – 514: add the transformation of animation to the regular model matrix - Line 424 - 433: moving target animation	1. I used 3D transformation with time to create simple animations for the AK47 to represent recoil while shooting and the “shaking” movement (When the player carry the gun and moving, the gun will “shake” in a frequency to show that player is moving). To implement this, I have defined my frequencies. The animation of shooting will finish in 0.1 second and the shaking will loop every second if the camera is walking. These are achieved by using sin function acting to time, which will return a scaler between -1 and 1 according to time. 2. The moving target: I have implemented a moving target, which is at the left of the scene. It moves back and forth around z-axis. The bullet holes on the target will follow the movement of target.
TR 3 – Lighting *Please give details of up to 2 lights*		
Light 1 *Please give a screenshot*	*reference specific code (filename and line)*	*Description of light 1*

 Picture 1. The lighting of gun  Picture 2. The lighting of rock  Picture 3. The lamp model and the light source	shader.frag - Line 23 – 24: define the light source with ambient, diffuse and specular attribute. - Line 118-121, 191: call "calculatePositionalLight" function to calculate the light of lamp Main.cpp - Line 540 – 545: set the lamp (default setting) Mesh.h - Line 132 – 134: set the light for loaded object	The lamp is located at (0, 3, 0) position, where I place a lamp model to represent the **positional light** (picture 3). The light source works both for loaded objects with texture mapping and the modeling objects. I removed other objects in the scene at screenshot 1&2. The picture clearly shows the lighting effect of loaded models (with bump mapping for rock model) The light source is defined in the fragment shader. The shader will distinguish the objects with textures. The attenuation is fixed.
## Light 2 *Please give a screenshot*	*reference specific code (filename and line)*	*Description of light 2*
 Picture 1. The torch illuminates one target and the loaded rock model  Picture 2. The torch illuminates the loaded lamp and a cube	shader.frag - Line 26 – 28: define the light source with ambient, diffuse and specular attribute. - Line 123-140, 194: call "calculateSpotLight" function to calculate the light of torch Main.cpp - Line 547 – 552: set the lamp (default setting) Mesh.h - Line 136 – 138: set the light for loaded object	The second light source is a torch. I implement this by using the **spot light** learned in Lab9. The player can press "SPACE" key to activate it and press "F" key to turn it down. The cut off angle is fixed to be 15 degrees (can be modified at line 556 of main.cpp). Once open, the torch will also follow the camera and illuminate the area in the front. This is achieved by setting light direction and position the same as the camera. The two light sources can be superimposed on each other.

TR 4 – Texture *Please give details of at least 2 textures*		
Texture 1 *Please give a screenshot*	*reference specific code (filename and line)*	*Description of texture*
 Picture 1. Skybox	Texture files: - assets/skybox/right.bmp - assets/skybox/left.bmp - assets/skybox/top.bmp - assets/skybox/bottom.bmp - assets/skybox/back.bmp - assets/skybox/front.bmp - assets/target/target.bmp texture.h - line 69 -103: set up the texture for cube map main.cpp: - line 248 – 256: set up the texture - line 305 – 314: set up the buffer array - line 344, 376 – 488: use function “displaySkybox” to draw it cubemap.vert cubemap.frag	1. I have implemented a skybox for the scene. This is achieved by using cube map of OpenGL. The skybox will always be drawn at the background because I mask the depth buffer when drawing it and make it follow the camera position. The shader for skybox is individual. 2. The shooting targets and the floor are also achieved by using 2D texture mapping.
Texture 2 *Please give a screenshot*	*reference specific code (filename and line)*	*Description of texture*
 Picture 1. The loaded weapon model with bump mapping  Picture 2. The loaded rock model with bump mapping	Texture files: - asserts/gun/ak/.bmp - asserts/rock/.bmp Model.h: - line 607 – 614: recognize and obtain the path of texture file and set it to material of mesh mesh.h: - line 189 – 234: set up the texture according to the mtl file - line 110 – 147: set the mapping texture to correct position in shader. (will explain in the last page)	Texture mapping of loaded object: 1. The rocks are using diffuse mapping and bump mapping. 2. The ak model is using diffuse mapping and specular mapping (I have turn off the bump mapping). 3. Mipmaps are applied. This is achieved by using the build-in function of OpenGL, which means it is generated at runtime (discussed at lab8).
TR 5 - Interactive camera *Please give a screenshot*	*reference specific code (filename and line)*	*Description of interactive camera*

 Picture 1&2. Zooming with mouse wheel	camera.h - Camera class - moveCamera(): line 48 - 69 - orientCamera(): line 71 - 80 - scrollCamera(): line 82 - 89 Main.cpp - line 147 – 227: handle the mouse and keyboard events (Perspective transformation, zooming and moving). - line 235 – 237: register events for the window.	I have implemented a fly-through camera where the y position of camera is fixed to 0. The camera supports the following movement: 1. The player can move with “W, A, S, D” key on the gaming plane (xz plane and -1 for y) 2. The player can use the cursor/mouse to change the perspective angle of camera. 3. The player can use mouse wheel to zoom (the weapon is fixed and will not be influenced by zooming). 4. When player constantly press “SHIFT” button with walking keys, the character will walk slower. This is to simulate the “silent walk” of FPS shooting game.
TR 6 - Interactive object *Please give a screenshot*	*reference specific code (filename and line)*	*Description of interactive object*
 Picture 1. Shooting the target	**circle2D.h:** - **circle2D class** - **Line 38 – 76:** “CreateCircle” function: generate circle with position and normal vector of the plane **target.h** - **Line 94 – 107:** “isHit” function: check if player hit the target - **Line 109 – 120:** “updateBulletHoles “ function: simulate a queue of bullet holes and dynamically update them. **plane.h** - **plane class** (super class) - **Line 34 – 44:** “hitPosition” function: calculate the intersection of a ray and a plane **main.cpp:** - **Line 453 – 465:** In the main loop, check whether the	1. The target: when a player shoots the target. It with automatically generate black circle to represent “bullet holes” at the position where the player hits. One target only keeps 10 latest points and will update automatically. 2. Both targets and bullet holes are defined as a “plane”. The “plane” object is the superclass of “target” object and “circle2D” object (represent bullet holes and the algorithm comes from lab3). The plane object is defined by a normal vector and a point on the plane (the “position” member). The plane also integrates functions to calculate the intersection of a ray with the plane and the model matrix. 3. The boarder checking is achieved by transforming

	targets are hit and update the bullet holes for targets and the timing. **util.h:** - **Line 30:** define the shooting frequency to be 0.1s	the plane back to the origin (the same as the array buffer). This involves calculating model matrix in CPU rather than in shader. The bullet holes has the same normal vector as the target
Conclusion *Please describe what you perceive to be the strengths and weaknesses of your project*	*Describe what aspect of it you are particularly proud of, and what you think would need to be improved.*	*Reflect on what you have learned during this project that you can apply in future projects to improve your performance.*
Strenths **1. Good lighting implementation:** have implement bump mapping for loaded objects. **2. Good mathematics calculations:** the calculation of shooting is achieved by using implicit definition of plane and various mathematics skills. Hopefully, this can demonstrate my understanding of 3D modelling **3. Successfully implement the OBJ parser** **Weaknesses** 1. The artwork does not fit the theme 2. No 3D modelling by hand for complex object. As I position myself to be a graphic programmer, I spent more time on the OBJ parser. Hopefully, this will demonstrate my understanding of 3D modelling.	Proud of: 1. **The bump/normal mapping:** I did some extra learning and successfully implemented bump mapping in this demo which can increase the intricacy of my loaded objects. 2. **The animation of gun shooting:** I have simulated the actions of shooting, silent walking and recoil in this demo. 3. **The OBJ parser:** I have implemented an OBJ parser in this demo. It fulfils me with knowledge of material, which is not discussed in the course. Possible improvements: 1. **Shooting with models:** Implement interaction with "enemy" models. The shooting target in this project is only 2D plane which involves little amount of mathematics calculations (intersection of planes and rays). If interaction with model is introduced, more planes should be considered. This can be achieved by adding an intermediate level of "triangle" to track the shooting position of model. Use the BVH method to build the meshes of loaded models and fine which triangle is hit by player (Calculated by CPU).	1. **Mathematics for 3D modelling:** I have implemented 2D plane by using implicit representation (use a normal vector and a point) in this demo. I also leaned how to use model matrix to transform normal. This can be applied to detect the shooting trajectory of weapons and complex models (with multiple triangle planes, Möller Trumbore Algorithm). 2. **The knowledge of camera:** To make the weapon always follow the camera, I learned how to use Yaw and Pitch to control the object. 3. **Knowledge of lighting:** I have done some extra reading when progressing the coursework, including bump mapping and ray tracing.

	2. **Materials:** The demo only implement the basic part of material, which is at variance with the complex lighting in real life. Neither the demo obey the conservation of energy and nor the refraction is considered in this case (“Ni” variable in mtl file). This can be improved by introducing BRDF. 3. **The artwork:** The models loaded in this demo are not decent for building a shooting game. The shooting animation also not well designed.	

TR 1 – 3D modelling & 3D Transformations

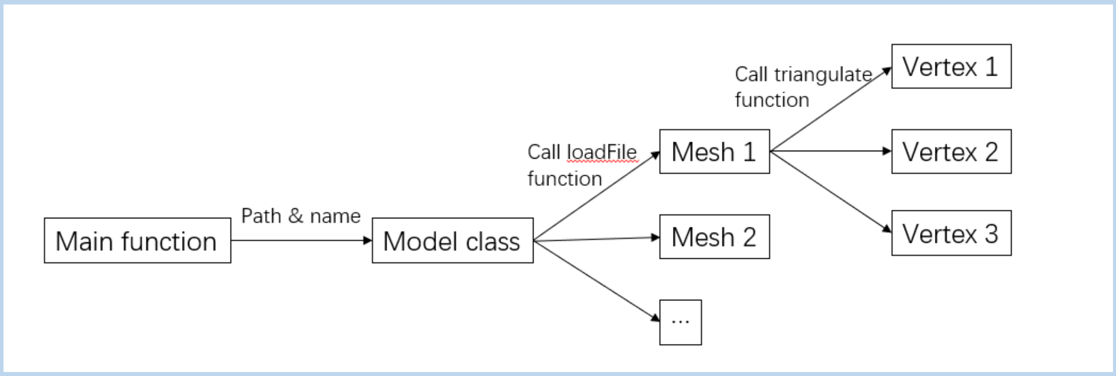
OBJ parser
Please give details for how you programmed the OBJ parser function
Include all of the important aspects including

1. *Memory allocation and freeing*
2. *Data structures you programmed (remember you can only use API which I provided in the course)*
3. *How you read the file*
4. *Logic for parsing the vertex attributes*
5. *Transferring the vertex attributes to OpenGL*

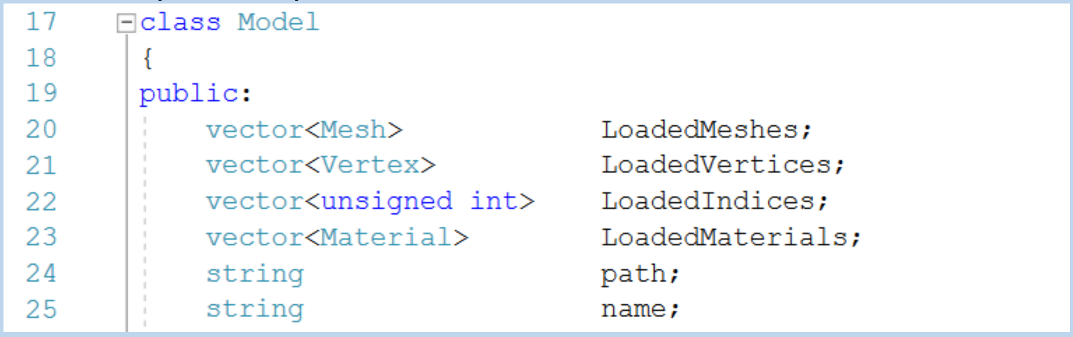
1 Memory allocation and freeing

The representation of model is using the class concept of C++. Each model is loaded as a model instance. The main function keeps all the object instances, so the memory of models is only stack-allocated. The memory of model will be recycled when the program ends.

2 Data structures

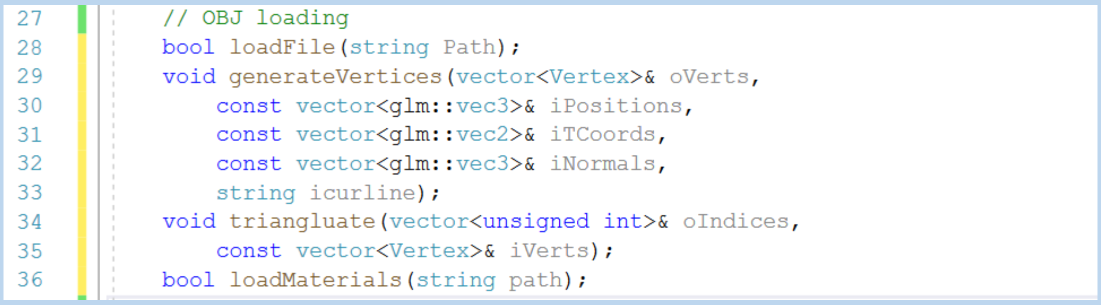


a. **Model class (in model.h)**:



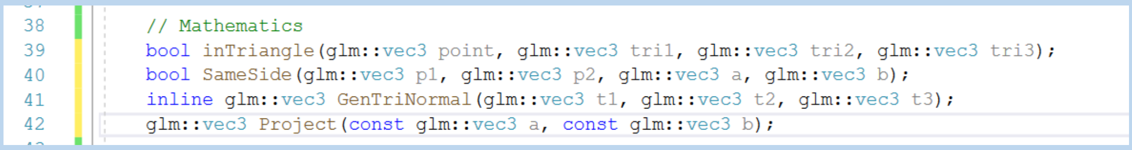
```
class Model
{
public:
    vector<Mesh>            LoadedMeshes;
    vector<Vertex>          LoadedVertices;
    vector<unsigned int>    LoadedIndices;
    vector<Material>        LoadedMaterials;
    string                  path;
    string                  name;
```

LoadMeshes stores all the meshes from the OBJ file.
LoadVertices records all the vertices.
LoadIndices records the vertices index for building EBO
loadedMaterials stores material library from .mtl file



```
    // OBJ loading
    bool loadFile(string Path);
    void generateVertices(vector<Vertex>& oVerts,
        const vector<glm::vec3>& iPositions,
        const vector<glm::vec2>& iTCoords,
        const vector<glm::vec3>& iNormals,
        string icurline);
    void triangluate(vector<unsigned int>& oIndices,
        const vector<Vertex>& iVerts);
    bool loadMaterials(string path);
```

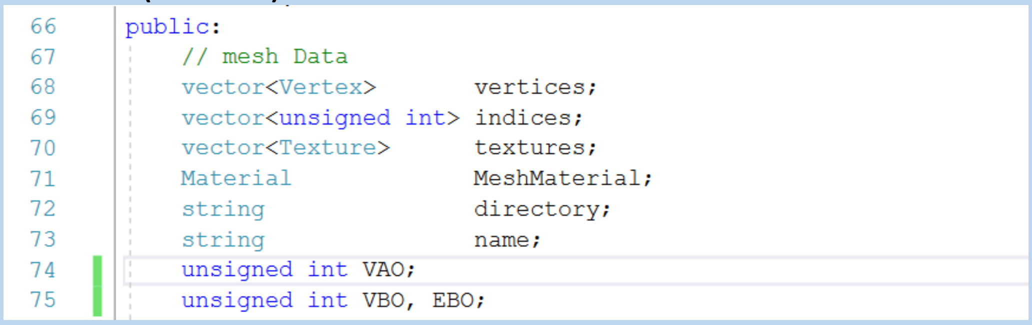
loadFile (line 48) load the .obj file with path. Will be activate when using the constructor.
generateVertices (line 263) when reading "f" line, generate vertex with the index provided with face information
triangulate (line 348) take the first 3 vertices of a face in "f" line to build a triangle. I did not implement triangle class. The main purpose of this function is to build ***LoadIndices*** array for EBO and calculate the normal of the face if the normal is not provided.
loadMaterials (line 507) will be called when read the line "mtllib". Will perform similar actions as the ***loadFile*** function to read the .mtl file that it specify.



```cpp
// Mathematics
bool inTriangle(glm::vec3 point, glm::vec3 tri1, glm::vec3 tri2, glm::vec3 tri3);
bool SameSide(glm::vec3 p1, glm::vec3 p2, glm::vec3 a, glm::vec3 b);
inline glm::vec3 GenTriNormal(glm::vec3 t1, glm::vec3 t2, glm::vec3 t3);
glm::vec3 Project(const glm::vec3 a, const glm::vec3 b);
```

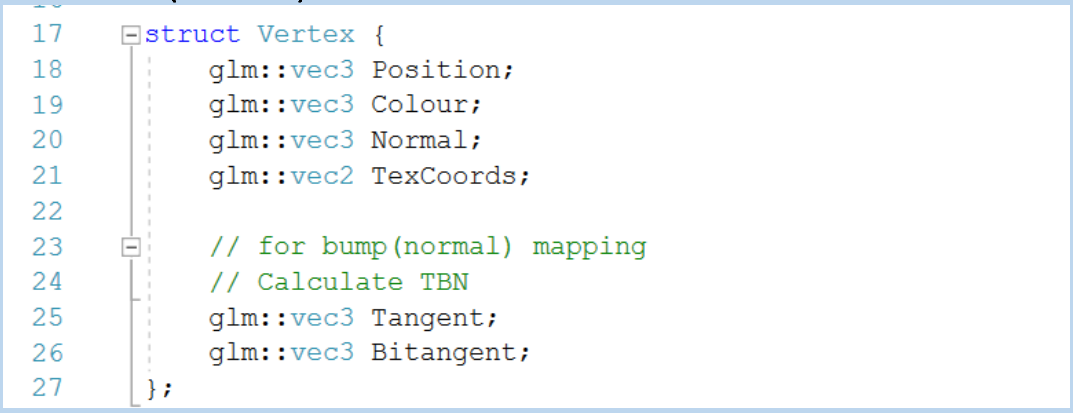
Functions to be use when triangulating faces.

b. **Mesh class (in mesh.h)**:



```cpp
public:
    // mesh Data
    vector<Vertex>       vertices;
    vector<unsigned int> indices;
    vector<Texture>      textures;
    Material             MeshMaterial;
    string               directory;
    string               name;
    unsigned int VAO;
    unsigned int VBO, EBO;
```

c. **Vertex struct (in mesh.h):**

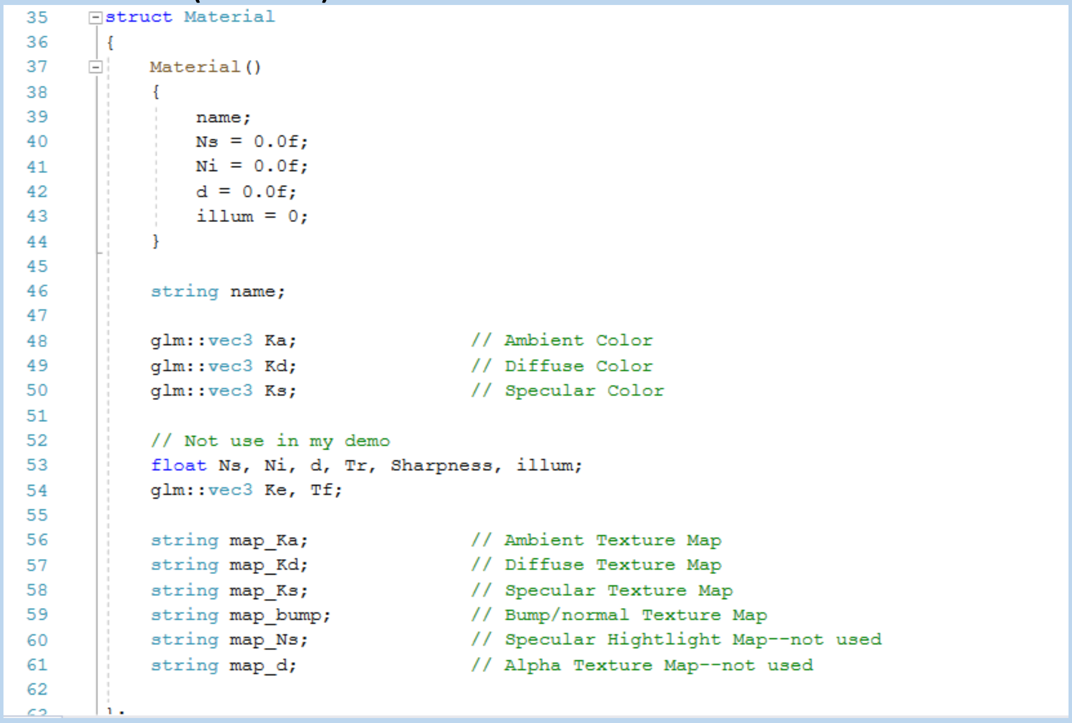


```cpp
struct Vertex {
    glm::vec3 Position;
    glm::vec3 Colour;
    glm::vec3 Normal;
    glm::vec2 TexCoords;

    // for bump(normal) mapping
    // Calculate TBN
    glm::vec3 Tangent;
    glm::vec3 Bitangent;
};
```

store the position, normal, colour, texture coordinate and tangent of a vertex. Positions and texture coordinate are necessary. Tangent and normal can be calculated once set up.

d. **Material struct (in mesh.h)**:



```cpp
struct Material
{
    Material()
    {
        name;
        Ns = 0.0f;
        Ni = 0.0f;
        d = 0.0f;
        illum = 0;
    }

    string name;

    glm::vec3 Ka;              // Ambient Color
    glm::vec3 Kd;              // Diffuse Color
    glm::vec3 Ks;              // Specular Color

    // Not use in my demo
    float Ns, Ni, d, Tr, Sharpness, illum;
    glm::vec3 Ke, Tf;

    string map_Ka;             // Ambient Texture Map
    string map_Kd;             // Diffuse Texture Map
    string map_Ks;             // Specular Texture Map
    string map_bump;           // Bump/normal Texture Map
    string map_Ns;             // Specular Hightlight Map--not used
    string map_d;              // Alpha Texture Map--not used

```

store the lighting information defined in MTL file

3 General process

a. The main function instantialize the model with path and name. The constructor of model class will call loadFile function.

b. Detect the first word of a line (not space or tab), e.g. “v” for vertex, “vn” for normal.

c. Split the line with space or "/", obtain each component and store it in corresponding temporary buffers
d. If the first token is "mtllib", it will than read the .mtl file and call the **loadMaterial** function. Perform similar action for .mtl file as loadFile.
e. If the first token is "f", it will locate the vertex previously store in buffer with the index and build the triangle with vertex positions and normal by calling **triangulate** function
f. If the first token is "o", it will push the current mesh to the **LoadedMesh** buffer and clean up all the temporary buffer.

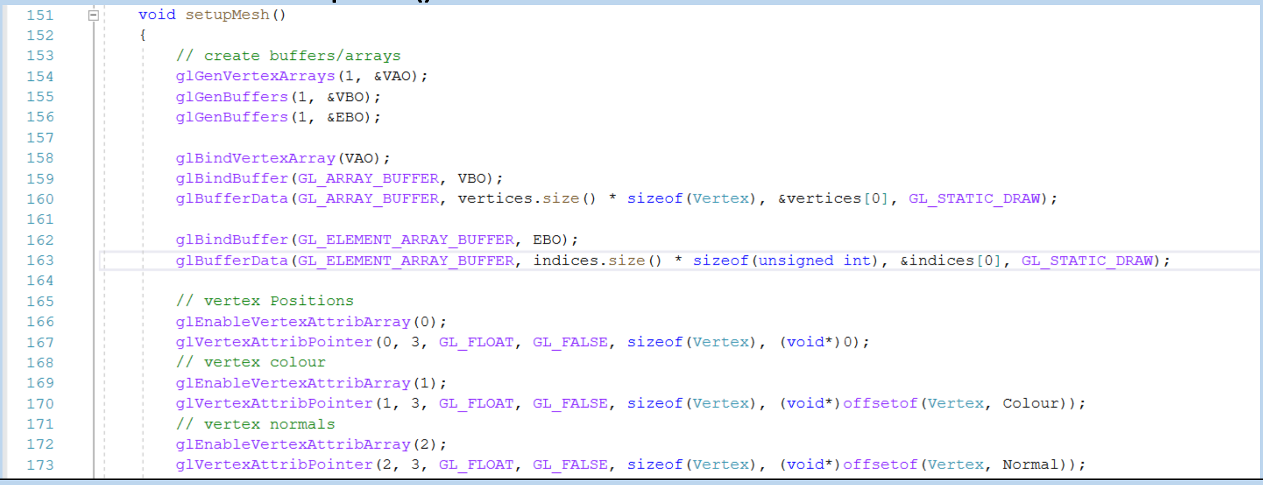
4 Logic for parsing the vertex attributes

In .obj file, the attribute of vertices will appear earlier than face element. When calling **loadFile** function in one mesh, I store all the "v", "vt" and "vn" in temporary buffers first, which is local variable "Positions", "TCoords" and "Normals". When it comes to "f" line, obtain the indices of vertices for building a triangle. Then call the **triangulate** function to and stores these vertices to "LoadedVertices". As the "f" line may contain more than 3 vertices. I only take the first 3 vertices for modelling. As a result, the function of my OBJ parser is defective. It doesn't work perfectly for meshes that contains face with vertex more than 3. If read a "o" line, it will build the mesh with all the temporary buffers and push it to the "LoadedMesh". Then clean up all temporary buffer and start to read the next mesh.

5 Building buffer array with index

The buffer array is building by using VAO, VBO and EBO, which means using an array of vertices and an array of index to indicate which vertex is used. This means that the vertices in buffer array will not repeat. The core idea is to use the indices buffer to track the vertices that used for shading. The OpenGL has the GL_ELEMENT_ARRAY_BUFFER to support this implementation. Here I build the EBO buffer with the indices obtained from OBJ file at "f" lines. Because the C use a continuous field to store the struct instance, the size of each element can be obtained by sizeof(Vertex).

The code is in **line 151 setUpMesh()** function in **mesh.h** file



```
151    void setupMesh()
152    {
153        // create buffers/arrays
154        glGenVertexArrays(1, &VAO);
155        glGenBuffers(1, &VBO);
156        glGenBuffers(1, &EBO);
157
158        glBindVertexArray(VAO);
159        glBindBuffer(GL_ARRAY_BUFFER, VBO);
160        glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(Vertex), &vertices[0], GL_STATIC_DRAW);
161
162        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
163        glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(unsigned int), &indices[0], GL_STATIC_DRAW);
164
165        // vertex Positions
166        glEnableVertexAttribArray(0);
167        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)0);
168        // vertex colour
169        glEnableVertexAttribArray(1);
170        glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Colour));
171        // vertex normals
172        glEnableVertexAttribArray(2);
173        glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Normal));
```