Computer Graphics comp3811 CW2

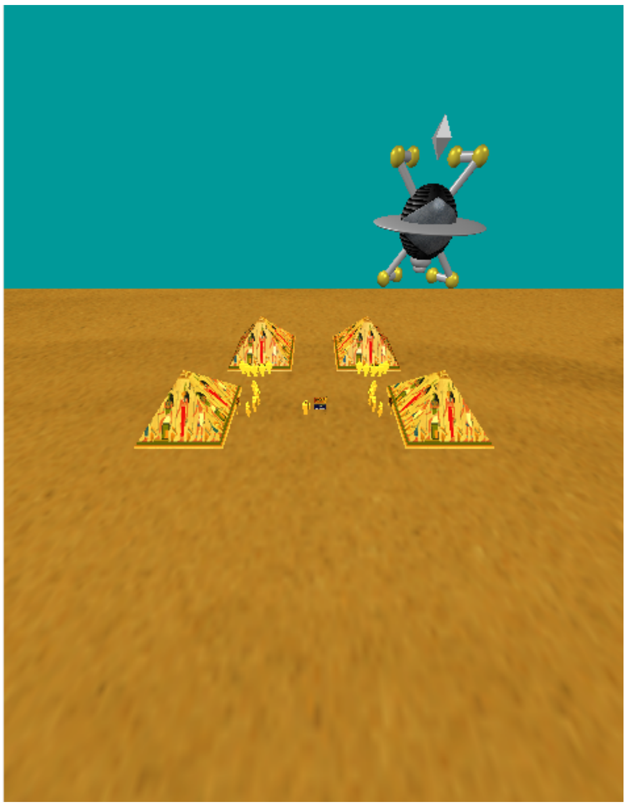
Stefanos Costa

sc18sc

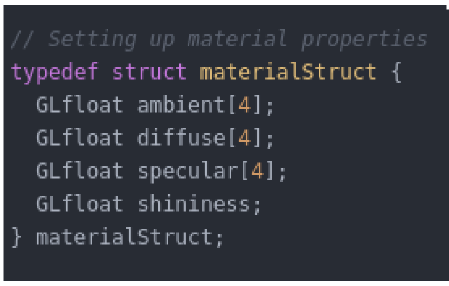
Introduction to the scene:

This project draws a scene staged in Egypt, where an alien circles around the centre of the scene and itself in the air. At the centre of the scene, a box man representing Marc is standing next to a billboard. The alien has the ability to abduct the man at the centre during day or night, and during the day there are other box men visiting the Museum in Egypt. The man at the centre is surrounded by pyramids and boxmen that are looking at billboards.

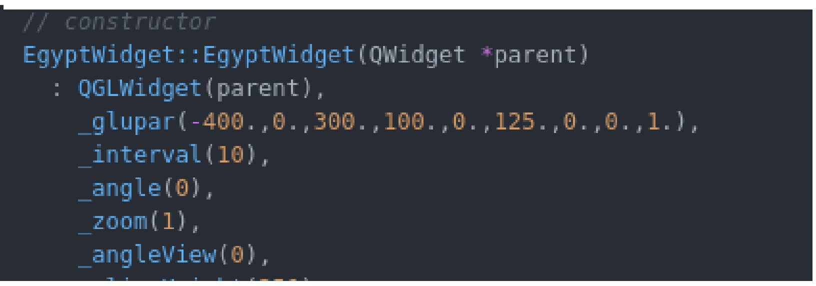
Setup:



The scene is drawn based on a QWidget called EgyptWidget whose class is declared in the header file EgyptWidget.h, and its functions and constructor are located in EgyptWidget.cpp. The basis for this widget is the orthographic tutorial provided in this module. The Struct that defines the coordinates of GluLookat that set the camera in OpenGL _glupar, as well as the materialStruct struct type containing the properties for ambient, diffuse , specular and shininess is maintained in the widget. Using the materialStruct type, the material properties used in this scene are Emerald,Polished Silver,Gold,Polished Gold, Obsidian, Pearl, Ruby and Brass Materials and they are declared in EgyptWindow.cpp.



```
// Setting up material properties
typedef struct materialStruct {
  GLfloat ambient[4];
  GLfloat diffuse[4];
  GLfloat specular[4];
  GLfloat shininess;
} materialStruct;
```



```
// constructor
EgyptWidget::EgyptWidget(QWidget *parent)
  : QGLWidget(parent),
    _glupar(-400.,0.,300.,100.,0.,125.,0.,0.,1.),
    _interval(10),
    _angle(0),
    _zoom(1),
    _angleView(0),
```

When the widget is created, the constructor initialises the variables required to run it. This includes the camera Struct.

The widget initialises the variable interval, which is the refresh rate in milliseconds, to 10, since the timer used to trigger repainting of the scene times out every 10 ms. This timer is located in EgyptWindow.cpp along with all the user interface components and is connected to the updateAngle() slot of the widget.

The constructor also sets the _angle variable to 0. This variable represents the current number of degrees of rotation. The updateAngle() slot increments the angle degrees by 1 every time the timer times out and repaints the scene setting the prerequisites for rotating animations.

The _zoom variable is also initialised to 1. This is used to control the field of view parameter of the gluPerspective() function, bringing the scene closer by subtracting the zoom value from the value of the field of view in the y direction. The zoom value is controlled using the zoomSlider in EgyptWindow.cpp that is connected to a slot called updateZoom() where the value of _zoom is set to be the value of the slider.

The _angleView variable represents the angle around the z-axis at which the user will view the scene and it is set to 0. This variable is controlled using the angleViewSlider whose values range from 0 to 360. It is connected to the AngleView() slot that sets the _angleView value to the value of the slider.

The _day Boolean variable represents whether it is day or not. It is controlled through the lightbtn and darkbtn that set its values to true or false. These buttons are connected to the corresponding slots that do this.

Then, the constructor initialises the variables used to manipulate the properties of an alien being:

_alienHeight – Used to set the height at which the alien is flying. Initialised to 250.

_alienRadius – Used to set the radius of the alien's core.

_alienRotationRadius – Used to set the radius at which the alien is rotating around the centre of the scene. Initialised to 150.

_alienRotationSpeed – Used to set the alien's speed of rotation around the origin in degrees per ms. Initialised to 2.

_alienSpinSpeed – Used to set the alien's speed of rotation around itself in degrees per ms. Initialised to 2.

_limbAngle – Used to set the angle at which the alien's limbs are rotated on the y axis

_gate – materialStruct that contains the material properties of the alien's gate. Initialised to polished Silver.

_armJoint – materialStruct that contains the material properties of the alien's cylinder joints. Initialised to polished Gold

_armJointSphere - materialStruct that contains the material properties of the alien's sphere joints. Initialised to polished Silver.

_alien – materialStruct that contains the material properties of the alien's core. Initialised to polished Silver.

_disc - materialStruct that contains the material properties of the alien's gate. Initialised to polished Silver.

The abduction animation is based on signal variables of Boolean type that are all initialised to false by the constructor, along with double variables representing translations initialised to 0.

abduct – signals that abduction is occurring

_asc -signals that the first alien ascent is occurring

_desc -signals that the first alien descent is occurring

_descend – holds the current value of height that alien is descending during first descent

_ascend – holds the current value of height that alien ascending during first ascent

_return – signals the return of the alien to start second ascent

_desc2 – signals the second descent of the alien

_marcAsc – signals the ascent of Marc boxman during abduction

_marcAscend – holds the current value of height that marc has ascended during abduction

_descend2 – holds the current value of height that alien is descending during second descent

_ascend2 – holds the current value of height that alien is ascending during second ascent

_exit – signals the alien's exiting animation to start

_leave – holds the current value of height (z-axis) and distance from the origin on the x axis that alien has ascended through while exiting

Loading images as textures is taken care of using the image class form the image.cpp and image.h files that have been taken from the banksy tutorial on texturing.

The constructor loads and stores each texture in Glubytes form in 6 image objects:

_image - Marc_dekamps.ppm

_image2 – Mercator-projection.ppm

_image3 – sand.jpg

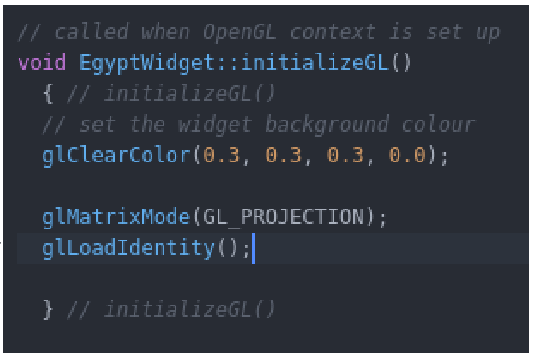
_image4 – hieroglyphics.jpg

_image5 – ship.jpg

_image6 -museum.jpg

<u>InitiallizeGL():</u>

The egyptWidget's initializeGL function sets the default color of the background to grey, changes its mode to the projection matrix sets it to the identity matrix.

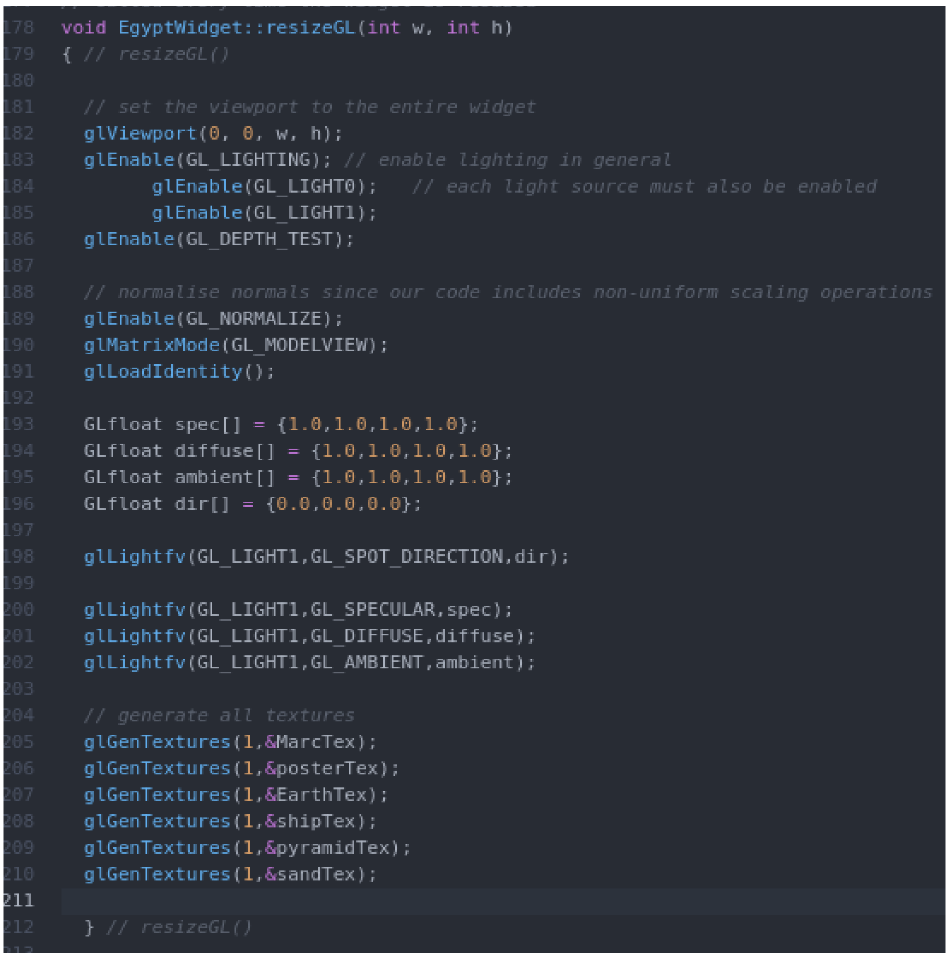


```
// called when OpenGL context is set up
void EgyptWidget::initializeGL()
  { // initializeGL()
  // set the widget background colour
  glClearColor(0.3, 0.3, 0.3, 0.0);

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();

  } // initializeGL()
```

Every time the widget is resized, <u>the resizeGL() function is called:</u>

It sets the viewport to be the size of the entire widget so that it fills the widget and can be resized. It then enables lighting along with two lights: GL_LIGHT0, and GL_LIGHT1. The default color of GL_LIGHT0 is white, it comes from all angles, simulating the sun, and its position is at infinity, while the default colour of GL_LIGHT1 is black.

It then enables the GL_DEPTH_TEST that draws objects depending on which one is closer.



```
void EgyptWidget::resizeGL(int w, int h)
{ // resizeGL()

  // set the viewport to the entire widget
  glViewport(0, 0, w, h);
  glEnable(GL_LIGHTING); // enable lighting in general
        glEnable(GL_LIGHT0);   // each light source must also be enabled
        glEnable(GL_LIGHT1);
  glEnable(GL_DEPTH_TEST);

  // normalise normals since our code includes non-uniform scaling operations
  glEnable(GL_NORMALIZE);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

  GLfloat spec[] = {1.0,1.0,1.0,1.0};
  GLfloat diffuse[] = {1.0,1.0,1.0,1.0};
  GLfloat ambient[] = {1.0,1.0,1.0,1.0};
  GLfloat dir[] = {0.0,0.0,0.0};

  glLightfv(GL_LIGHT1,GL_SPOT_DIRECTION,dir);

  glLightfv(GL_LIGHT1,GL_SPECULAR,spec);
  glLightfv(GL_LIGHT1,GL_DIFFUSE,diffuse);
  glLightfv(GL_LIGHT1,GL_AMBIENT,ambient);

  // generate all textures
  glGenTextures(1,&MarcTex);
  glGenTextures(1,&posterTex);
  glGenTextures(1,&EarthTex);
  glGenTextures(1,&shipTex);
  glGenTextures(1,&pyramidTex);
  glGenTextures(1,&sandTex);

  } // resizeGL()
```

Since the code of this project involves non-uniform scaling and this will affect the normals of the glut/glu objects, the normals to the surfaces of the objects need to be normalised. This is achieved by enabling GL_NORMALISE that normalizes all the normals of the objects. This is can be taxing in terms of performance, however it helps with the overall design of the project being that objects are created at unit length and then any size transformations are made using scaling. It is important for normals to be calculated correctly and normalised so that lighting effects can function properly. It then loads the model/view matrix and replaces it with the identity matrix so that the scene is reset and ready to be placed into view space of the camera.

It then sets the diffuse, specular and ambient properties of GL_LIGHT1 to white and sets the light’s direction to the origin.

Finally, it generates the texture names for each texture to be used in the scene and stores them in their corresponding GLUint variables:

Drawing the scene:

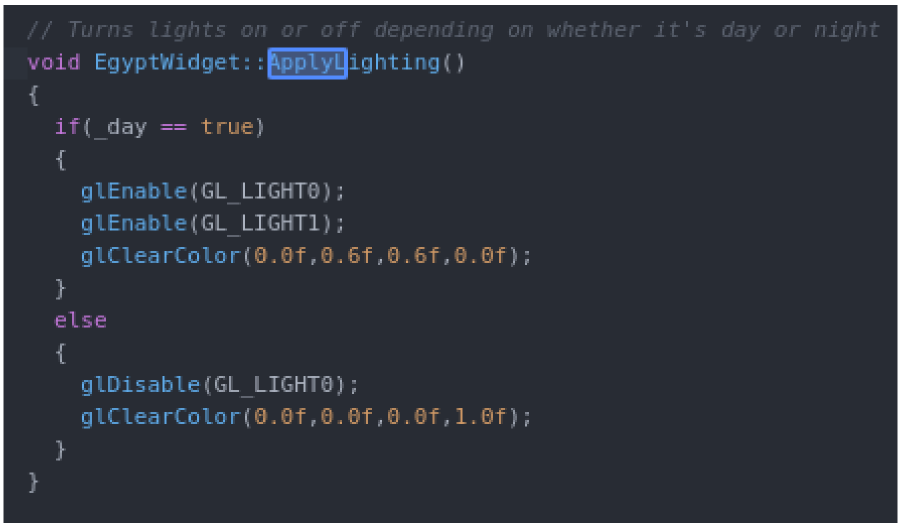
The scene is drawn in paintGL():

Every time paintGL() is called, it clears the colour buffer and depth buffers, loads the projection matrix and sets it to the identity matrix, essentially resetting it.

It then calls gluPerspective that sets a perspective projection matrix where the field of view in the y direction is 100 -_zoom, the aspect ratio is 4:3 , the near plane is 100 and the far plane is 1000. The far plane is set to 1000 to allow the camera to see beyond the scene's main objects, giving the effect of a horizon.
After the perspective is set, the model/view matrix is loaded, the depth test is enabled and the model/view matrix is set to the identity matrix

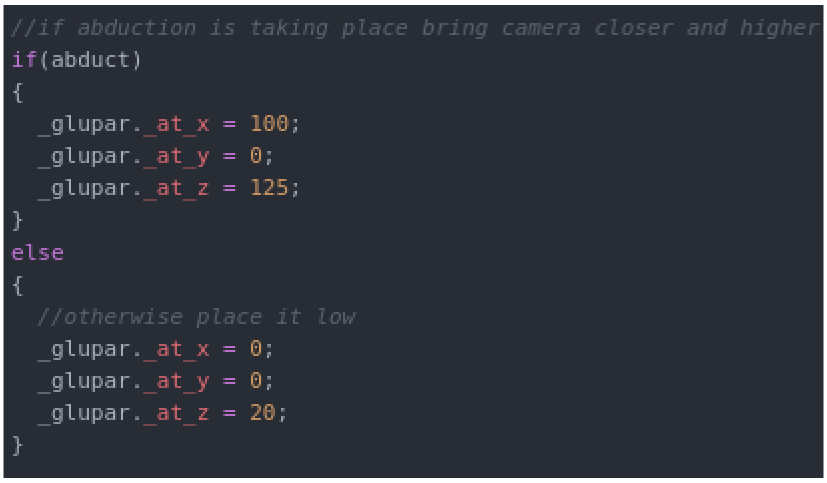
Then, the ApplyLighting() method of the widget is called to apply the lighting conditions of the scene:

If it is day, then both GL_LIGHT0 and GL_LIGHT1 will be enabled, and the background colour will be set to turqiose ,otherwise the GL_LIGHT0 is disabled and the background is set to black.



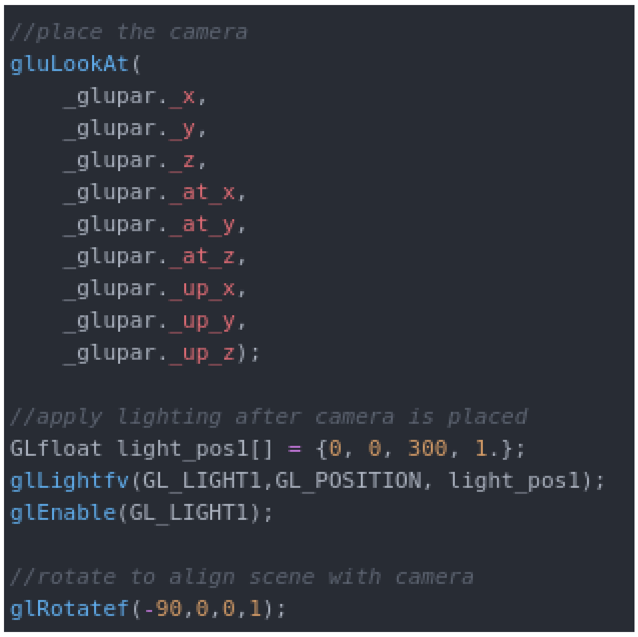
```
// Turns lights on or off depending on whether it's day or night
void EgyptWidget::ApplyLighting()
{
  if(_day == true)
  {
    glEnable(GL_LIGHT0);
    glEnable(GL_LIGHT1);
    glClearColor(0.0f,0.6f,0.6f,0.0f);
  }
  else
  {
    glDisable(GL_LIGHT0);
    glClearColor(0.0f,0.0f,0.0f,1.0f);
  }
}
```

If an abduction is occurring, then the camera is set to look at (100,0,125) which is at half the height of the abduction, and 100 behind Marc to allow the scene to have depth, otherwise it is set to look at (0,0,20) which is the middle between Marc and the billboard.



```
//if abduction is taking place bring camera closer and higher
if(abduct)
{
  _glupar._at_x = 100;
  _glupar._at_y = 0;
  _glupar._at_z = 125;
}
else
{
  //otherwise place it low
  _glupar._at_x = 0;
  _glupar._at_y = 0;
  _glupar._at_z = 20;
}
```

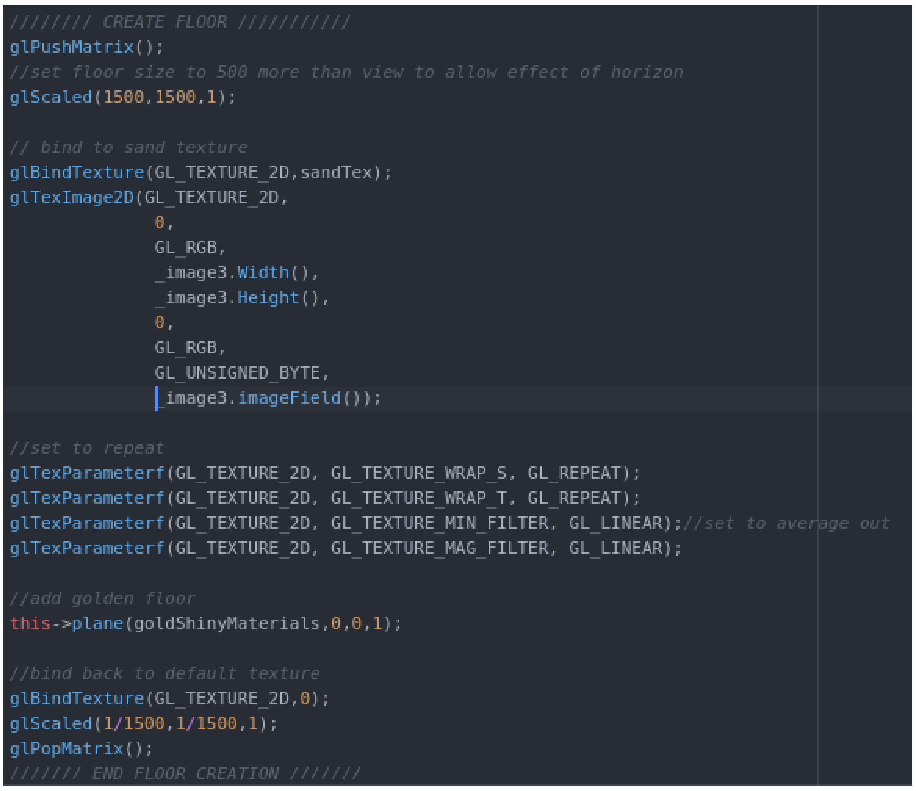
The camera's position, view direction and up direction is then set using gluLookat(). This way, the z-axis is the up direction, the x-axis is the forwards/backwards direction and the y axis is the sideways direction. Then, GL_LIGHT1 is placed at (0,0,300) above the origin and re-enabled. Due to personal preferences, the scene is rotated anticlockwise around the z-axis such that the z-axis is still the up direction, but the y axis is the forwards/backwards direction and the x-axis is the sideways direction. This also adheres to the convention of maintaining a right-handed system in OpenGL:



```
//place the camera
gluLookAt(
    _glupar._x,
    _glupar._y,
    _glupar._z,
    _glupar._at_x,
    _glupar._at_y,
    _glupar._at_z,
    _glupar._up_x,
    _glupar._up_y,
    _glupar._up_z);

//apply lighting after camera is placed
GLfloat light_pos1[] = {0, 0, 300, 1.};
glLightfv(GL_LIGHT1,GL_POSITION, light_pos1);
glEnable(GL_LIGHT1);

//rotate to align scene with camera
glRotatef(-90,0,0,1);
```

Any further rotations around the z-axis are then applied to the scene depending on the user's input from the slider through the _angleView variable.

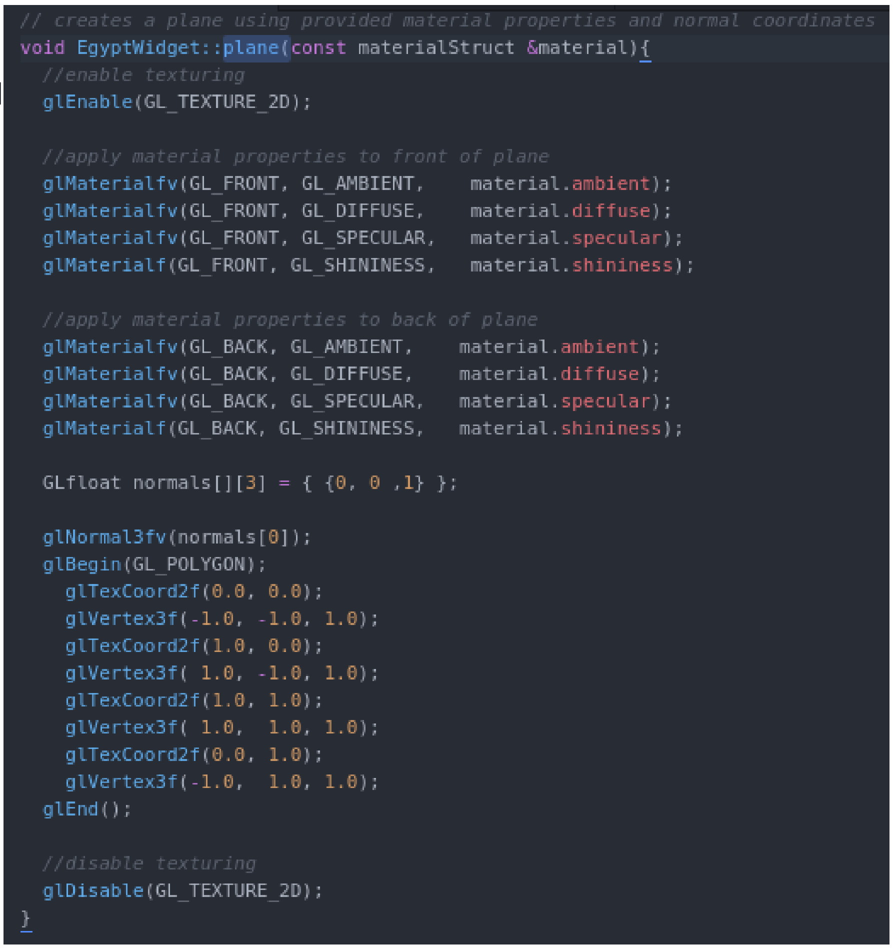
 Once the scene settings are set, the floor is created:



```
//////// CREATE FLOOR //////////
glPushMatrix();
//set floor size to 500 more than view to allow effect of horizon
glScaled(1500,1500,1);

// bind to sand texture
glBindTexture(GL_TEXTURE_2D,sandTex);
glTexImage2D(GL_TEXTURE_2D,
             0,
             GL_RGB,
             _image3.Width(),
             _image3.Height(),
             0,
             GL_RGB,
             GL_UNSIGNED_BYTE,
             _image3.imageField());

//set to repeat
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);//set to average out
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

//add golden floor
this->plane(goldShinyMaterials,0,0,1);

//bind back to default texture
glBindTexture(GL_TEXTURE_2D,0);
glScaled(1/1500,1/1500,1);
glPopMatrix();
/////// END FLOOR CREATION //////
```

The current matrix is scaled such that the x and y components, meaning the length and width of the plane are multiplied by 1500, while its z component, meaning its height remains the same. Then, it binds the sand.jpg texture buffer to the 2d texture GL_TEXTURE_2D and defines the texture image using glTexImage2d(). The image is defined as having no border, with the width and height of the sand.jpg image, level 0 detail, texel datatype GL_UNSIGNED BYTE and the pointer to the image which is provided by the image class's imagefield() method. Finally, it sets the Texture parameters of the current texture to repeat the texture, since the texture resolution is small. Since the texture has thousands of small pieces of sand, the texture is set to average out the texture coordinates that are closest in both cases where the texture needs to be minified or magnified by setting them to GL_LINEAR. This creates a smooth sand floor and makes up for the texture's low resolution.

Once the floor is created, the buffer binds back to the default texture. The floor is a rectangular plane created when the plane() function is called:

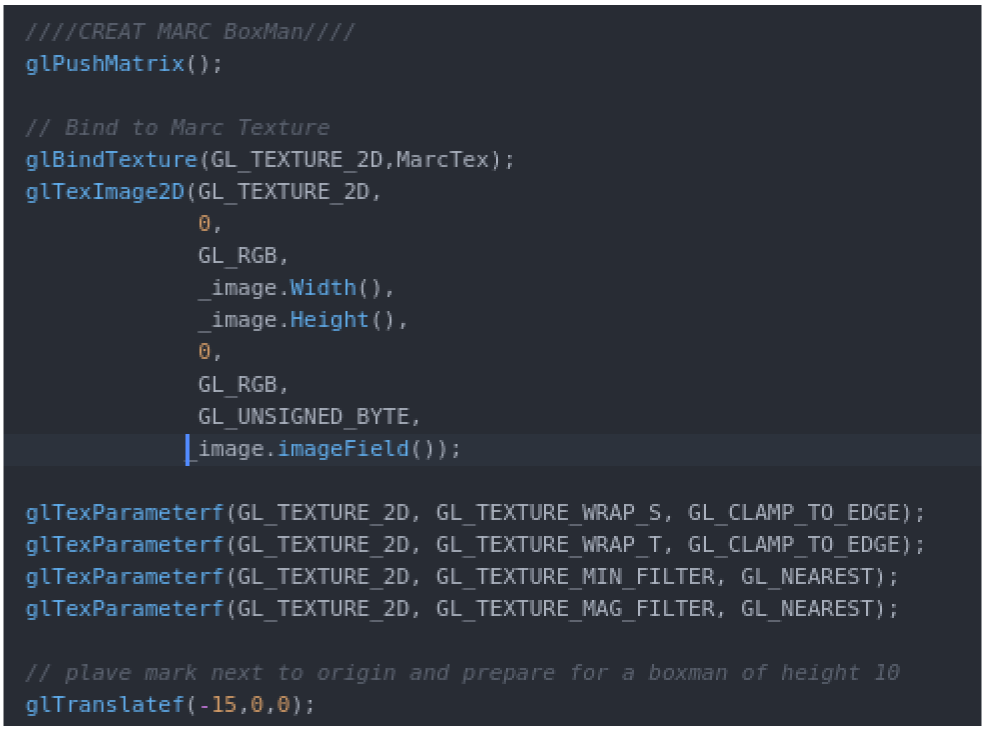


```
// creates a plane using provided material properties and normal coordinates
void EgyptWidget::plane(const materialStruct &material){
  //enable texturing
  glEnable(GL_TEXTURE_2D);

  //apply material properties to front of plane
  glMaterialfv(GL_FRONT, GL_AMBIENT,    material.ambient);
  glMaterialfv(GL_FRONT, GL_DIFFUSE,    material.diffuse);
  glMaterialfv(GL_FRONT, GL_SPECULAR,   material.specular);
  glMaterialf(GL_FRONT, GL_SHININESS,   material.shininess);

  //apply material properties to back of plane
  glMaterialfv(GL_BACK, GL_AMBIENT,    material.ambient);
  glMaterialfv(GL_BACK, GL_DIFFUSE,    material.diffuse);
  glMaterialfv(GL_BACK, GL_SPECULAR,   material.specular);
  glMaterialf(GL_BACK, GL_SHININESS,   material.shininess);

  GLfloat normals[][3] = { {0, 0 ,1} };

  glNormal3fv(normals[0]);
  glBegin(GL_POLYGON);
    glTexCoord2f(0.0, 0.0);
    glVertex3f(-1.0, -1.0, 1.0);
    glTexCoord2f(1.0, 0.0);
    glVertex3f( 1.0, -1.0, 1.0);
    glTexCoord2f(1.0, 1.0);
    glVertex3f( 1.0,  1.0, 1.0);
    glTexCoord2f(0.0, 1.0);
    glVertex3f(-1.0,  1.0, 1.0);
  glEnd();

  //disable texturing
  glDisable(GL_TEXTURE_2D);
}
```

This function receives material properties that are applied to both the front and back of the plane. It creates an upward facing plane, therefore the normalised normal to the plane is (0,0,1), since z is perpendicular to both the x and y vectors that are equivalent to the vectors forming the plane facing up. The 2d texels are applied to the plane such that the lower left corner of the texture ((0.0,0.0)) is applied to the coordinate at the bottom left of the plane (-1.0,-1.0,1.0), the bottom right corner (1.0,0.0) is applied to the coordinate at the bottom right (1.0,-1.0,1.0) and so on… . This is done using the glTexCoord2f() call before defining a vertex of the plane using glVertex3f(). Texturing is enabled at the beginning of this function and disabled at its end so that the texture is only applied while drawing. This is a design choice that is made to prevent textures from being applied to objects where it is not required.

Note: After every occurrence of instancing where an object is created and a transformation takes place, the matrix is reset to its default state being unscaled, not translated or rotated, with its position at the origin, through the use of glPushMatrix() and glPopMatrix() meaning that the entire scene uses the origin as a reference point.

Next the code binds the marc texture to GL_TEXTURE_2D, making it a 2d texture,. As before, it sets the texture properties to the image's properties, sets the texture parameters such that the image clamps to the edges of the object and when minified or magnified, such that the nearest texture coordinates are used. It translates -15 on the x axis to move marc to the left and checks if the marc's abduction animation is occurring, that is if an abduction is occurring and marc should be ascending, which is signified by the corresponding boolean variables.

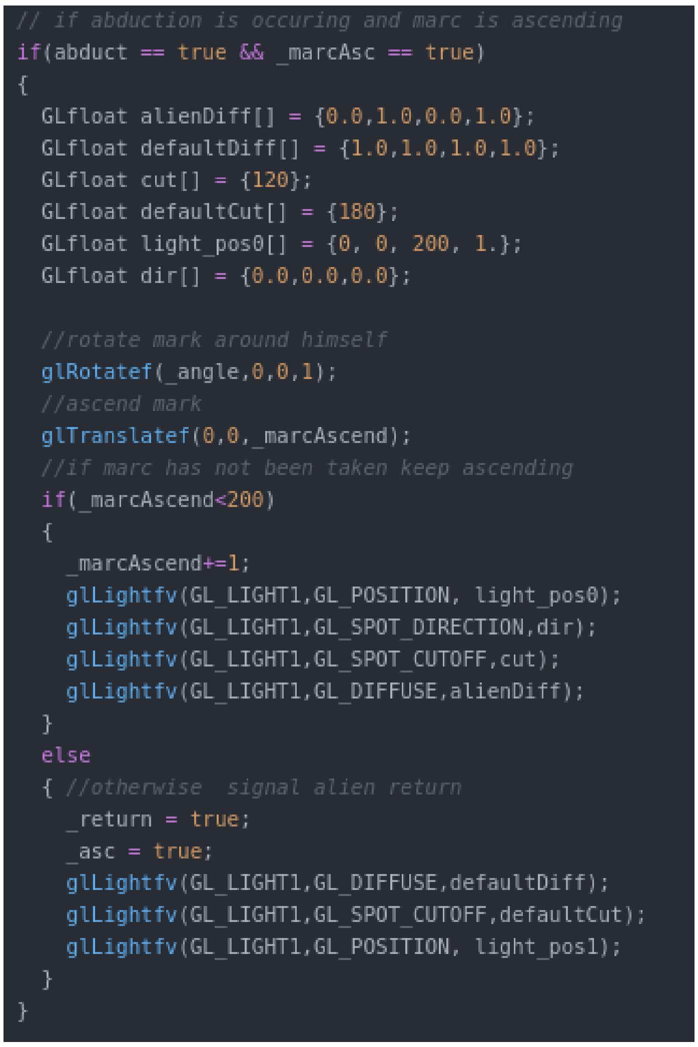


```
////CREAT MARC BoxMan////
glPushMatrix();

// Bind to Marc Texture
glBindTexture(GL_TEXTURE_2D,MarcTex);
glTexImage2D(GL_TEXTURE_2D,
             0,
             GL_RGB,
             _image.Width(),
             _image.Height(),
             0,
             GL_RGB,
             GL_UNSIGNED_BYTE,
             _image.imageField());

glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

// plave mark next to origin and prepare for a boxman of height 10
glTranslatef(-15,0,0);
```

If that is true, then Marc Is rotated around the z-axis on his centre using the angle variable, and starts ascending upwards by a translation through the z-axis. Once this becomes true, then while he is not in the alien, he continues to ascend and GL_LIGHT1 is placed at (0,0,200), facing towards the origin, its diffuse component is set to green and the angle of the light is cut at 120 degrees. On the other hand, If Marc has entered the alien, meaning that he has reached a height of 200, the return variable is set to true to signal the return of the alien and the position and light properties of GLLIGHT1 are set to their previous values.

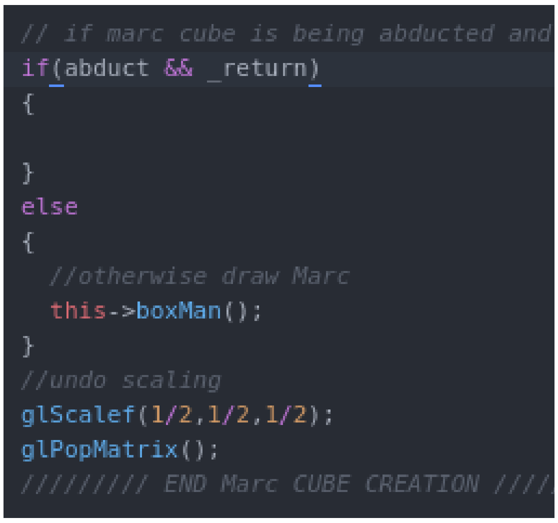
Before Marc is drawn, all dimensions of the current matrix are doubled so that marc's size will also double. The only case where Marc should not be drawn is when an abduction is occurring and the alien's return has been signalled. Otherwise Marc is drawn.

Marc's Abduction animation:



```
// if abduction is occuring and marc is ascending
if(abduct == true && _marcAsc == true)
{
  GLfloat alienDiff[] = {0.0,1.0,0.0,1.0};
  GLfloat defaultDiff[] = {1.0,1.0,1.0,1.0};
  GLfloat cut[] = {120};
  GLfloat defaultCut[] = {180};
  GLfloat light_pos0[] = {0, 0, 200, 1.};
  GLfloat dir[] = {0.0,0.0,0.0};

  //rotate mark around himself
  glRotatef(_angle,0,0,1);
  //ascend mark
  glTranslatef(0,0,_marcAscend);
  //if marc has not been taken keep ascending
  if(_marcAscend<200)
  {
    _marcAscend+=1;
    glLightfv(GL_LIGHT1,GL_POSITION, light_pos0);
    glLightfv(GL_LIGHT1,GL_SPOT_DIRECTION,dir);
    glLightfv(GL_LIGHT1,GL_SPOT_CUTOFF,cut);
    glLightfv(GL_LIGHT1,GL_DIFFUSE,alienDiff);
  }
  else
  { //otherwise  signal alien return
    _return = true;
    _asc = true;
    glLightfv(GL_LIGHT1,GL_DIFFUSE,defaultDiff);
    glLightfv(GL_LIGHT1,GL_SPOT_CUTOFF,defaultCut);
    glLightfv(GL_LIGHT1,GL_POSITION, light_pos1);
  }
}
```

Marc is drawn:



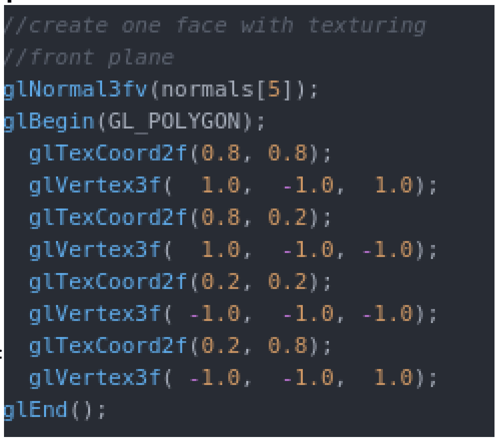
```
// if marc cube is being abducted and
if(abduct && _return)
{

}
else
{
  //otherwise draw Marc
  this->boxMan();
}
//undo scaling
glScalef(1/2,1/2,1/2);
glPopMatrix();
///////// END Marc CUBE CREATION ////
```

Marc is drawn by calling the boxMan() method. This method uses scaled cubes to create a man out of boxes. The cubes are created using the cube() function.

Cube():

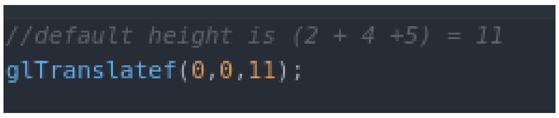
This cube function is is based on the one from the banksy tutorial in solidCubeWidget.cpp, It creates a brass cube with the same coordinates as the solidCubeWidget, placing a texture on the plane on the front of the cube. The texels are set such that 20% of the height of the image on top and bottom are cropped, and 20% on the left and right of the image are cropped. Like all objects, the texturing is enabled at the beginning of the method and disabled at the end of it, while GL_TEXTURE_2D is bound to the default texture right after the plane with texturing is created..
Since the normals for each plane are provided, there is no need to calculate them.

Texturing applied in cube() front-facing plane



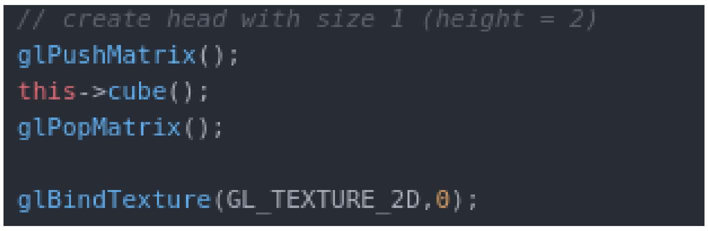
```
//create one face with texturing
//front plane
glNormal3fv(normals[5]);
glBegin(GL_POLYGON);
  glTexCoord2f(0.8, 0.8);
  glVertex3f(  1.0,  -1.0,  1.0);
  glTexCoord2f(0.8, 0.2);
  glVertex3f(  1.0,  -1.0, -1.0);
  glTexCoord2f(0.2, 0.2);
  glVertex3f( -1.0,  -1.0, -1.0);
  glTexCoord2f(0.2, 0.8);
  glVertex3f( -1.0,  -1.0,  1.0);
glEnd();
```

In the boxman() function, the boxman is created from top to bottom. The function translates to the top of the boxman at a height of 11.



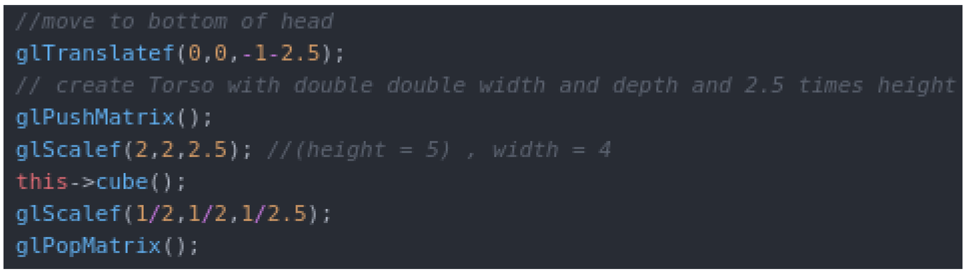
```
//default height is (2 + 4 +5) = 11
glTranslatef(0,0,11);
```

It creates a cube using the cube function that by default has height 2 (since its highest point is 1 and its lowest is -1) with the texture currently binded to representing the head and then binds back to default texture.



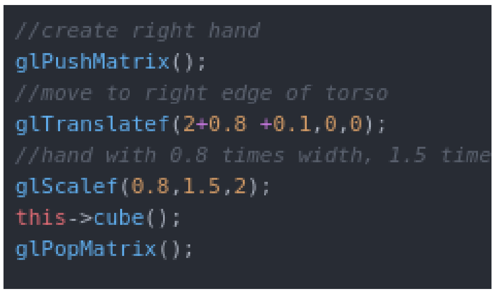
```
// create head with size 1 (height = 2)
glPushMatrix();
this->cube();
glPopMatrix();

glBindTexture(GL_TEXTURE_2D,0);
```

It then translates to the bottom of the head and creates another cube that is double the width and depth, but 2.5 times the height of the head using scaling to represent the torso. Since the cube is drawn from its centre, the translation needs to be equal to the half the height of the head plus half the height of the torso downwards (-(1+2.5).

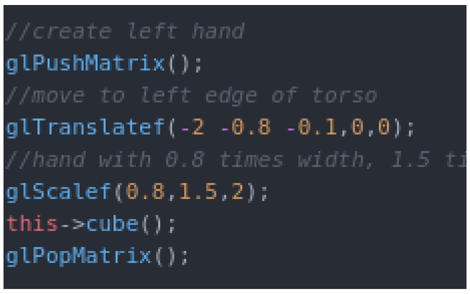


```
//move to bottom of head
glTranslatef(0,0,-1-2.5);
// create Torso with double double width and depth and 2.5 times height
glPushMatrix();
glScalef(2,2,2.5); //(height = 5) , width = 4
this->cube();
glScalef(1/2,1/2,1/2.5);
glPopMatrix();
```

It then translates to the left and right of the torso and creates two more cubes that are scaled such that they have 0.8 times the width, 1.5 times the depth and double the height of the head to represent the arms. These translations from the centre of the torso are set to be equal to half the width of the torso plus half the width of the hand (4/2+1.6/2) . An extra 0.1 is added to make the separation of the arms from the torso more visible and aesthetically pleasing.

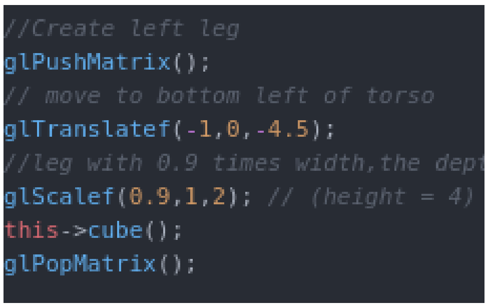


```
//create right hand
glPushMatrix();
//move to right edge of torso
glTranslatef(2+0.8 +0.1,0,0);
//hand with 0.8 times width, 1.5 time
glScalef(0.8,1.5,2);
this->cube();
glPopMatrix();
```

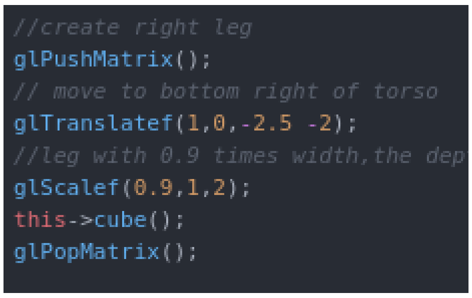


```
//create left hand
glPushMatrix();
//move to left edge of torso
glTranslatef(-2 -0.8 -0.1,0,0);
//hand with 0.8 times width, 1.5 ti
glScalef(0.8,1.5,2);
this->cube();
glPopMatrix();
```

Finally, from the centre of the torso, it translates to the bottom left and right of the torso by translating half the distance of half the torso's width (2/2) to the left or right and then half the torso's height plus half the height of the leg cube downwards (2.5+2). The leg is set to have 0.9 times the width of the head so that separation between the two legs is visible, the same depth as the head and double the head's height.



```
//Create left leg
glPushMatrix();
// move to bottom left of torso
glTranslatef(-1,0,-4.5);
//leg with 0.9 times width,the dept
glScalef(0.9,1,2); // (height = 4)
this->cube();
glPopMatrix();
```



```
//create right leg
glPushMatrix();
// move to bottom right of torso
glTranslatef(1,0,-2.5 -2);
//leg with 0.9 times width,the dep
glScalef(0.9,1,2);
this->cube();
glPopMatrix();
```

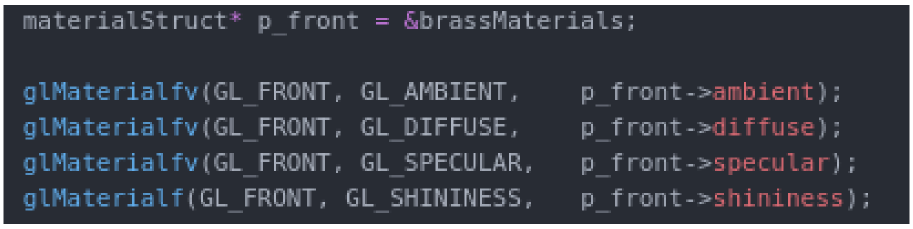
In this way, The boxman's vertically stacked components all add up to 11, which is the height translated to, and the size of all its components is related through the head..

Once the boxman is drawn, the changes to the current matrix are discarded using glPopMatrix() and a brass billboard is drawn at the origin using the drawBillboard() function.

drawBillBoard():

This function accepts a GLUint texture name, two unsigned integers representing the width and height dimensions of the 2D image to be drawn, along with the pointer to that image's contents stored in Glubyte type as parameters.

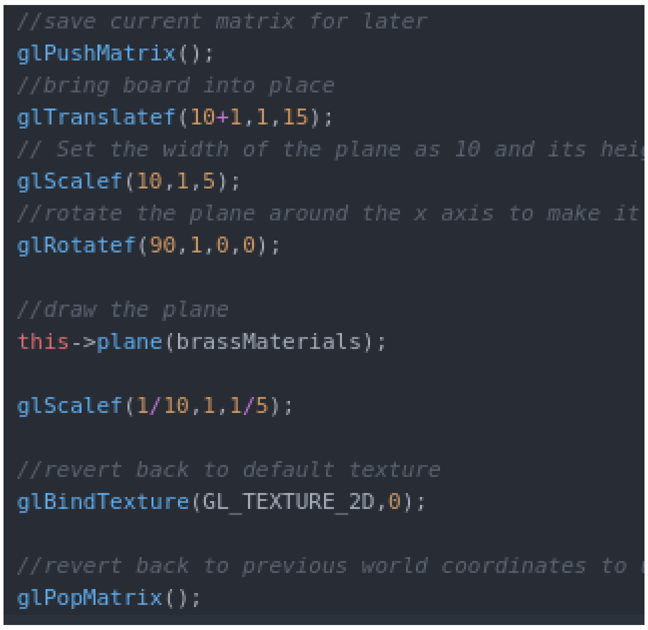
The function sets the material properties to brassMaterials using the materialStruct and glMaterialfv() to set the material properties for each light component for the front side of the billboard objects.



```
materialStruct* p_front = &brassMaterials;

glMaterialfv(GL_FRONT, GL_AMBIENT,    p_front->ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE,    p_front->diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR,   p_front->specular);
glMaterialf(GL_FRONT, GL_SHININESS,   p_front->shininess);
```

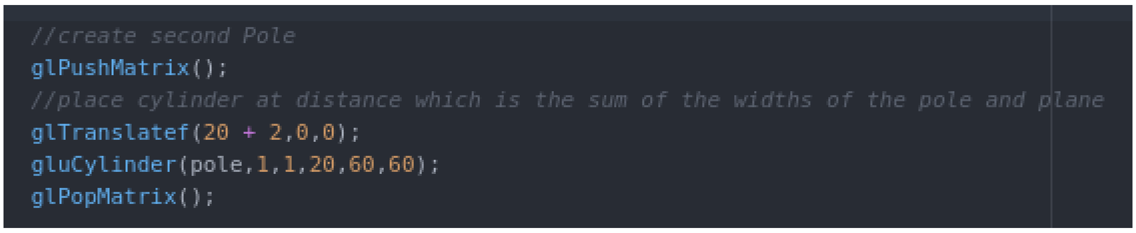
It draws a gluCylinder with a constant radius of 1 and a height of 20. It then applies the texture passed to it as parameter using the image properties and clamps the texture on a plane.

The plane is elevated to a height of 15 and is brought to the right of the centre of the cylinder through a distance of the radius + half the length of the plane. The plane is then rotated 90 degrees around the x-axis to face the camera and is drawn to be brass, applying the texture provided in texels. The default texture is then restored.



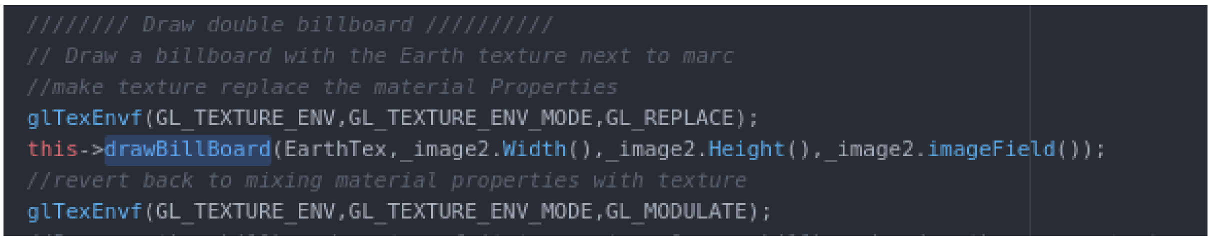
```
//save current matrix for later
glPushMatrix();
//bring board into place
glTranslatef(10+1,1,15);
// Set the width of the plane as 10 and its hei
glScalef(10,1,5);
//rotate the plane around the x axis to make it
glRotatef(90,1,0,0);

//draw the plane
this->plane(brassMaterials);

glScalef(1/10,1,1/5);

//revert back to default texture
glBindTexture(GL_TEXTURE_2D,0);

//revert back to previous world coordinates to
glPopMatrix();
```

Another identical glucylinder is placed at a distance of the length of the plane + the length of the diameter of the glucylinder from the centre of the first cylinder.



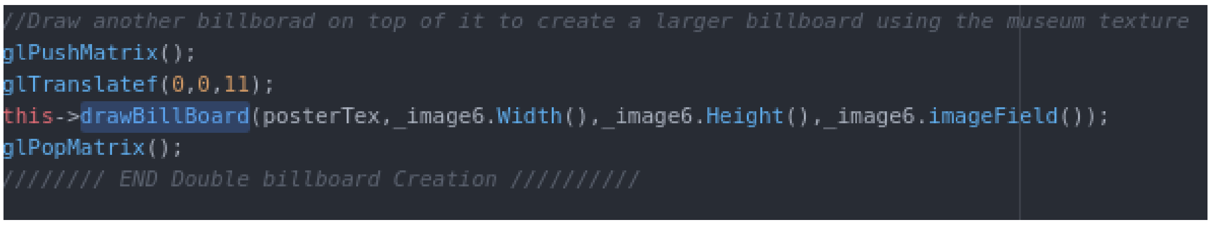
```
//create second Pole
glPushMatrix();
//place cylinder at distance which is the sum of the widths of the pole and plane
glTranslatef(20 + 2,0,0);
gluCylinder(pole,1,1,20,60,60);
glPopMatrix();
```

Before the first billboard is drawn, the texture environment mode is set to replace the material properties of the plane, and reset back to the default which mixes the material properties of the plane with the texture after it is drawn. This is to make the earthmap texture appear bright regardless of light conditions and remain unaffected by the object's material properties.



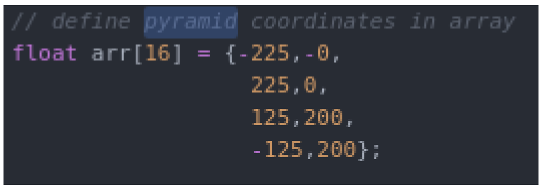
```
//////// Draw double billboard //////////
// Draw a billboard with the Earth texture next to marc
//make texture replace the material Properties
glTexEnvf(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,GL_REPLACE);
this->drawBillBoard(EarthTex,_image2.Width(),_image2.Height(),_image2.imageField());
//revert back to mixing material properties with texture
glTexEnvf(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,GL_MODULATE);
```

To simulate a double billboard, another billboard including the museum.jpg image is drawn above the first one, merging the poles of the billboards to appear as one.

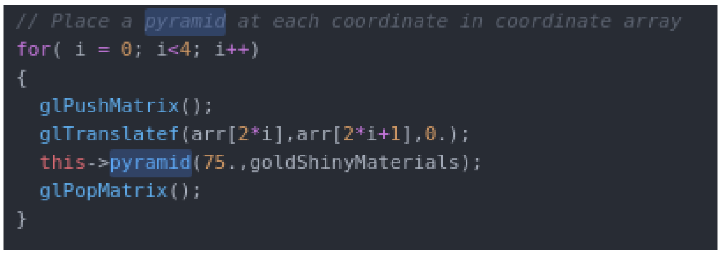


```
//Draw another billborad on top of it to create a larger billboard using the museum texture
glPushMatrix();
glTranslatef(0,0,11);
this->drawBillBoard(posterTex,_image6.Width(),_image6.Height(),_image6.imageField());
glPopMatrix();
//////// END Double billboard Creation //////////
```

Once this is done, the pyramid texture is bound to with the same texture parameters as the Marc cube and 4 pyramids are drawn by storing their coordinates in a float array such that every pair of float values form the x and y coordinates of a pyramid at height 0. For each pair, a pyramid of size 75 and material properties of goldShinyMaterials is drawn at those coordinates as a translation from the origin.



```
// define pyramid coordinates in array
float arr[16] = {-225,-0,
                 225,0,
                 125,200,
                 -125,200};
```

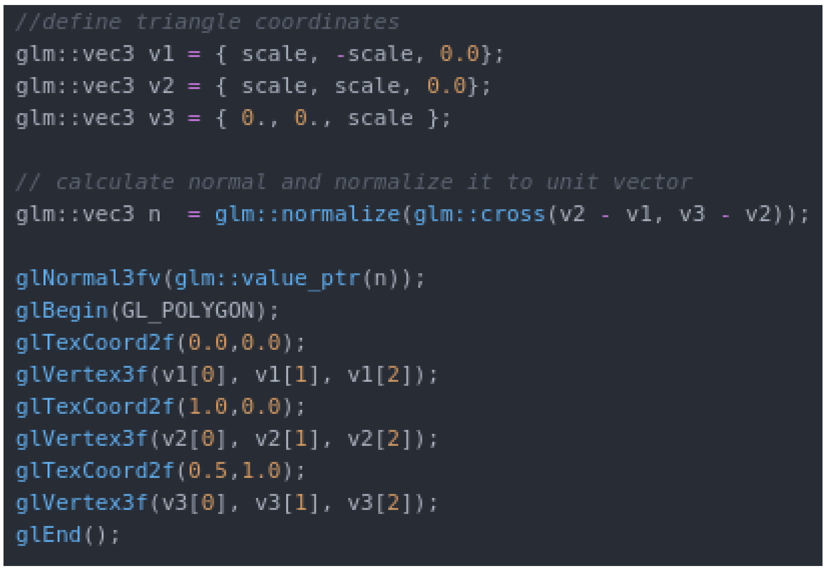


```
// Place a pyramid at each coordinate in coordinate array
for( i = 0; i<4; i++)
{
  glPushMatrix();
  glTranslatef(arr[2*i],arr[2*i+1],0.);
  this->pyramid(75.,goldShinyMaterials);
  glPopMatrix();
}
```

Each pyramid is drawn using a slightly altered pyramid() function from the orthographic tutorial. This function receives a float value named scale that defines the size of the pyramid and a materialStruct that defines its material properties. At the beginning, the function enables texturing and applies the material properties to the pyramid.

Then, it defines 3 points in vector coordinates for each triangle forming the pyramid using the scale value to set the length of each triangle side and its height. Once the triangle points are defined, two pairs of points are subtracted from each other in counter-clockwise order. When points are subtracted from one another, a direction is derived so two vectors formed by this subtraction are obtained that are two directions that form the surface of the triangle. Then, the cross product of these two vectors is calculated using glm::cross(), which gives us a vector that is perpendicular to both the vectors previously obtained, which is the normal to the surface of the triangle. Since the points were taken in counter-clockwise order, the normal's direction is on the outside of the surface and glm::normalise is called to normalise the normal so that it defines a direction.

Texturing is applied to all triangles such that the bottom left and right texel coordinates (0,0) and (1,0) of the image are mapped to the bottom left and right coordinates of the triangle, while the middle top texel coordinate (0.5,1.0) is mapped to the top coordinate of the triangle, taking on the form of hieroglyphics/wallpaper.



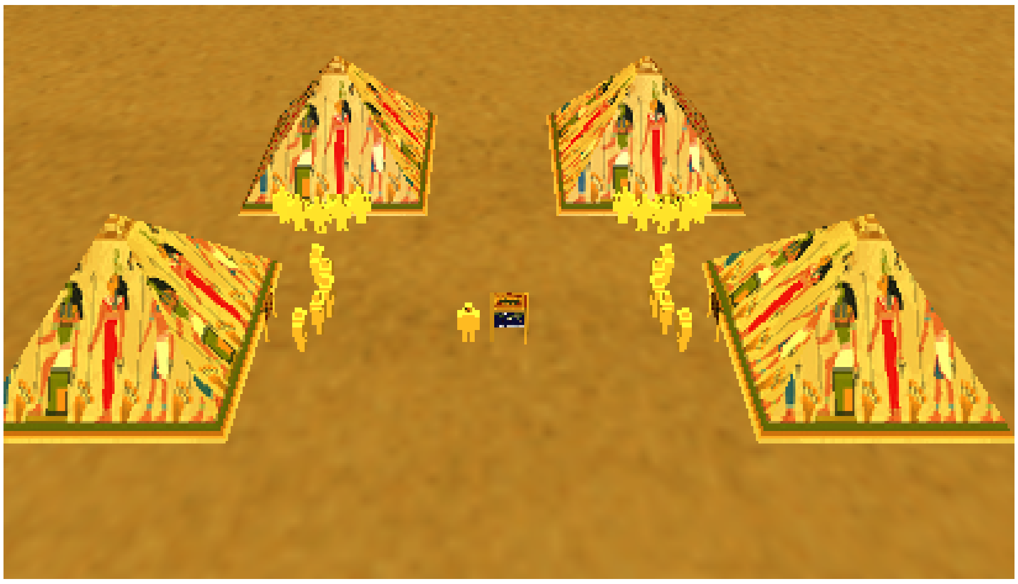
```
//define triangle coordinates
glm::vec3 v1 = { scale, -scale, 0.0};
glm::vec3 v2 = { scale, scale, 0.0};
glm::vec3 v3 = { 0., 0., scale };

// calculate normal and normalize it to unit vector
glm::vec3 n  = glm::normalize(glm::cross(v2 - v1, v3 - v2));

glNormal3fv(glm::value_ptr(n));
glBegin(GL_POLYGON);
glTexCoord2f(0.0,0.0);
glVertex3f(v1[0], v1[1], v1[2]);
glTexCoord2f(1.0,0.0);
glVertex3f(v2[0], v2[1], v2[2]);
glTexCoord2f(0.5,1.0);
glVertex3f(v3[0], v3[1], v3[2]);
glEnd();
```

In the same way, another array containing 4 pairs of coordinate translations from the origin is traversed through to draw 4 billBoards with the museum texture next to the pyramids and for each iteration the equivalent degrees for rotation around the z axis to make the billboards face away from the pyramids are obtained from an integer array of 4 angles.

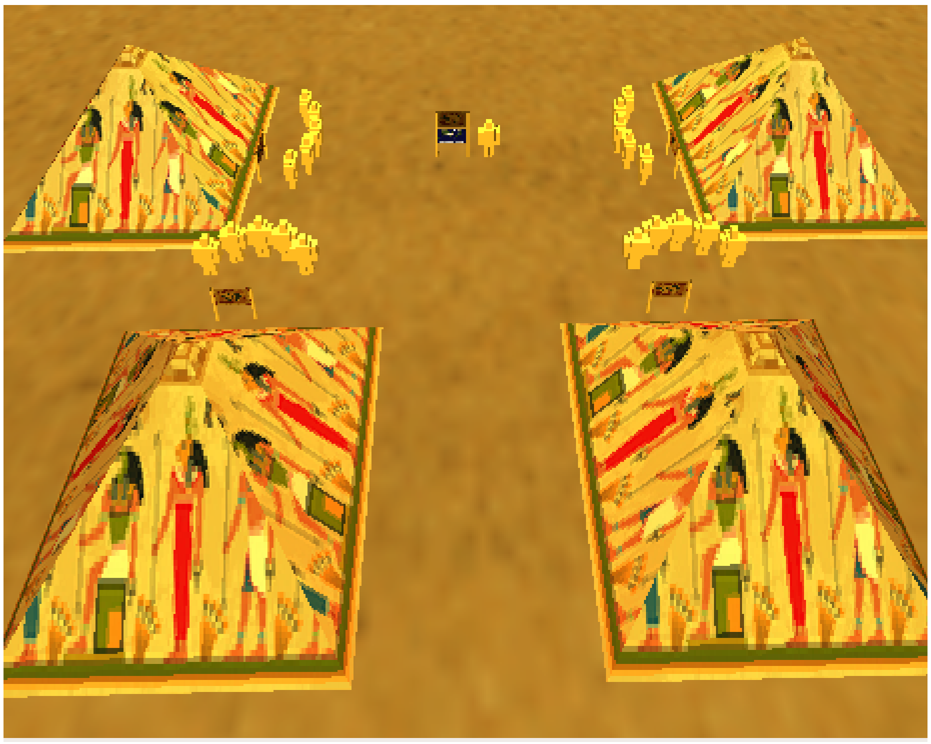
Then, if it the user input variable _day is true, then it loops through the array of billboard coordinates and rotates around the z- axis at those points at the angles in an angle array that are in sequence incremented by intervals of 22.5 degrees. The rotation occurs at a radius of 40 and at each angle a boxman is drawn such that it is facing towards the billboard. Again, the boxman is drawn at twice its size through scaling.

This is the resulting scene created through instancing:

Front View:



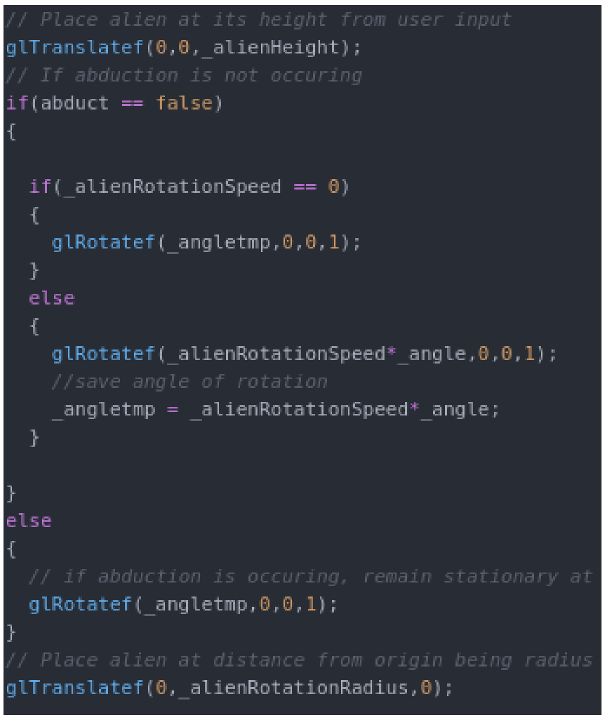
Back View:



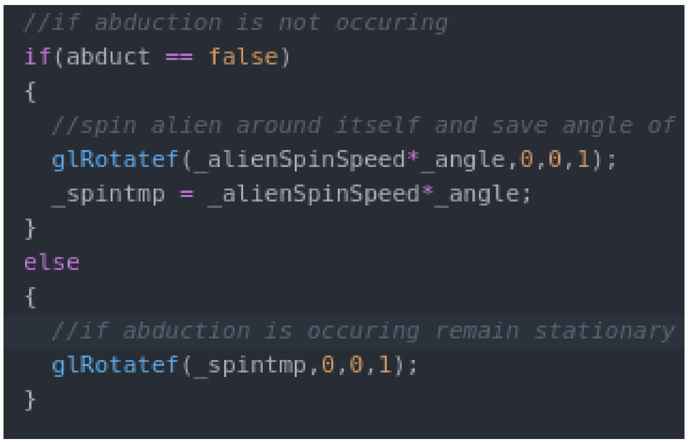
Animation:

The abduction animation is handled by the animate() function of the EgyptWidget class, The animation places the alien at the user input _alienHeight value on the z-axis. Note that the matrix is the one that is being manipulated, however it is referred to as the alien as a means of better explanation.

If the abduction is not occurring, and the alien's rotation speed is 0, it stops the rotation around the z-axis at the last recorded angle held by _angletmp double variable. If the abduction is not occurring and the alien's rotation speed is not zero, then it rotates the current value of the _angle by the _alienRotationSpeed, meaning that every time the screen is refreshed the angle of rotation is incremented by the value of _alienRotationSpeed. The angle of rotation is then saved to variable _angletmp and the alien is translated on the positive y axis by the _alienRotationRadius value, bringing the alien at an orbiting position and rotating around the origin. Then, given that the abduction is not occurring, the alien is rotated around the z-axis where the z-axis is at its centre causing it to spin using a rotation of degrees _alienSpinSpeed *angle and the angle of rotation is saved to a variable called _spintmp.



```
// Place alien at its height from user input
glTranslatef(0,0,_alienHeight);
// If abduction is not occuring
if(abduct == false)
{

  if(_alienRotationSpeed == 0)
  {
    glRotatef(_angletmp,0,0,1);
  }
  else
  {
    glRotatef(_alienRotationSpeed*_angle,0,0,1);
    //save angle of rotation
    _angletmp = _alienRotationSpeed*_angle;
  }

}
else
{
  // if abduction is occuring, remain stationary at
  glRotatef(_angletmp,0,0,1);
}
// Place alien at distance from origin being radius
glTranslatef(0,_alienRotationRadius,0);
```

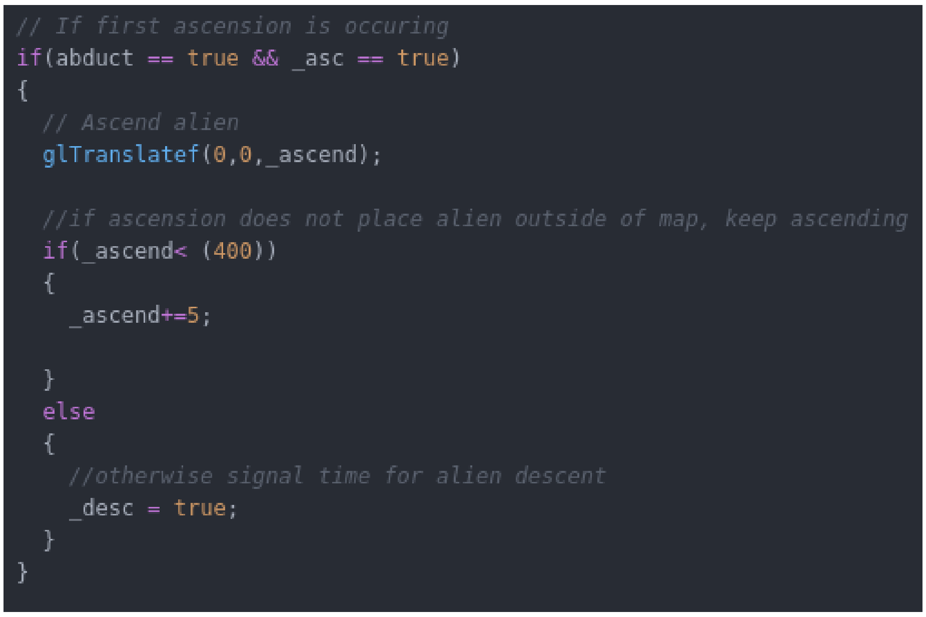


```
//if abduction is not occuring
if(abduct == false)
{
  //spin alien around itself and save angle of
  glRotatef(_alienSpinSpeed*_angle,0,0,1);
  _spintmp = _alienSpinSpeed*_angle;
}
else
{
  //if abduction is occuring remain stationary
  glRotatef(_spintmp,0,0,1);
}
```

The alienAbduct() slot sets the abduct and _asc variables to true, signalling that an abduction is taking place and that the alien should be ascending.

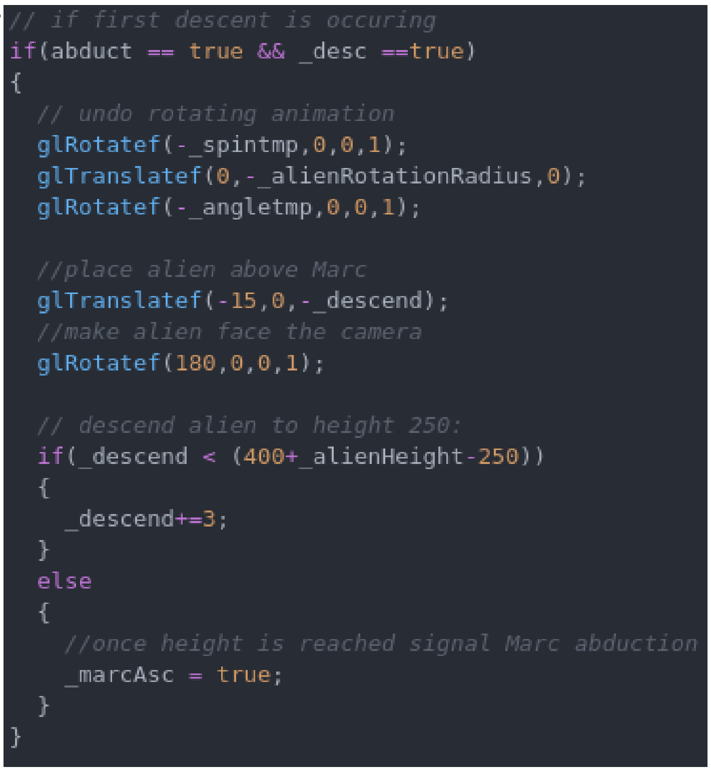
In the case that an abduction is occurring, the matrix of the alien is rotated by the last recorded angle around the origin, and translated to the corresponding height and radius from the origin, as well as rotated around the z-axis at the position of orbit by the last recorded spin angle stored in _spintmp.

If abduct and _asc are true, then the alien matrix is translated upwards by _ascend that increases by 5 every time the screen is refreshed if it is less 400 which is the top height that the alien should ascend through and takes him outside the view volume of the camera. The screen is refreshed until _ascend is higher than 400 creating this animation. When the limit is exceeded, _desc is set to true signalling that the alien should descend.



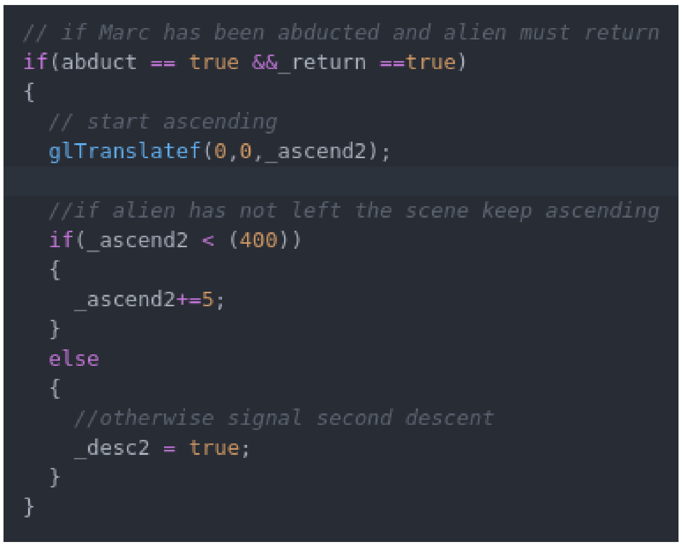
```
// If first ascension is occuring
if(abduct == true && _asc == true)
{
  // Ascend alien
  glTranslatef(0,0,_ascend);

  //if ascension does not place alien outside of map, keep ascending
  if(_ascend< (400))
  {
    _ascend+=5;

  }
  else
  {
    //otherwise signal time for alien descent
    _desc = true;
  }
}
```

If abduct and _desc are true, then the inverse transformations are performed to undo the last rotating animation of the alien around the origin, the alien is translated on the negative x-axis by -15 to be placed above Marc, and translated on downwards through the z-axis by descend value. The alien is also rotated 180 degrees so that the textured eye faces the camera. In order to ensure that the alien has descended to the coordinates (-15,0,250), the animate function checks if descend has surpassed the value of (400+alienHeight -250) , since the height ascended was 400+alienHeight and once the alien descends downwards 250 points less than that required to reach the ground, it should reach the coordinates required. If _descend has not reached/exceeded the value yet, _descend is incremented by3. If _descend has reached the required value, then _marAsc is set to true signalling that mark needs to be ascending, triggering marc's animation.

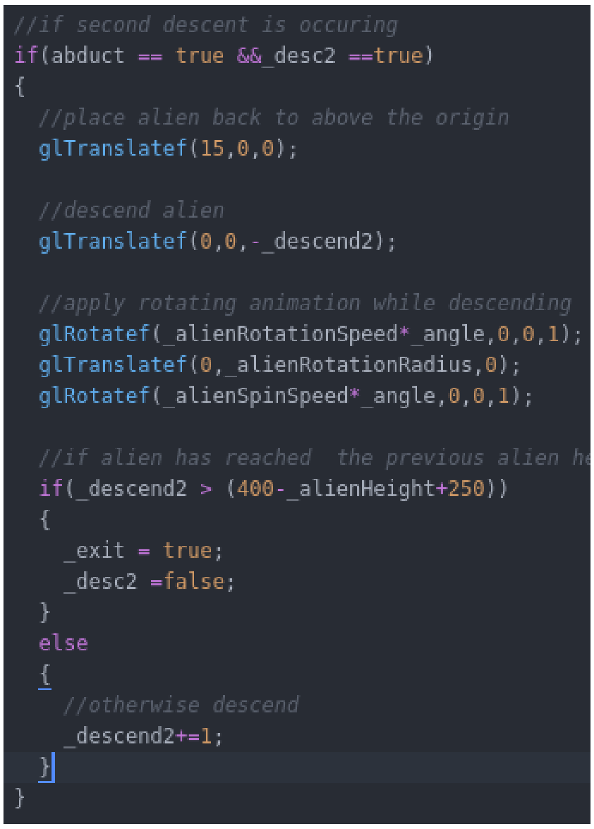


```
// if first descent is occuring
if(abduct == true && _desc ==true)
{
  // undo rotating animation
  glRotatef(-_spintmp,0,0,1);
  glTranslatef(0,-_alienRotationRadius,0);
  glRotatef(-_angletmp,0,0,1);

  //place alien above Marc
  glTranslatef(-15,0,-_descend);
  //make alien face the camera
  glRotatef(180,0,0,1);

  // descend alien to height 250:
  if(_descend < (400+_alienHeight-250))
  {
    _descend+=3;
  }
  else
  {
    //once height is reached signal Marc abduction
    _marcAsc = true;
  }
}
```

Once Marc has entered the alien and is no longer being drawn, _return should be true, meaning that upon all the previous transformations, the current matrix will also be translated upwards by the value of _ascend2 until it reaches 400 and re-exits the camera's view space. Once this Is done, the second descend is signalled by setting _desc2 to true.

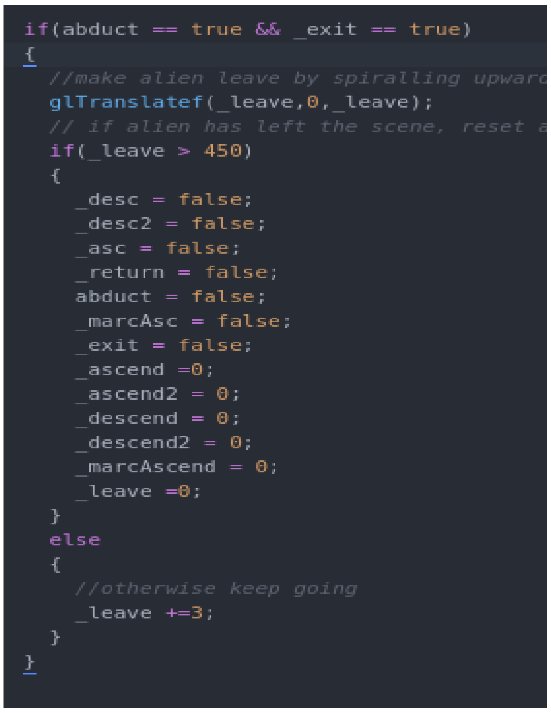


```
// if Marc has been abducted and alien must return
if(abduct == true && _return ==true)
{
  // start ascending
  glTranslatef(0,0,_ascend2);

  //if alien has not left the scene keep ascending
  if(_ascend2 < (400))
  {
    _ascend2+=5;
  }
  else
  {
    //otherwise signal second descent
    _desc2 = true;
  }
}
```



```
//if second descent is occuring
if(abduct == true && _desc2 ==true)
{
  //place alien back to above the origin
  glTranslatef(15,0,0);

  //descend alien
  glTranslatef(0,0,-_descend2);

  //apply rotating animation while descending
  glRotatef(_alienRotationSpeed*_angle,0,0,1);
  glTranslatef(0,_alienRotationRadius,0);
  glRotatef(_alienSpinSpeed*_angle,0,0,1);

  //if alien has reached  the previous alien he
  if(_descend2 > (400-_alienHeight+250))
  {
    _exit = true;
    _desc2 =false;
  }
  else
  {
    //otherwise descend
    _descend2+=1;
  }
}
```

If abduct is true and _desc2 are true, then on top of all previous transformations, the translation on the negative x-axis to position the alien above Marc is undone using the inverse transformation, the alien is translated to descend on the z-axis by the value of_descend2, and the rotation animation is reapplied, causing the alien to descend while rotating. If the alien has not reached its initial height then it will keep descending by increments of 1. This is checked by doing the opposite calculation than during ascent. If the alien has reached its original height, then _exit is set to true, signalling the exit of the alien and setting _desc2 to false, cancelling the _desc2 transformation.

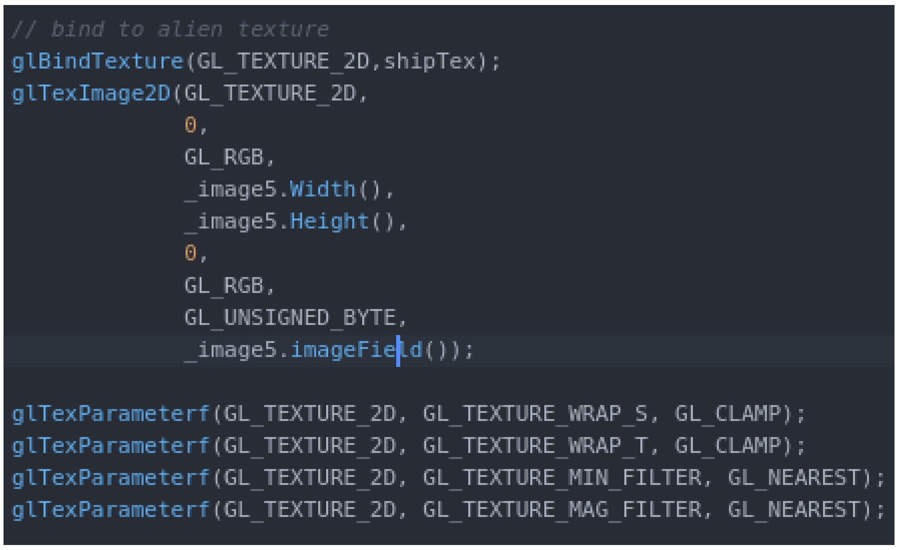


```
if(abduct == true && _exit == true)
{
  //make alien leave by spiralling upwarc
  glTranslatef(_leave,0,_leave);
  // if alien has left the scene, reset a
  if(_leave > 450)
  {
    _desc = false;
    _desc2 = false;
    _asc = false;
    _return = false;
    abduct = false;
    _marcAsc = false;
    _exit = false;
    _ascend =0;
    _ascend2 = 0;
    _descend = 0;
    _descend2 = 0;
    _marcAscend = 0;
    _leave =0;
  }
  else
  {
    //otherwise keep going
    _leave +=3;
  }
}
```

If abduct and _exit are true, then the alien is translated in the x and z direction by _leave. _leave is incremented every time the screen refreshes by 3, causing the alien to spiral upwards and leave. Once it is translated upwards by 450, all the variables that control the animation are reset to false and 0, so that the animation can be called again.

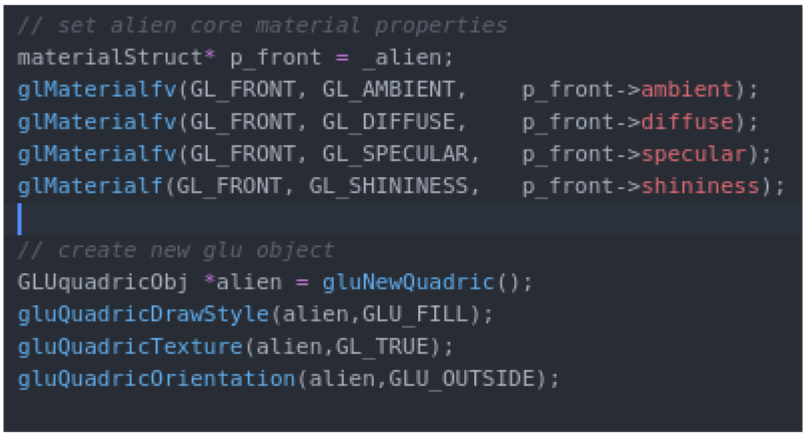
Hierarchical Modelling:

After the animation is checked and applied, the alien is drawn at the matrix transformed by it. Before this, the shipTex texture is bound to, and the 2d image is loaded. An alien is then drawn using the alien() function.



```
// bind to alien texture
glBindTexture(GL_TEXTURE_2D,shipTex);
glTexImage2D(GL_TEXTURE_2D,
             0,
             GL_RGB,
             _image5.Width(),
             _image5.Height(),
             0,
             GL_RGB,
             GL_UNSIGNED_BYTE,
             _image5.imageField());

glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
```

The alien function takes the alien core materialStruct properties and applies them to the front of the upcoming objects. It then creates a new GLUquadricObject and enables texturing for that object with GL_TRUE, on gluQuadricTexture setting its normals to face outwards .



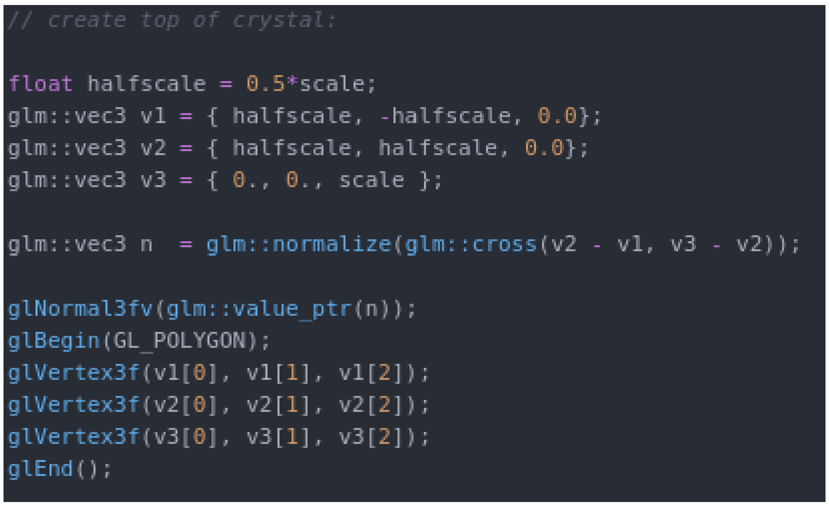
```
// set alien core material properties
materialStruct* p_front = _alien;
glMaterialfv(GL_FRONT, GL_AMBIENT,    p_front->ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE,    p_front->diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR,   p_front->specular);
glMaterialf(GL_FRONT, GL_SHININESS,   p_front->shininess);

// create new glu object
GLUquadricObj *alien = gluNewQuadric();
gluQuadricDrawStyle(alien,GLU_FILL);
gluQuadricTexture(alien,GL_TRUE);
gluQuadricOrientation(alien,GLU_OUTSIDE);
```

Then, it places a crystal 2* the alien's radius above the centre of the alien's core, at half the core's size and with the core's material properties.

A convex object created manually: Octahedron()

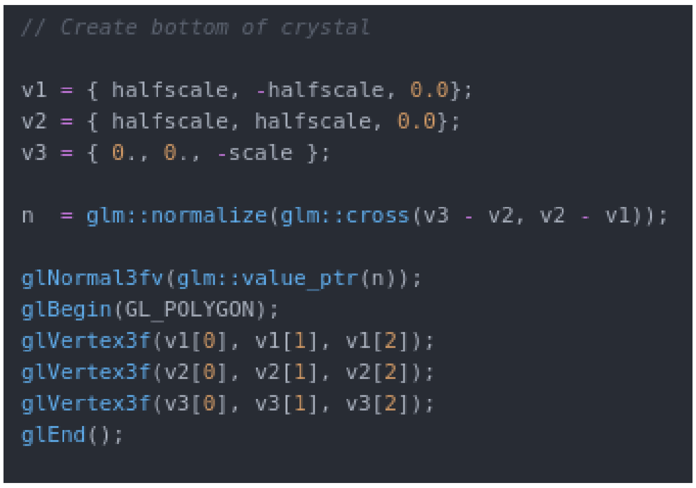
A crystal is generated by creating a scaled octahedron using the octahedron() function. This function takes a float value named scale and a materialStruct named material as parameters. It first applies the material properties through the use of glMaterialfv() to the front of the octahedron. Then, the octahedron is constructed using triangles. Part of the implementation inspired from the pyramid() function. An octahedron can also be seen as being made up of two pyramids, one facing upward s and one facing upwards.

The bottom coordinates of each triangle of the upper pyramid the octahedron is composed of, are set to half the value of scale to make the upper pyramid's base smaller, and upper coordinate of each triangle is set to the value of scale so that its height is greater. Once the triangle points are defined, two pairs of points of each triangle are subtracted from each other in counter-clockwise order. When points are subtracted from one another, a direction is derived so two vectors formed by this subtraction are obtained that are two directions that form the surface of the triangle. Then, the cross product of these two vectors is calculated using glm::cross(), which gives us a vector that is perpendicular to both the vectors previously obtained, which is the normal to the surface of the triangle. Since the points were taken in counter-clockwise order, the normal is facing away from the surface and glm::normalise is called to normalise the normal to unit length so that it defines a direction. This creates the top of the octahedron.



```
// create top of crystal:

float halfscale = 0.5*scale;
glm::vec3 v1 = { halfscale, -halfscale, 0.0};
glm::vec3 v2 = { halfscale, halfscale, 0.0};
glm::vec3 v3 = { 0., 0., scale };

glm::vec3 n  = glm::normalize(glm::cross(v2 - v1, v3 - v2));

glNormal3fv(glm::value_ptr(n));
glBegin(GL_POLYGON);
glVertex3f(v1[0], v1[1], v1[2]);
glVertex3f(v2[0], v2[1], v2[2]);
glVertex3f(v3[0], v3[1], v3[2]);
glEnd();
```

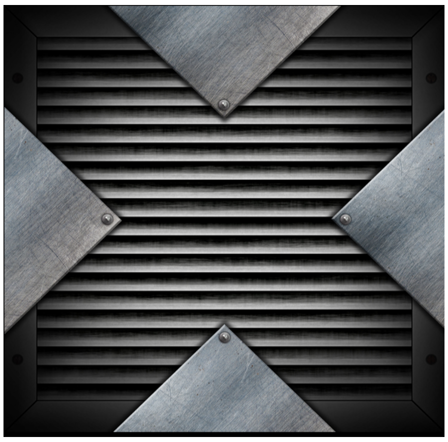
To create the bottom part of the octahedron an upside down pyramid is required. This means that the top point of each triangle which has been common between them until now needs to be changed to its negative on the z-axis counterpart, from (0,0,scale) to (0,0,-scale), so that the pyramid is now facing downwards. This is applied. Now that the top coordinates of the triangles are upside down, the order at which the triangle coordinates need to be taken needs to change so that the vectors are taken in counter-clockwise order, which is done by reversing the vectors in the cross product. The new normals are then normalised and applied to each triangle.

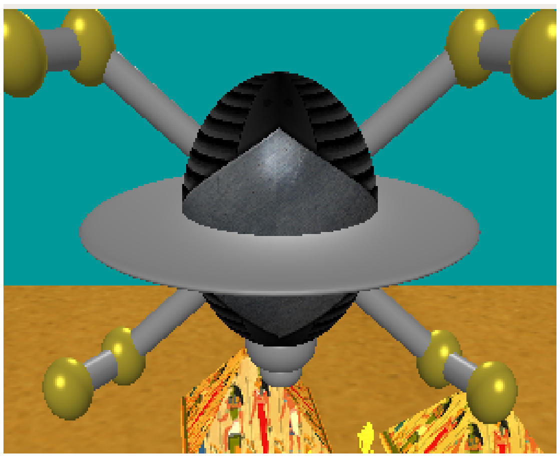


```
// Create bottom of crystal

v1 = { halfscale, -halfscale, 0.0};
v2 = { halfscale, halfscale, 0.0};
v3 = { 0., 0., -scale };

n  = glm::normalize(glm::cross(v3 - v2, v2 - v1));

glNormal3fv(glm::value_ptr(n));
glBegin(GL_POLYGON);
glVertex3f(v1[0], v1[1], v1[2]);
glVertex3f(v2[0], v2[1], v2[2]);
glVertex3f(v3[0], v3[1], v3[2]);
glEnd();
```

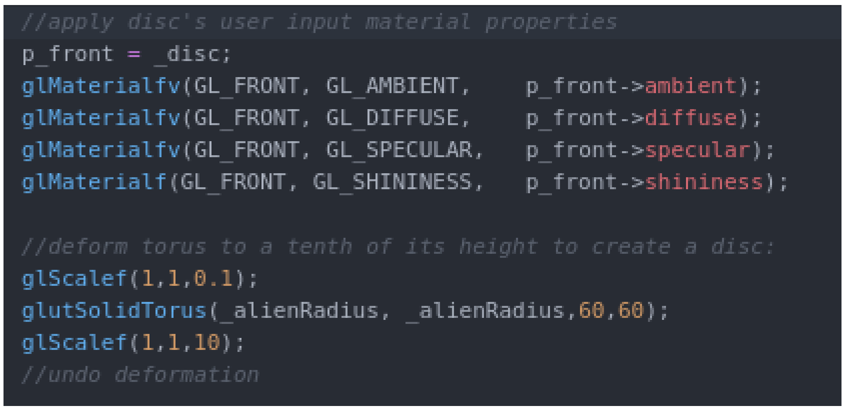
Once the crystal is created, the previous transformations are undone using glPopMatrix(), returning to the root of the tree structure of the alien, texturing is enabled and a gluSphere object is created using the properties of the alien gluQuadricObject, with _alienRadius being the user's input as the radius of the sphere. The texture of the alien has been selected such that when applied to the sphere, the two ends of the image form a rhombus eye of the alien:

Texture: ship.jpg   Applied to sphere:



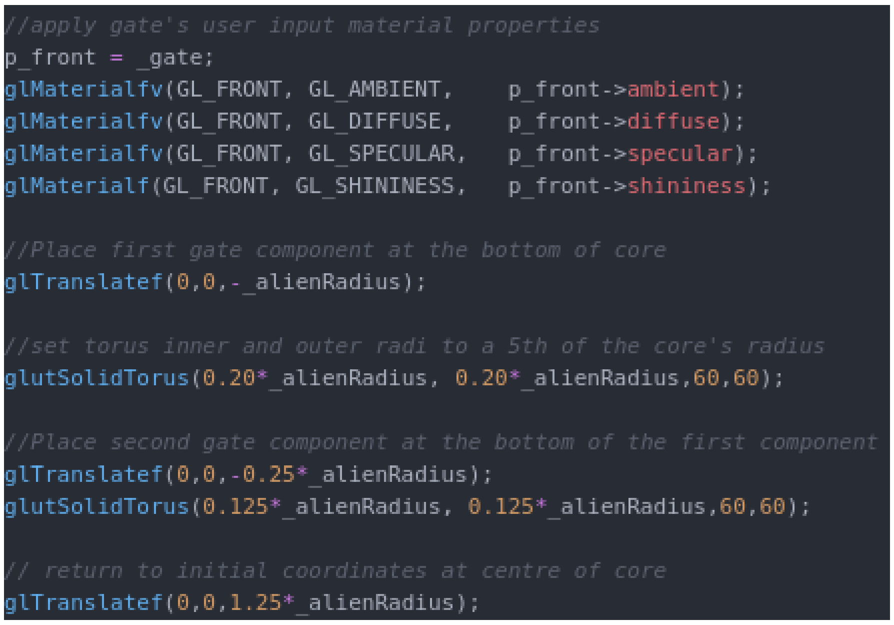


Then, the _disc material properties are loaded and applied to a glutSolidTorus() with both inside and outside radii equal to the _alienRadius that undergoes non-uniform scaling with its z-coordinates scaled to a tenth of their value, forming a disc.



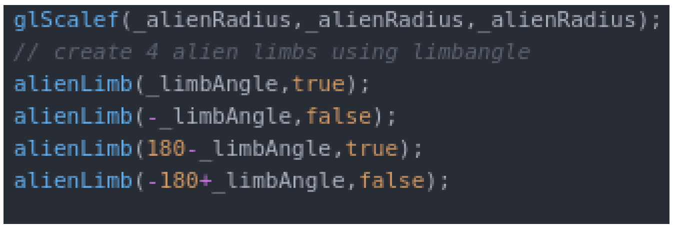
```
//apply disc's user input material properties
p_front = _disc;
glMaterialfv(GL_FRONT, GL_AMBIENT,    p_front->ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE,    p_front->diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR,   p_front->specular);
glMaterialf(GL_FRONT, GL_SHININESS,   p_front->shininess);

//deform torus to a tenth of its height to create a disc:
glScalef(1,1,0.1);
glutSolidTorus(_alienRadius, _alienRadius,60,60);
glScalef(1,1,10);
//undo deformation
```

The next node is the alien gate whose material properties stored in _gate are retrieved and applied. The function places the current matrix transform to the bottom of the core by translating by its negative radius on the z-axis, and creates Another glutSolidTorus at a fifth of the alien's radius, then reaches the bottom of the torus by translating further down the z-axis by one fourth of the alien's radius and drawing another glutSolidTorus at an eighth of the alien's radius, forming the gate.



```
//apply gate's user input material properties
p_front = _gate;
glMaterialfv(GL_FRONT, GL_AMBIENT,    p_front->ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE,    p_front->diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR,   p_front->specular);
glMaterialf(GL_FRONT, GL_SHININESS,   p_front->shininess);

//Place first gate component at the bottom of core
glTranslatef(0,0,-_alienRadius);

//set torus inner and outer radi to a 5th of the core's radius
glutSolidTorus(0.20*_alienRadius, 0.20*_alienRadius,60,60);

//Place second gate component at the bottom of the first component
glTranslatef(0,0,-0.25*_alienRadius);
glutSolidTorus(0.125*_alienRadius, 0.125*_alienRadius,60,60);

// return to initial coordinates at centre of core
glTranslatef(0,0,1.25*_alienRadius);
```

Finally, 4 more children nodes are created by calling alienLimb() with the _LimbAngle variable.

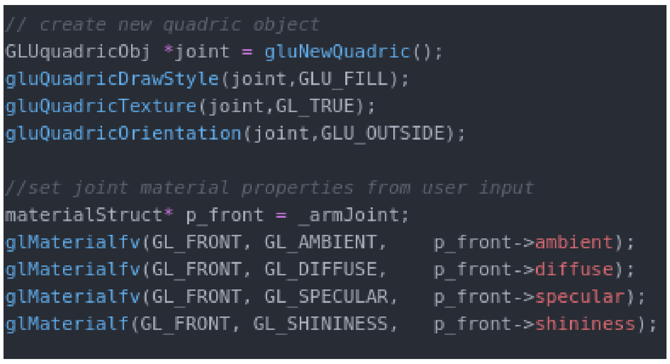


```
glScalef(_alienRadius,_alienRadius,_alienRadius);
// create 4 alien limbs using limbangle
alienLimb(_limbAngle,true);
alienLimb(-_limbAngle,false);
alienLimb(180-_limbAngle,true);
alienLimb(-180+_limbAngle,false);
```

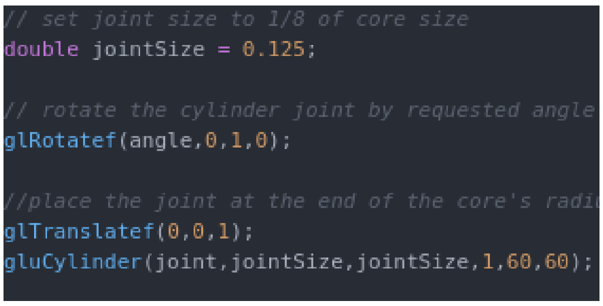
AlienLimb() function:

AlienLimb() takes a float angle value and a boolean value indicating whether the angle provided is clockwise or counter-clockwise as parameters. It then declares a sign variable and multiplies its value by -1 if the angle is anti-clockwise.

It creates a GLUquadricObject called joint and applies the material properties for cylinder joints from _armJoint . The cylinder joint size is set to one eighth of the alien core. The cylinder joint matrix is rotated by the requested angle on the y axis and translated to the end of the core’s radius at that angle. There it creates a gluCylinder with upper and lower radius the cylinder joint size and height the core’s radius.

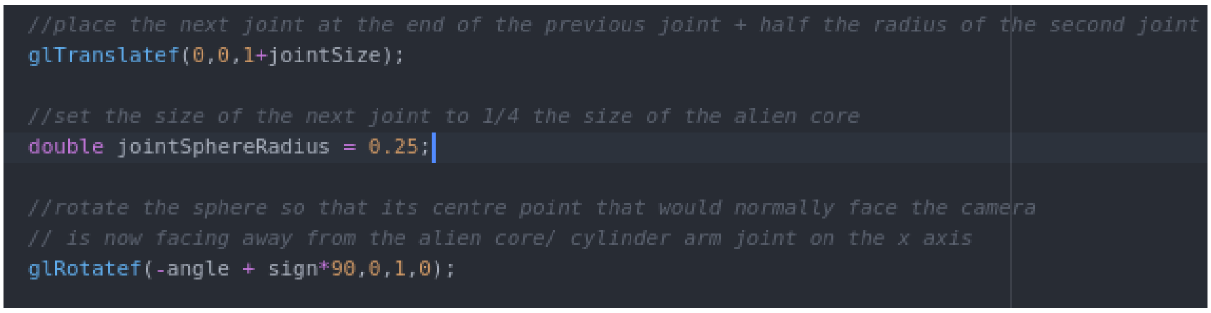


```
// create new quadric object
GLUquadricObj *joint = gluNewQuadric();
gluQuadricDrawStyle(joint,GLU_FILL);
gluQuadricTexture(joint,GL_TRUE);
gluQuadricOrientation(joint,GLU_OUTSIDE);

//set joint material properties from user input
materialStruct* p_front = _armJoint;
glMaterialfv(GL_FRONT, GL_AMBIENT,    p_front->ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE,    p_front->diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR,   p_front->specular);
glMaterialf(GL_FRONT, GL_SHININESS,   p_front->shininess);
```



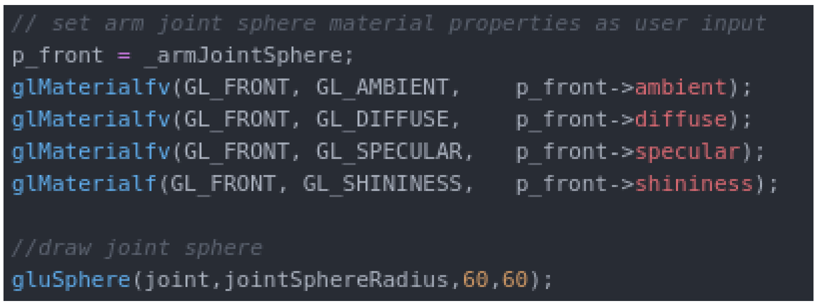
```
// set joint size to 1/8 of core size
double jointSize = 0.125;

// rotate the cylinder joint by requested angle
glRotatef(angle,0,1,0);

//place the joint at the end of the core's radi
glTranslatef(0,0,1);
gluCylinder(joint,jointSize,jointSize,1,60,60);
```

The next joint which is a sphere is placed at the end of the previous joint plus half its radius so that it is centred. The size of this joint is set to one fourth the size of the alien core radius. The coordinate system/matrix is then rotated further around the y axis such that its z direction is facing towards the positive or negative x axis of our initial coordinate system (depending on the angle) and the centre of the object to be placed at that position will be facing away from the alien, towards the x-axis instead of facing towards the camera.



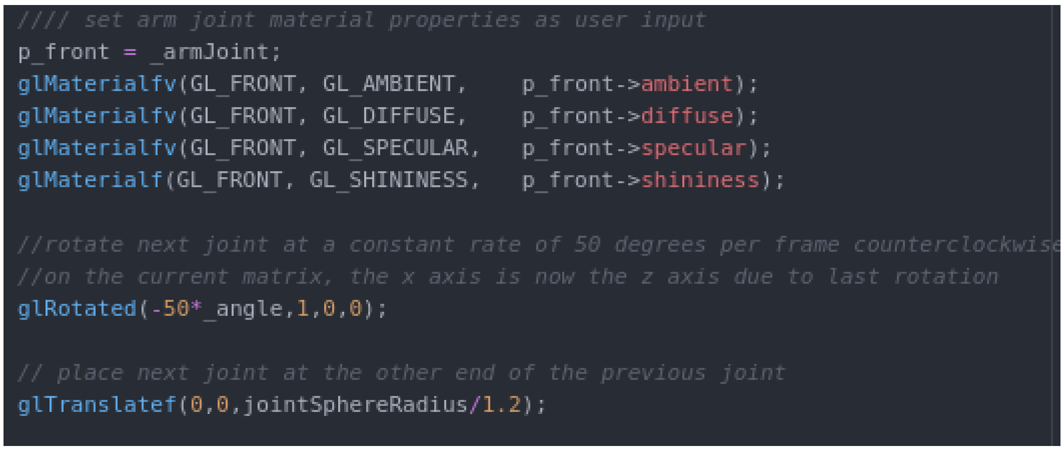
```
//place the next joint at the end of the previous joint + half the radius of the second joint
glTranslatef(0,0,1+jointSize);

//set the size of the next joint to 1/4 the size of the alien core
double jointSphereRadius = 0.25;

//rotate the sphere so that its centre point that would normally face the camera
// is now facing away from the alien core/ cylinder arm joint on the x axis
glRotatef(-angle + sign*90,0,1,0);
```

The sphere joint material properties are derived from _armJointSphere and applied to the front of the gluSphere created with radius a fourth of the radius of the alien's core.



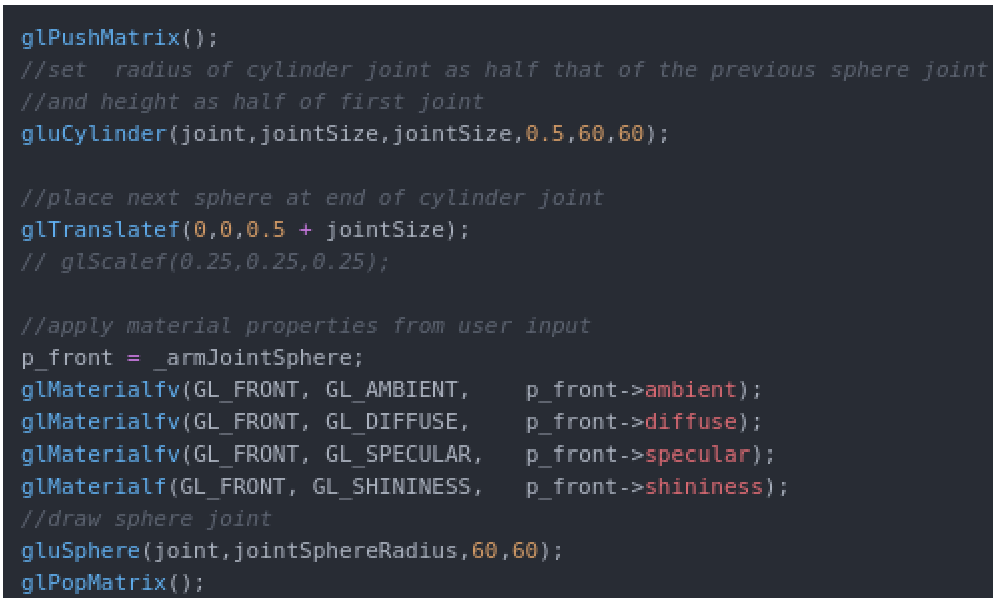
```
// set arm joint sphere material properties as user input
p_front = _armJointSphere;
glMaterialfv(GL_FRONT, GL_AMBIENT,    p_front->ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE,    p_front->diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR,   p_front->specular);
glMaterialf(GL_FRONT, GL_SHININESS,   p_front->shininess);

//draw joint sphere
gluSphere(joint,jointSphereRadius,60,60);
```

The front material properties are then re-set to the cylinder joints' and the next joint is rotated with a speed of 50 degrees/ms counter-clockwise around the x-axis which is the z-axis in our initial coordinate system. The cylinder is placed at the end of the sphere by translating on the z-axis which is now the x-axis by the Sphere Joint's Radius divided by 1.2 to accommodate for the fact that the next cylinder joint's height will be half the alien's core radius instead of 1.



```
//// set arm joint material properties as user input
p_front = _armJoint;
glMaterialfv(GL_FRONT, GL_AMBIENT,    p_front->ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE,    p_front->diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR,   p_front->specular);
glMaterialf(GL_FRONT, GL_SHININESS,   p_front->shininess);

//rotate next joint at a constant rate of 50 degrees per frame counterclockwise
//on the current matrix, the x axis is now the z axis due to last rotation
glRotated(-50*_angle,1,0,0);

// place next joint at the other end of the previous joint
glTranslatef(0,0,jointSphereRadius/1.2);
```

The cylinder is placed, the matrix translates to the end of the cylinder in the same way, reapplies the material properties of the sphere joint and creates a sphere at its end, reaching the leaf node of the tree for that branch.

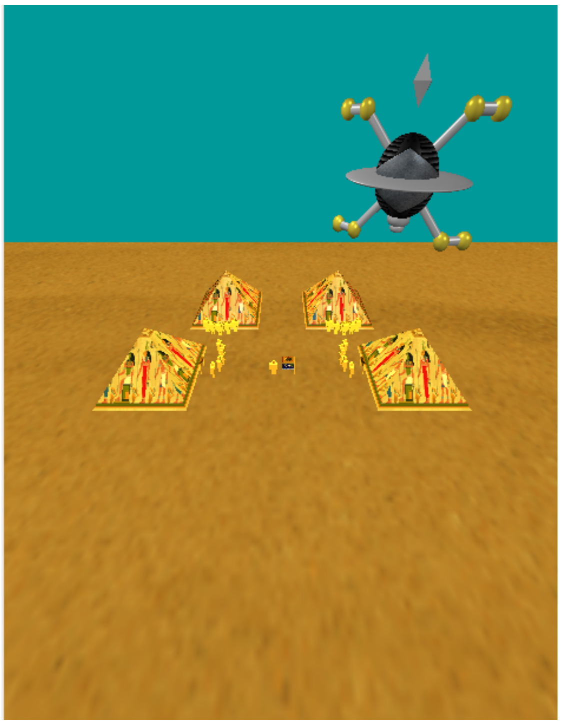


```
glPushMatrix();
//set  radius of cylinder joint as half that of the previous sphere joint
//and height as half of first joint
gluCylinder(joint,jointSize,jointSize,0.5,60,60);

//place next sphere at end of cylinder joint
glTranslatef(0,0,0.5 + jointSize);
// glScalef(0.25,0.25,0.25);

//apply material properties from user input
p_front = _armJointSphere;
glMaterialfv(GL_FRONT, GL_AMBIENT,    p_front->ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE,    p_front->diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR,   p_front->specular);
glMaterialf(GL_FRONT, GL_SHININESS,   p_front->shininess);
//draw sphere joint
gluSphere(joint,jointSphereRadius,60,60);
glPopMatrix();
```

The Limbangle() function is called with different angles, such that a moving limb appears on every quadrant of the sphere symmetrically. The limbs rotate around the sphere around the y axis through the function updateLimbAngle() that moves the angle of rotation of the alien limbs back and forth from 40 to 60 by incrementing/decrementing by 1 every time the screen is refreshed since it is called in updateAngle().

After the alien and its limbs are created, the identity matrix is loaded to reset everything and the scene generated is flushed to the screen.

Here is the final scene produced:



User Interface and interaction:

The user interface is created in an EgyptWindow widget whose class exists in EgyptWindow.cpp and EgyptWindow.h.. The entirety of the UI is created in its constructor.

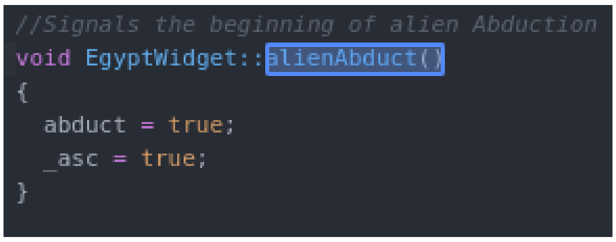
This is based on the OrthoDemoWindow class from the orthographic tutorial, and maintains the menubar, quit action and QBoxLayout. Nevertheless, in this implementation, the QBoxLayout's sequence is set to LeftToRIght so that widgets and layouts added to it are placed from left to right.

The constructor creates these, along with the egyptWidget, being the widget where the scene is to be rendered.

The User interface is comprised of two control boxes.

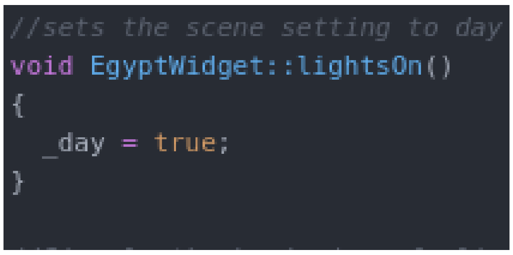
The first control box is the controls widget. The controls widget is created along with a QVBoxlayout and a QHBoxlayout and 3 QPushButtons. These 3 buttons are the abductbtn, the lightbtn,, and the darkbtn. The abductbtn is added straight to the QVBoxLayout, whereas the light and darkbtns are added to the QHBoxLayout and then that horizontal layout is added to the aforementioned QVBoxLayout.

The abductbtn is connected to the slot alienAbduct() that starts an abduction upon being clicked.



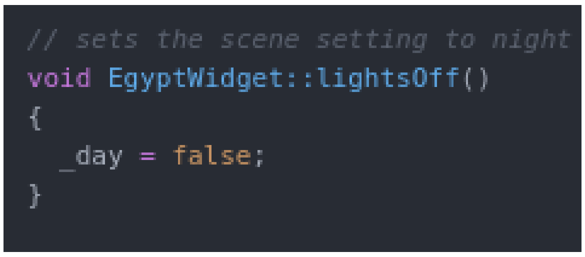
```
//Signals the beginning of alien Abduction
void EgyptWidget::alienAbduct()
{
  abduct = true;
  _asc = true;
}
```

The light button is connected to the lightsOn() slot that sets the value of day as true upon clicking and in return turning on GL_LIGHT0.



```
//sets the scene setting to day
void EgyptWidget::lightsOn()
{
  _day = true;
}
```

The dark button is connected to the lightsOff() slot that sets the value of day as false upon clicking and in turn turns off GL_LIGHT0.

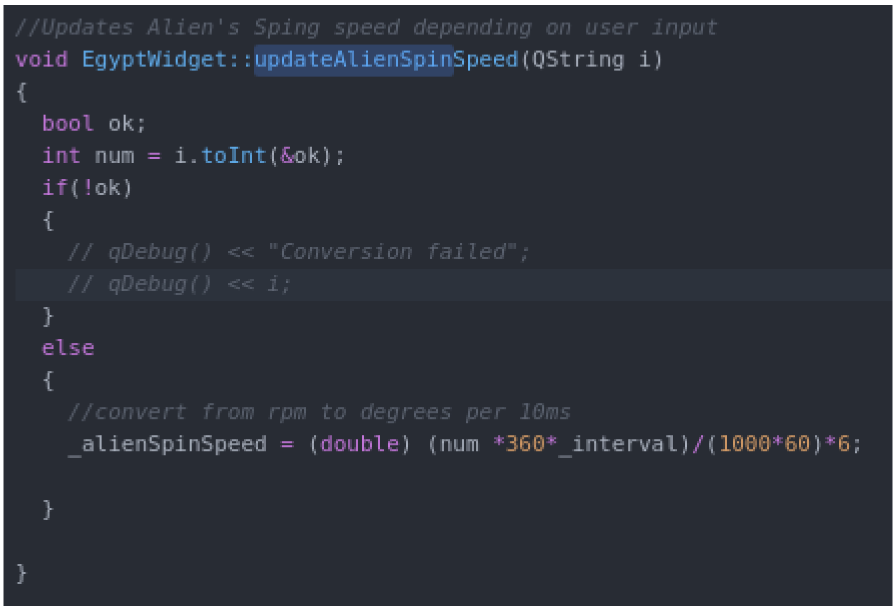


```
// sets the scene setting to night
void EgyptWidget::lightsOff()
{
  _day = false;
}
```

Then , a QFormLayout named alienForm is created. This contains 3 rows.

The first row of the form contains a label indicating that Alien spin speed must be input along with a QlineEdit named spinValue for user input. The same fixed width is set for all rows, as well as the corresponding placeholder text.

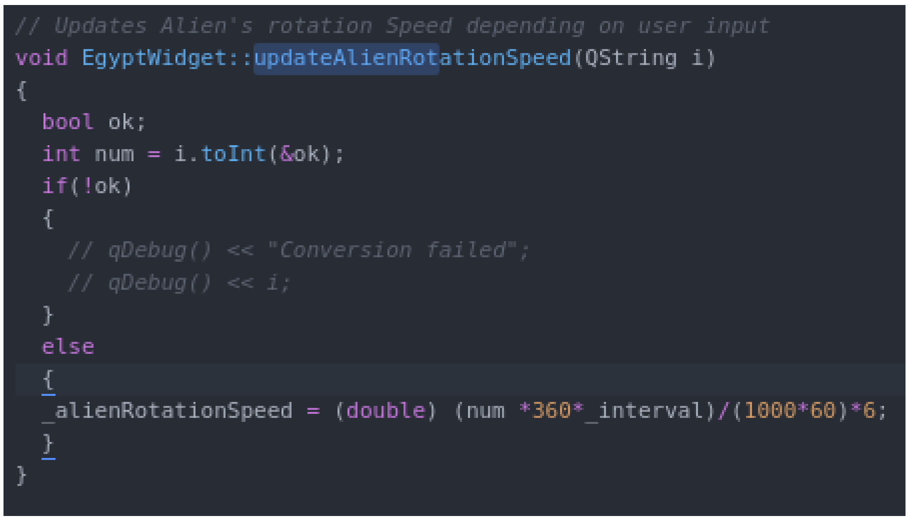
The spinVlaue QlineEdit   is connected to the egyptWidget's slot updateAlienSpinSpeed() that sets its _alienSpinSpeed variable to the cpnverted to revolutions per minute numeric text inside when the text is changed.



```
//Updates Alien's Sping speed depending on user input
void EgyptWidget::updateAlienSpinSpeed(QString i)
{
  bool ok;
  int num = i.toInt(&ok);
  if(!ok)
  {
    // qDebug() << "Conversion failed";
    // qDebug() << i;
  }
  else
  {
    //convert from rpm to degrees per 10ms
    _alienSpinSpeed = (double) (num *360*_interval)/(1000*60)*6;

  }

}
```

The third row contains a label indicating that alien rotation speed must be input, along with a qlineEdit named rotateValue for user input.

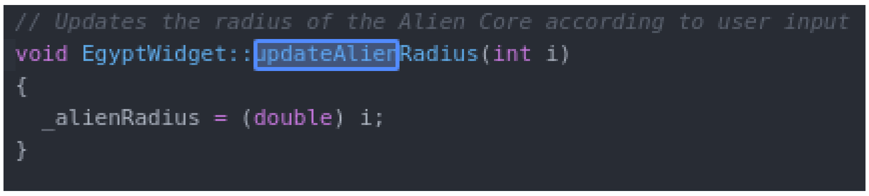
The rotateVlaue QlineEdit   is connected to the egyptWidget's slot updateAlienRotationSpeed() that sets its _alienSpinSpeed variable to the converted to revolutions per minute numeric text inside it when it is changed.



```
// Updates Alien's rotation Speed depending on user input
void EgyptWidget::updateAlienRotationSpeed(QString i)
{
  bool ok;
  int num = i.toInt(&ok);
  if(!ok)
  {
    // qDebug() << "Conversion failed";
    // qDebug() << i;
  }
  else
  {
  _alienRotationSpeed = (double) (num *360*_interval)/(1000*60)*6;
  }
}
```

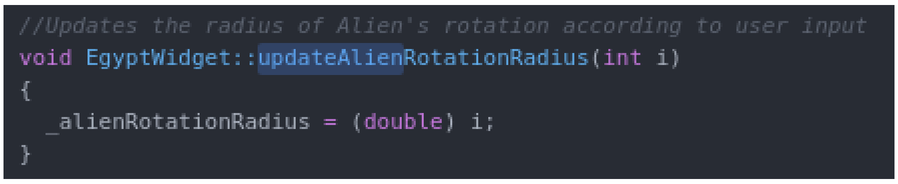
The constructor then creates 3 horizontal sliders along with their corresponding labels. AlienRtRadiusSlider for changing the rotation radius of the alien, alienSizeSlider for changing the radius of the core of the alien and the alienHeightSlider for changing the height at which the alien is placed.

The alienSizeSlider is connected to the egyptWidget's slot updateAlienRadius that sets the value of the _alienRadius variable to the Slider's when it is changed.



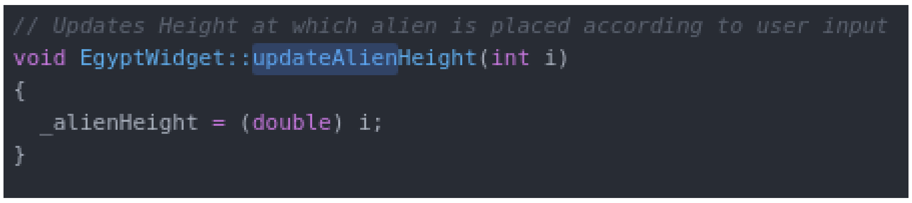
```
// Updates the radius of the Alien Core according to user input
void EgyptWidget::updateAlienRadius(int i)
{
  _alienRadius = (double) i;
}
```

The alienRtRadiusSlider is connected to the egyptWidget's updateAlienRotationRadius() slot that sets the _alienRotationRadius variable to the slider's value when it is changed.



```
//Updates the radius of Alien's rotation according to user input
void EgyptWidget::updateAlienRotationRadius(int i)
{
  _alienRotationRadius = (double) i;
}
```

The alienheightSlider is connected to the egyptWidget's updateAlienHeight() slot that sets its _alienHeight variable to the slider's value when it is changed.



```
// Updates Height at which alien is placed according to user input
void EgyptWidget::updateAlienHeight(int i)
{
  _alienHeight = (double) i;
}
```

The Sliders' maximum and minimum values are set to be reasonable within the scene's coordinates, along with their initial value set such that they are the same as the initialised value of the variables that they control in the EgyptWidget constructor.

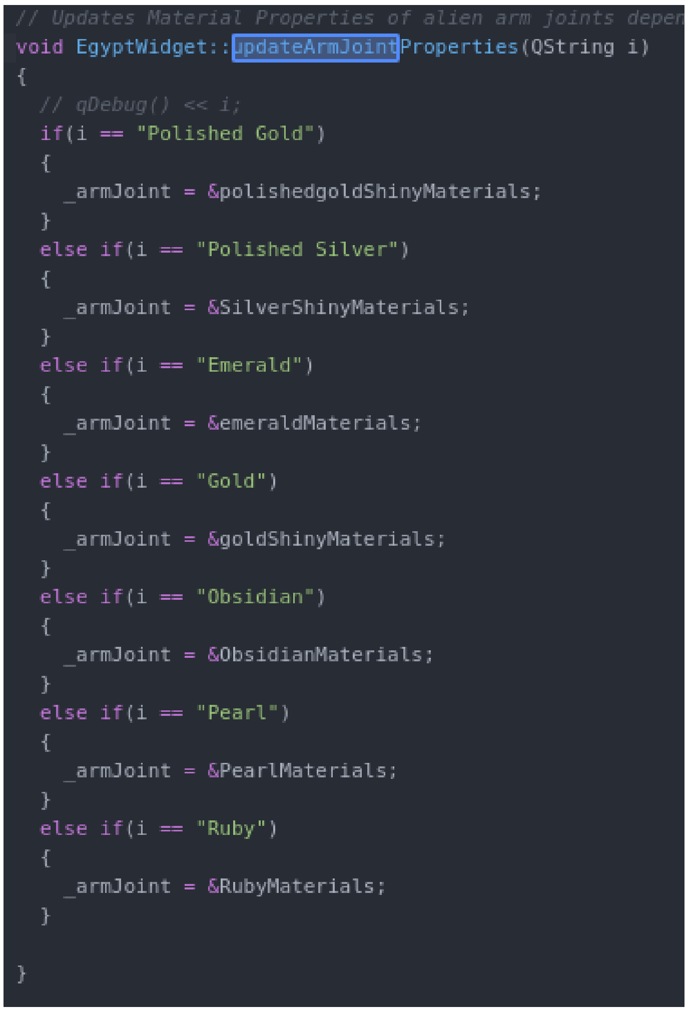
The sliders are then added to the alienForm along with their corresponding labels. Once these rows are added to the form, the form is then added to the QHBoxLayout alienLayout which is in turn added to the main QVLayout that is set as the layout of the controls widget.

Then, a QGroupBox., a QcomboBox, a QgridLayout and a Qlabel indicating that material properties in are displayed are created. The Combo box choices are set as a list of the material properties that can applied to the armjoints of the alien, and the current selection is set to the material properties of the joints as they are initialised in the EgyptWidget. The list involves the following material properties: "Emerald","Polished Silver","Gold","Polished Gold", "Obsidian", "Pearl", "Ruby".Then, the Label and combobox are added to the QgridLayout and the QgridLayout is set as the layout of the QGroupBox.

The same process is repeated to create combo boxes with the same material property selection for the arm joint sphere, the alien core, the alien disc and the abduct gate. The GroupBoxes are then added to the main QVLayout and that Layout is set as the layout of the controls widget. The controls widget's width is then set to be fixed at 300.

Each Qcombobox is connected to its equivalent slot that updates the material properties of the alien component's materialStruct to the selected option when the option is changed.

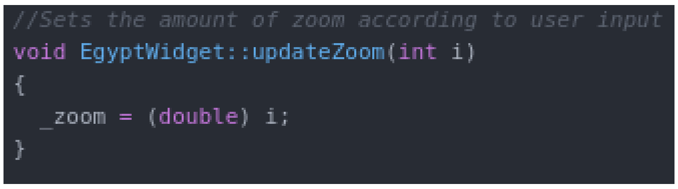
An example is updateArmJointProperties():



```
// Updates Material Properties of alien arm joints depen
void EgyptWidget::updateArmJointProperties(QString i)
{
  // qDebug() << i;
  if(i == "Polished Gold")
  {
    _armJoint = &polishedgoldShinyMaterials;
  }
  else if(i == "Polished Silver")
  {
    _armJoint = &SilverShinyMaterials;
  }
  else if(i == "Emerald")
  {
    _armJoint = &emeraldMaterials;
  }
  else if(i == "Gold")
  {
    _armJoint = &goldShinyMaterials;
  }
  else if(i == "Obsidian")
  {
    _armJoint = &ObsidianMaterials;
  }
  else if(i == "Pearl")
  {
    _armJoint = &PearlMaterials;
  }
  else if(i == "Ruby")
  {
    _armJoint = &RubyMaterials;
  }

}
```

After the first controls widget is complete, a controls2 QVBoxLayout is created. The EgyptWidget containing the scene is added to the controls2 layout. A zoom Qlabel and a zoom Qslider are created with the slider's max value being 95 so as to not decrease the field of view parameter of gluPerspective() in the EgyprWidget more than needed.

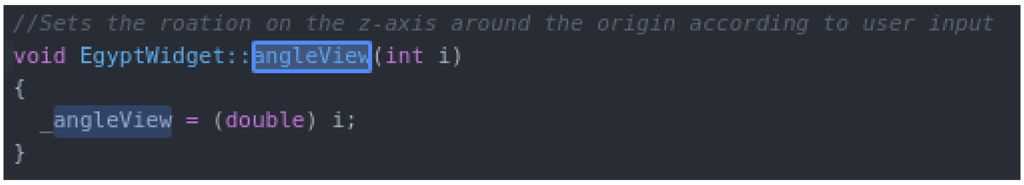
The zoomSlider is connected to the egyptWidget's slot updateZoom() that sets the _zoom variable's value to the slider's, effectively zooming in when the value increases.



```
//Sets the amount of zoom according to user input
void EgyptWidget::updateZoom(int i)
{
  _zoom = (double) i;
}
```

 A Qlabel indicating that the scene can be rotated and an angleViewSlider are also created to be used to rotate the scene, with its min value being 0 , and its max being 360 to allow for a full rotation.

The angleView Slider is connected to the egyptWidget's slot angleView(), that set the _angleView variable of the widget to the slider's, effectively rotating the scene.

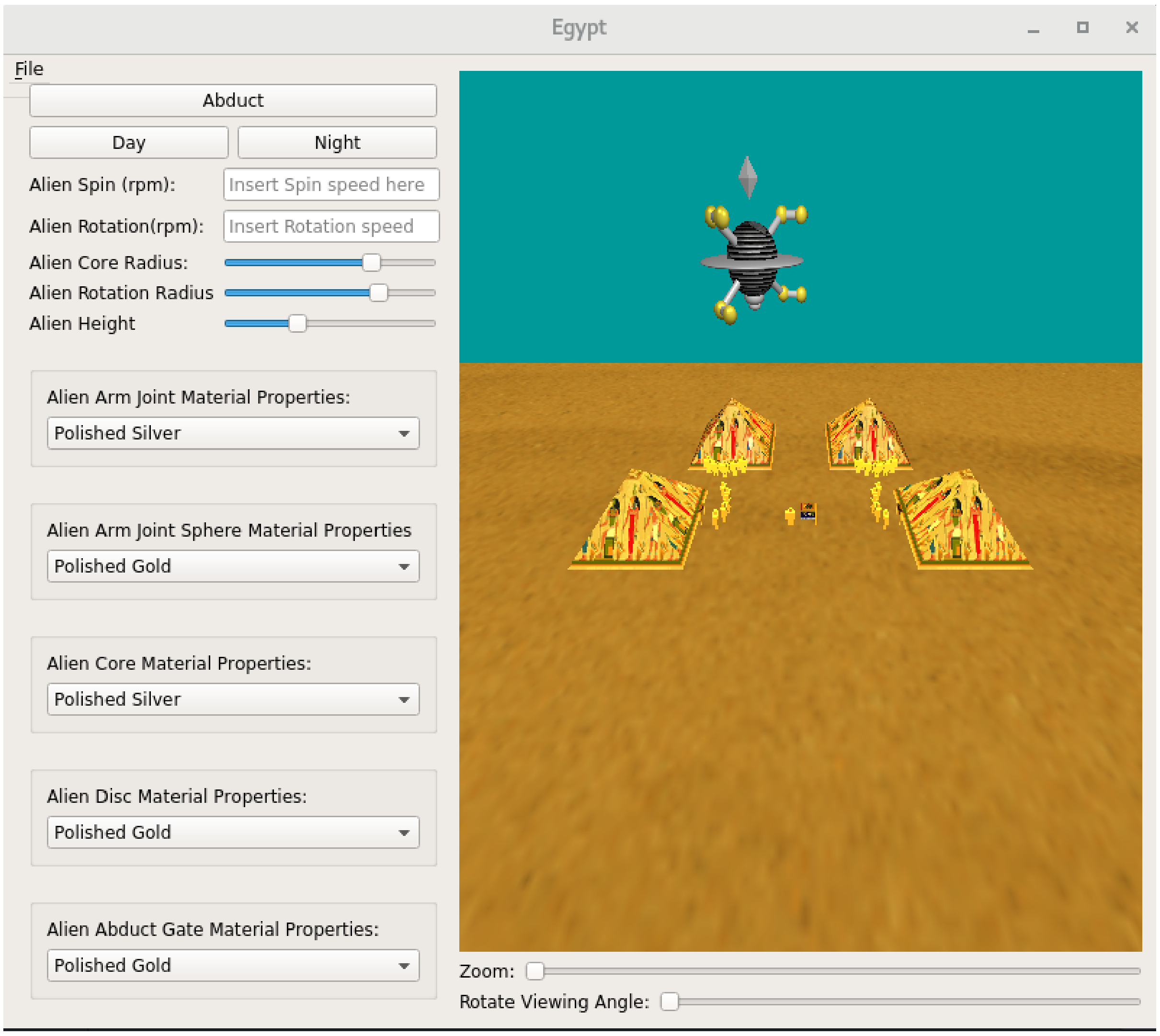


```
//Sets the roation on the z-axis around the origin according to user input
void EgyptWidget::angleView(int i)
{
  _angleView = (double) i;
}
```

Once these are created, each slider with its corresponding label are added to a different QHBoxLayout and then those 2 QHBoxLayouts are added to the controls2 layout, forming the final controls2 layout which is added to the parent widget windowLayout after the controls widget.

Finally, a Qtimer named otimer is created and is set to time out every 10 ms. It is connected to the EgyptWidget's updateAngle() slot and it increments the value of the _angle variable and applies the alien limb rotation animation by calling updateLimbAngle() every time it times out.

This is the final user interface produced:



Alien Arm Joint Material Properties: Polished Silver

Alien Arm Joint Sphere Material Properties Polished Gold

Alien Core Material Properties: Polished Silver

Alien Disc Material Properties: Polished Gold

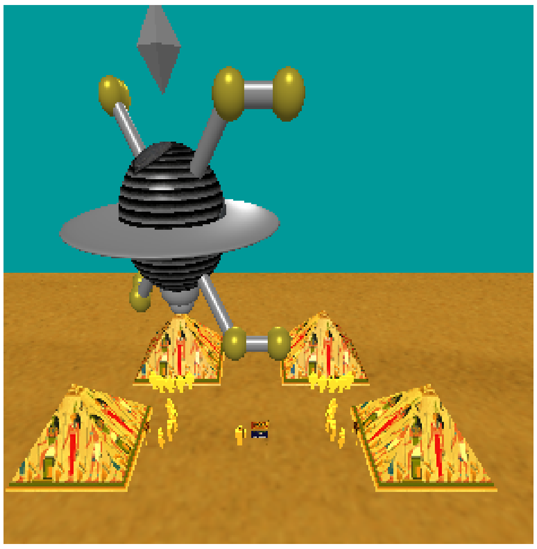
Alien Abduct Gate Material Properties: Polished Gold

Window setting:

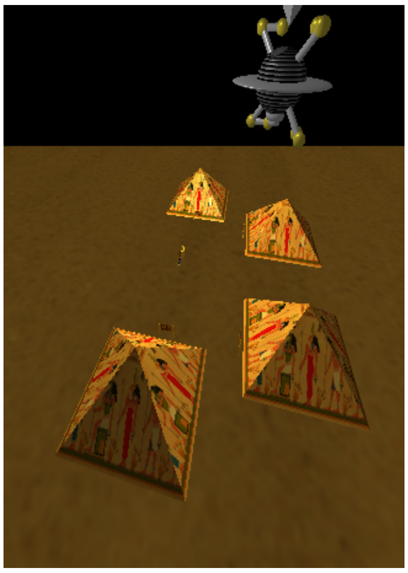
The EgyptWindow is created in EgyptMain.cpp. The window's resolution is set to 800 x600 to match the 4/3 aspect ratio of the EgyptWidget gluPerspective call.

Highlights:

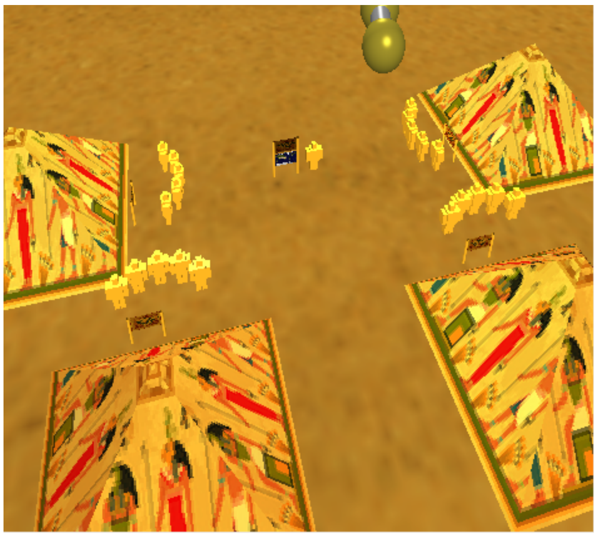
Specular lighting components:



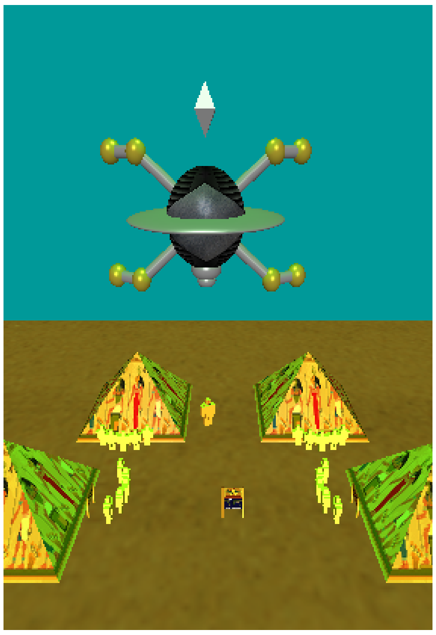
Diffuse lighting components:



Complex scene:

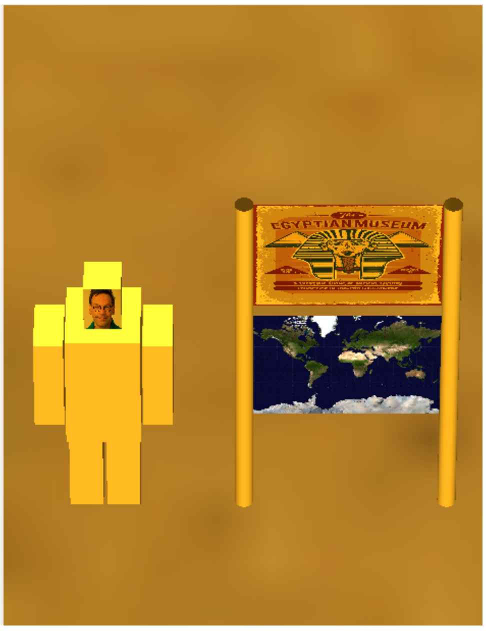


Animation:



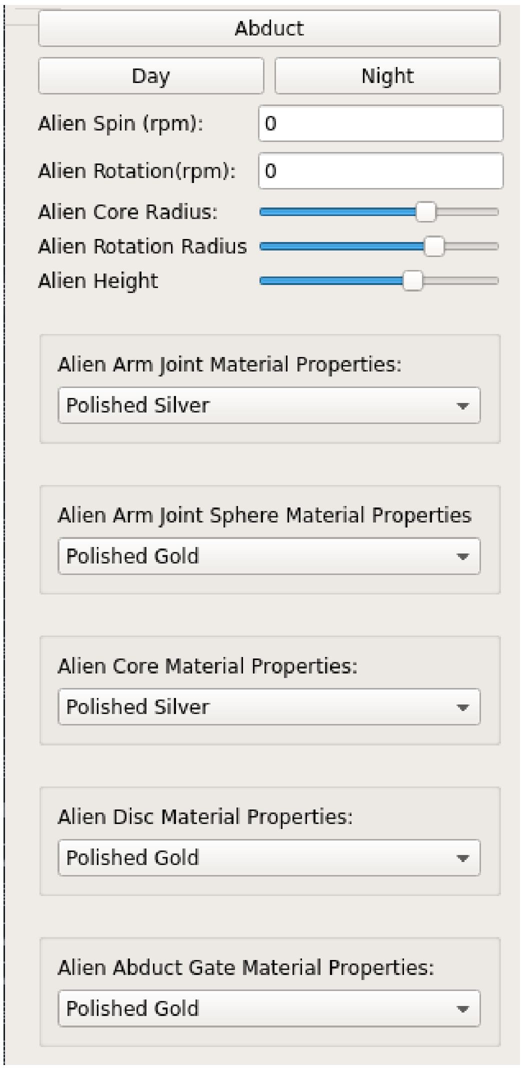
Convex object constructed from polygons:

Texture mapping:



Hierarchical Modelling Object with motion:

User Interaction:



Abduct

Day | Night

Alien Spin (rpm): 0

Alien Rotation(rpm): 0

Alien Core Radius:

Alien Rotation Radius

Alien Height

Alien Arm Joint Material Properties: Polished Silver

Alien Arm Joint Sphere Material Properties: Polished Gold

Alien Core Material Properties: Polished Silver

Alien Disc Material Properties: Polished Gold

Alien Abduct Gate Material Properties: Polished Gold

References:

sand.jpg:

<a href="https://www.freepik.com/photos/background">Background photo created by jannoon028 - www.freepik.com</a>

museum.jpg

<a href='https://www.freepik.com/vectors/poster'>Poster vector created by macrovector - www.freepik.com</a>

hieroglyphics.jpg:

<a href="https://www.freepik.com/vectors/background">Background vector created by macrovector - www.freepik.com</a>

ship.jpg

<a href="https://www.freepik.com/photos/background">Background photo created by kjpargeter - www.freepik.com</a>

Images downscaled for higher performance using:

https://convert-my-image.com/ImageConverter