## Instructions

1. Double-click OpenGL\_16\_17.pro and open project with QTcreator.
2. Access **mainview.h** and look for the variable ***setting***
3. There will be 2 options available for this setting
   1. ***setting*** = 0;//Cube
   2. ***setting*** = 1;//Sphere
4. **Build** and **run** to see results
   1. If you were to encounter any issues that prevent you from seeing the correct result, please **clean** and **build** before **running** the project again
5. Renderer and Model Dropdown-list isn’t working

## Modifications and Changes

1. **mainview.h**
   1. Variables(public)
      1. QVector3D *rotationXYZ* // declared to store the values of xyz used for rotating the model
      2. float *scaleValue* // used for altering the scale of the model by manipulating the value during user input (mouse scroll, slider)
      3. float *startX* // captured to calculate the offset during mouse movement used for rotation
      4. float *startY* // captured to calculate the offset during mouse movement used for rotation
   2. Variables(private)
      1. GLuint *VAO* // pointer of Array in GPU use for storing buffer object
      2. GLuint *colorBO* // pointer in GPU used for storing color buffer object
      3. GLuint *normalBO* // pointer in GPU used for storing normal buffer object
      4. GLint *modelPtr* // pointer for uniform variable **model**. Used for pushing a transformed model into the shader
      5. GLint *viewPtr* // pointer for uniform variable **view**. Used for a transformed view into the shader
      6. GLint *projPtr* // pointer for uniform variable **proj**. Used for pushing transformed proj into the shader
      7. GLint *setting* // variable used for changing the **settings** between cube and sphere
      8. GLint *normalPtr* // pointer for uniform variable **normal**. Used for pushing transformed normal into the shader
      9. GLint *materialColorPtr* // pointer for uniform variable **materialColor**. Used for pushing color of the sphere into the shader
      10. GLint *intensityPtr* // pointer for uniform variable **intensity**. Used for pushing intensity of sphere into the shader
      11. GLint *lightPosPtr* // pointer for uniform variable **lightPos**. Used for pushing transformed position of light into the shader
      12. GLint *positionPtr* // pointer for uniform variable **position**. Used for pushing position of sphere data into the shader
      13. GLint *settingPtr* // pointer for uniform variable **setting**. Used for pushing value of setting into the shader
      14. GLint *camPtr* // pointer for uniform variable **cam**. Used for pushing camera position into the shader
      15. QMatrix3x3 *normal* // variable used for handling transformation of **normal**
      16. QMatrix4x4 *model* // variable used for handling transformation of **model**
      17. QMatrix4x4 *view* // variable used for handling transformation of **view**
      18. QMatrix4x4 *proj* // variable used for handling transformation of **proj**
2. **mainview.cpp**
   1. createShaderPrograms()
      1. in the createShaderPrograms(), we had declared many variables to store the pointer of each uniform variable. This is to ensure that if we have any data to push to the shader, with the pointer, we can do it easily via glUniform
   2. createBuffers()
      1. Generally, codes in this method are to generate buffers and binding them into the vertex array. The sequence of glGenBuffer, glBindBuffer, enableVertexAttribArray and glVertexAttribPointer is very important.
   3. loadModel()
      1. In this method, we have to prepare data to push over to the GPU via BufferObject Pointer. We have to ensure that 3 vertices have the same rgb value otherwise there would be color gradient. Ensuring the bufferdata is correct is a critical because by declaring insufficient size, the bufferdata is unable to store all the data.
   4. resizeGL()
      1. This method give us the ability to rescale our obj model properly. Without the new width and height to rescale, our obj will deform
   5. paintGL()
      1. This method generally prepares the transformation of the view, projection and model. All the rotation and scaling are done here as each time the update() function is called, everything in paintGL will be triggered.
3. **raytracerscene.cpp**
   1. renderSphere
      1. This method generally prepares the transformation of the view, projection and model. All the rotation and scaling are done here as each time the update() function is called, everything in paintGL will be triggered. In case of sphere, renderSphere will be triggered along with paintGL

## Challenges

1. Initially, due to neatness, we rearranged the sequence of generating/binding/enabling of buffer. Little do we know that by rearranging the sequence, we are not binding it correctly into the VAO. Thus we are stuck at the part where we suppose to achieve the white screen for a few hours.
2. Due to the competency in c++ we faced a little problem in figuring out the functions we were provided by the library.
3. During debugging, any issues encountered in glsl are not recognized as compiling issue. Therefore we have to spend time figuring out the syntax and logic in the glsl.

## Without the transformed normal

Without the transformed normal, specular appears at the back of the sphere.