Lefkos Savvides, December 2017

# Graphics Programming OPENGL

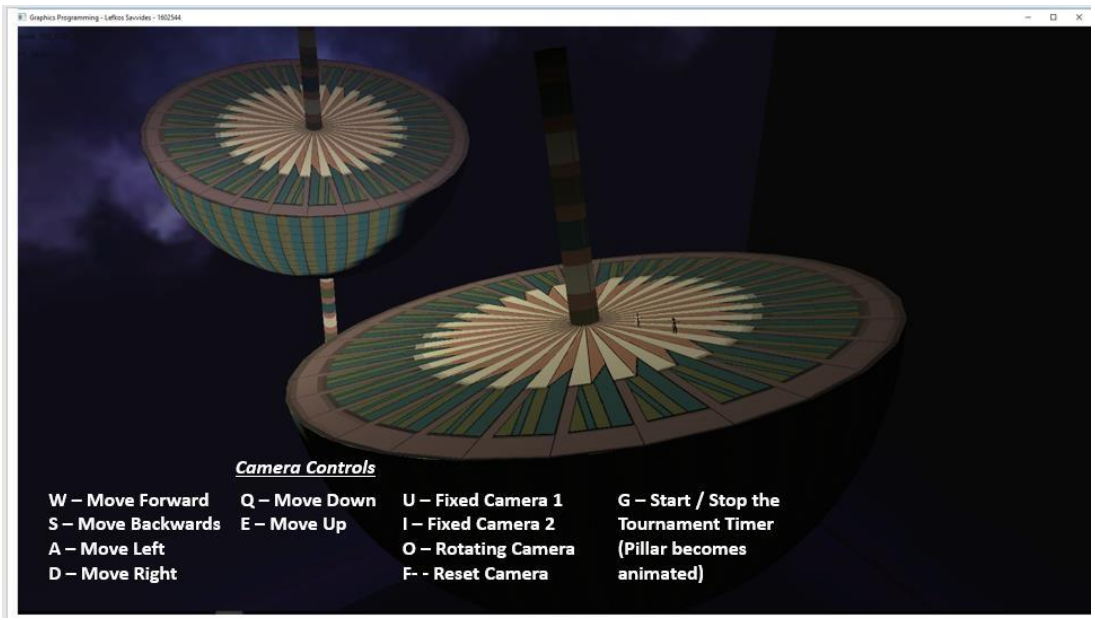
-

# Tournament of Power Arena

-

## Control Scheme

**(Caps lock needs to be turned off)**




**The camera rotates with mouse movement, non inverted.**

A working camera. The user must be able to manipulate the view through using the mouse and keyboard to control the camera. Additionally, you should provide multiple cameras each with a different focus such as limited controls, fixed views, procedurally controlled views or different camera types.

Contents of Camera.h file.

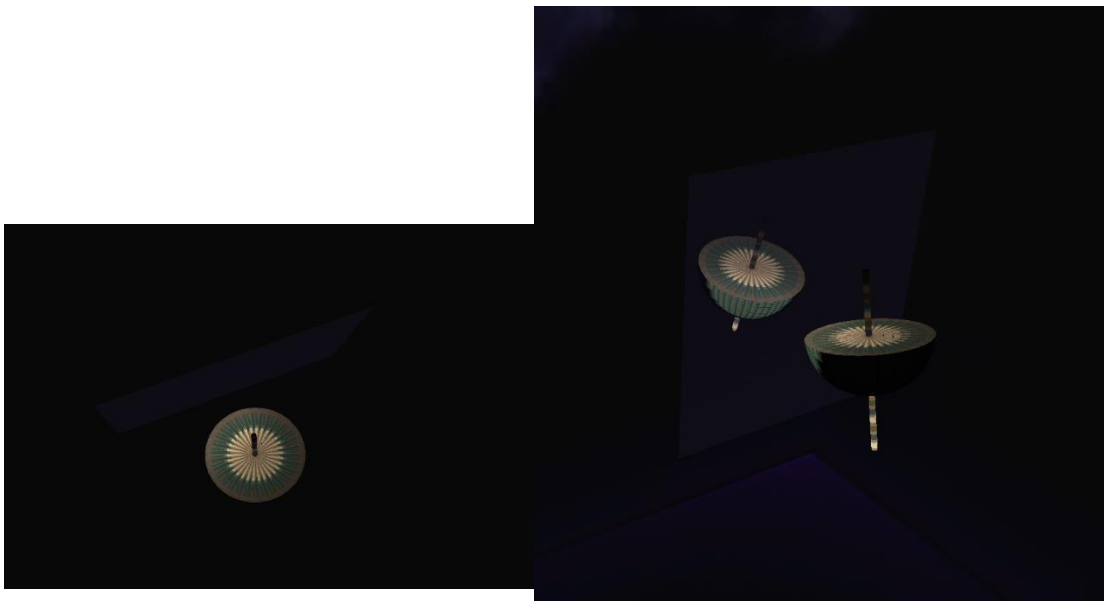
Explanation of functions in comment form in the Camera.cpp file.

## Camera 1 : Fixed Side Camera

```
    //Different Camera Functions
    void CameraReset();
    void FirstCamera();
    void SecondCamera();
    void ThirdCamera();

    //Getter Functions
    Vector3 getPosition();
    Vector3 getOrientation();
    Vector3 getLookAt();
    Vector3 getUp();
    Vector3 getForward();
    Vector3 getRight();


    //Vector Variables
    Vector3 Orientation, Position, LookAt, Up, Forward, Right;

    //Float Variables
    float Speed = 5;
};
```

```
    //Setter Functions
    void setPosition(Vector3 NewPosition);
    void setOrientation(Vector3 NewOrientation);
    void setLookAt(Vector3 NewLookAt);
    void setUp(Vector3 NewUp);
    void setForward(Vector3 NewForward);
    void setRight(Vector3 NewRight);

    void setXrotation(float x);
    void setYrotation(float y);
    void setZrotation(float z);

    //Movement Functions
    void moveForward(float dt);
    void moveBackwards(float dt);
    void moveRight(float dt);
    void moveLeft(float dt);
    void moveUp(float dt);
    void moveDown(float dt);

    //Different Camera Functions
    void CameraReset();
    void FirstCamera();
```

Meant to showcase the arena as a whole from a more topdown view while also showcasing the reflection. Main view that represents all the early shots form the anime version.



**Camera 2 : Fixed Topdown view**

Used to showcase the top part of the arena, the reflection is not in sight.

**Camera 3: Rotating upwards facing camera**

Used to showcase the whole scene in a broader scale while capturing the unlit parts of the arena and displaying that the reflection is lit in the correct place as well. Same hold true for the first camera as well.

```
//Camera Reset
if (input->isKeyDown('f'))
{
    CameraAnimation=0;
    glutWarpPointer(700, 810);
    camera->CameraReset();
    FixedCamera = false;
    AnimatedCamera = false;
    input->SetKeyUp('f');
}

//First Camera - Fixed
if (input->isKeyDown('u'))
{
    CameraAnimation = 0;
    camera->setXrotation(-1110);
    camera->setYrotation(300);
    camera->FirstCamera();
    FixedCamera = true;
    AnimatedCamera = false;
    input->SetKeyUp('u');
}
//Second Camera - Fixed
if (input->isKeyDown('i'))
{
    CameraAnimation = 0;
    camera->setXrotation(-1165);
    camera->setYrotation(650);
    camera->SecondCamera();
    FixedCamera = true;
    AnimatedCamera = false;
    input->SetKeyUp('i');
}
//Second Camera - Animated Camera
if (input->isKeyDown('o'))
{
    CameraAnimation = 0;
    FixedCamera = false;
    AnimatedCamera = true;
    camera->setXrotation(-1050);

    camera->ThirdCamera();
```

- The code for the 3 different cameras.

- They are all initialized in the .cpp file and called in Scene. The mouse position is set in scene which dictates the camera angle for those cameras.

- Different Boolean flags to dictate the selected camera.

**Lighting: (**The scene must show lighting from multiple lights of different types, colours and some animated.)

- Multiple Light Sources – I'm using a Directional Light enabled only to light up the Skybox and nothing else (Push Matrix ; glEnable(GL_LIGHT1) ; Render Skybox; glDisable(GL_LIGHT1);)

- In addition, I am using a Spotlight to Light up a part of the Arena.

- Due to the context of my scene, I decided those 2 lights were the only ones needed, and with them I am tackling the requirements of the brief
- 2 different lights, of different types.
- The skybox colour changes after the pillar in the middle reaches the bottom part and starts going back up again. This is a reference to the scene in the anime where the sky changes colour when half time is reached in the tournament.

```
//--------------------LIGHT SOURCES START--------------------


//SpotLight
GLfloat Light_Ambient_Spot[] = { 0.0f, 0.0f, 0.0f, 1.0f };
GLfloat Light_Diffuse_Spot[] = { 1.0f, 1.0f, 1.0f, 1.0f };
GLfloat Light_Position_Spot[] = { -10.0f, 0.0f, 6.0f, 1.0f };
GLfloat spot_Direction_Spot[] = { 0.0f, -1.0f, 0.0f };

glLightfv(GL_LIGHT0, GL_AMBIENT, Light_Ambient_Spot);
glLightfv(GL_LIGHT0, GL_DIFFUSE, Light_Diffuse_Spot);
glLightfv(GL_LIGHT0, GL_POSITION, Light_Position_Spot);
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 95.0f);
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spot_Direction_Spot);
glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 00.0);

//Directional Light (Enabled only for Skybox, illuminates the whole Sky)
GLfloat Light_Ambient_Dire[] = { 0.0f, 0.0f, 0.0f, 1.0f };
GLfloat Light_Diffuse_Dire[] = { 1.0f, 1.0f, 1.0f, 1.0f };
GLfloat Light_Position_Dire[] = { 1.0f, 1.0f, 1.0f, 0.0f };
GLfloat spot_Direction_Dire[] = { 1.0f, 1.0f, 1.0f };

glLightfv(GL_LIGHT1, GL_AMBIENT, Light_Ambient_Dire);
glLightfv(GL_LIGHT1, GL_POSITION, Light_Position_Dire);

//--------------------LIGHT SOURCES END--------------------
```

```
if (PillarFlag == false && EnableTimer == true)
{
    PillarTimer = PillarTimer - 0.01;
    GLfloat Light_Diffuse_Dire[] = { 1.0f, 1.0f, 1.0f, 1.0f };
    glLightfv(GL_LIGHT1, GL_DIFFUSE, Light_Diffuse_Dire);

}
if (PillarFlag == true && EnableTimer == true)
{
    PillarTimer = PillarTimer + 0.01;
    GLfloat Light_Diffuse_Dire_Green[] = { 0.0f, 1.0f, 0.0f, 1.0f };
    glLightfv(GL_LIGHT1, GL_DIFFUSE, Light_Diffuse_Dire_Green);
}
```

**Texturing: (**The scene must show use of texturing. Additionally, demonstrating texture filtering)

- All geometry rendered has textures loaded onto them.
- All models loaded in have textures accompanying them
- All procedurally generated geometry are mapped and have textures on them.
- Every single texture uses Trilinear Filtering for a consistent look.

- Both requirements are met, here's some screenshots of code used for texturing.

Loading in Textures

```
//Textures Loading
sky = SOIL_load_OGL_texture
(
    "gfx/Sky.png",
    SOIL_LOAD_AUTO,
    SOIL_CREATE_NEW_ID,
    SOIL_FLAG_MIPMAPS | SOIL_FLAG_NTSC_SAFE_RGB | SOIL_FLAG_COMPRESS_TO_
);

ArenaFloor = SOIL_load_OGL_texture
(
    "gfx/ArenaFloor.png",
    SOIL_LOAD_AUTO,
    SOIL_CREATE_NEW_ID,
    SOIL_FLAG_MIPMAPS | SOIL_FLAG_NTSC_SAFE_RGB | SOIL_FLAG_COMPRESS_TO_
);

Pillar = SOIL_load_OGL_texture
(
    "gfx/Pillar.png",
    SOIL_LOAD_AUTO,
```

Applying

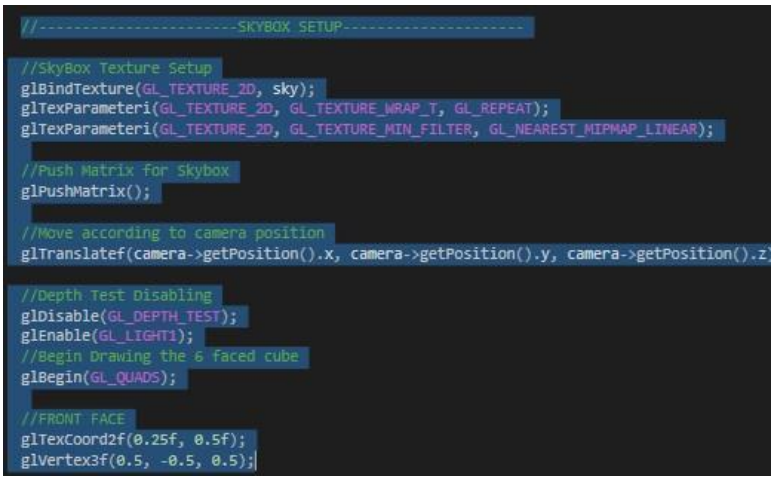
```
//Texture Setup for the Arena Floor (Disc)
glBindTexture(GL_TEXTURE_2D, ArenaFloor);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_LINEAR);
```

Vegeta Model with Texture that came along with it. The file provided was not a single texture file containing all required textures, that's why he looks like Pride from FMA. But the eyes are on point! I'm pretty sure this will be a new transformation down the line so I'm going to say that this is on point. /flex You can also see the ground having its texture.

## **Matrix Stack:**

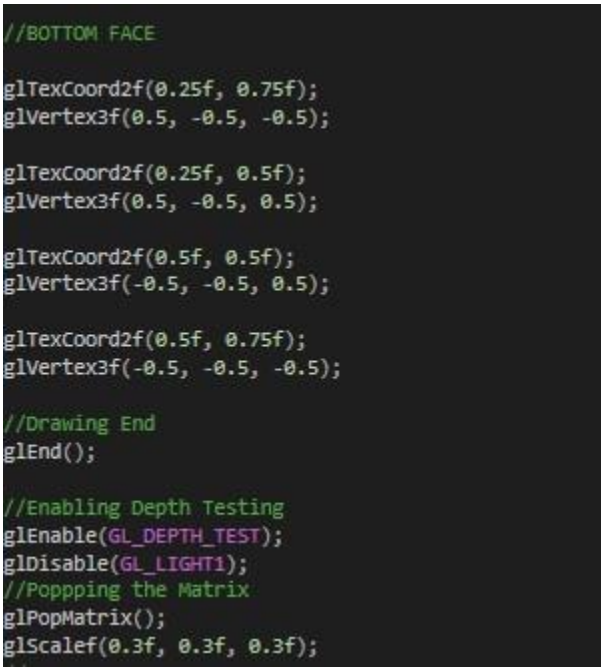
I've used the Matrix Stack throughout the whole of my program giving an initial rotation and tilt to the whole of the scene while afterwards animating the pillar in the middle by itself without having to affect the rest of the scene.

Other than that, the Matrix Stack was also used to create, render and animate the Skybox by having values and changes **ONLY** done to it and not the rest of the scene. Many more instances



```
//----------------------SKYBOX SETUP---------------------

//SkyBox Texture Setup
glBindTexture(GL_TEXTURE_2D, sky);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_LINEAR);

//Push Matrix for Skybox
glPushMatrix();

//Move according to camera position
glTranslatef(camera->getPosition().x, camera->getPosition().y, camera->getPosition().z)

//Depth Test Disabling
glDisable(GL_DEPTH_TEST);
glEnable(GL_LIGHT1);
//Begin Drawing the 6 faced cube
glBegin(GL_QUADS);

//FRONT FACE
glTexCoord2f(0.25f, 0.5f);
glVertex3f(0.5, -0.5, 0.5);
```

can be found within the code where the Matrix Stack is used.

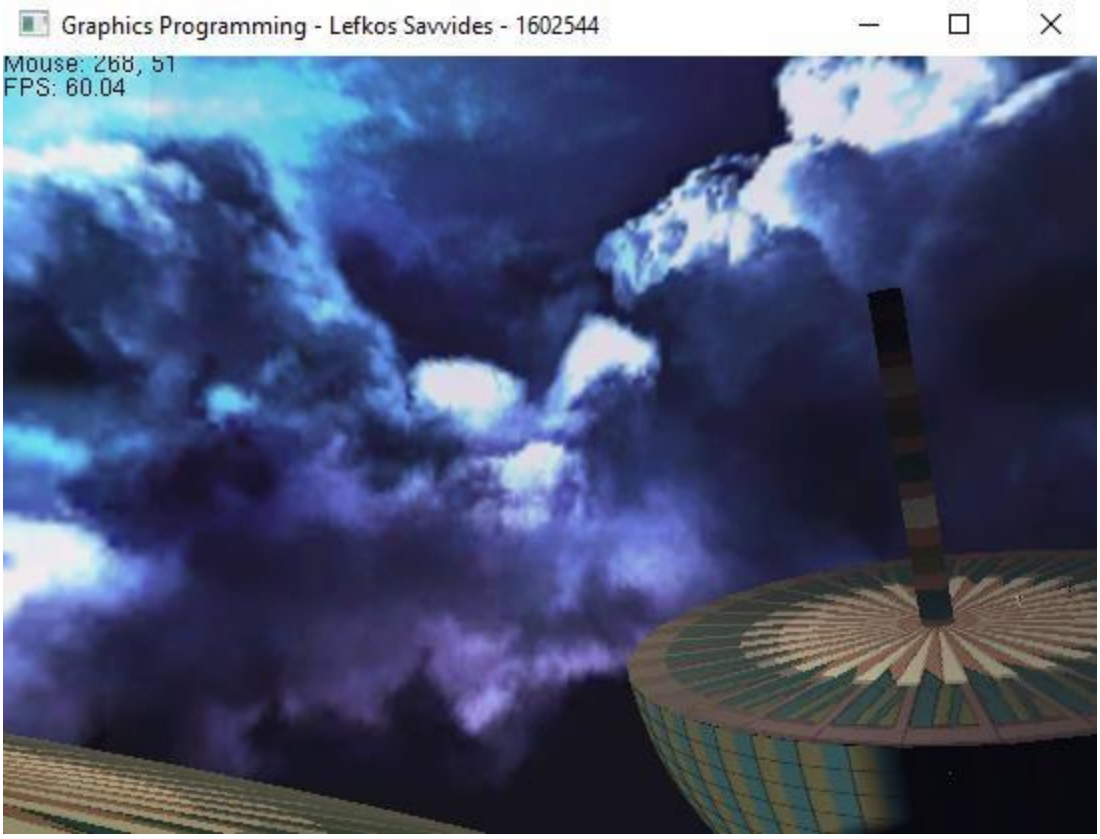


```
//BOTTOM FACE

glTexCoord2f(0.25f, 0.75f);
glVertex3f(0.5, -0.5, -0.5);

glTexCoord2f(0.25f, 0.5f);
glVertex3f(0.5, -0.5, 0.5);

glTexCoord2f(0.5f, 0.5f);
glVertex3f(-0.5, -0.5, 0.5);

glTexCoord2f(0.5f, 0.75f);
glVertex3f(-0.5, -0.5, -0.5);

//Drawing End
glEnd();

//Enabling Depth Testing
glEnable(GL_DEPTH_TEST);
glDisable(GL_LIGHT1);
//Poppping the Matrix
glPopMatrix();
glScalef(0.3f, 0.3f, 0.3f);
```

## Code for the Cylinder Animation

```
//Pushing the Matrix
	glPushMatrix();

	//Cylinder Texture
	glBindTexture(GL_TEXTURE_2D, Pillar);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_LINEAR);
	//Translating the Cylinder downwards so it pierces the Stage
glTranslatef(0.0f, PillarTimer, 0.0f);
	//Rotates the Cylinder so it's Vertical on the Stage
	GLfloat Light_Position_SpotS[] = { -10.0f, 0.0f, 6.0f, 1.0f };
  glRotatef(-90, 1.0f, 0.0f, 0.0f) //Function to Generate the Cylinder
ShapeGeneration.GenerateCylinder(1, 60); glPopMatrix();
```

## Transparency Effect / Alpha Blending / Stencil Buffer:



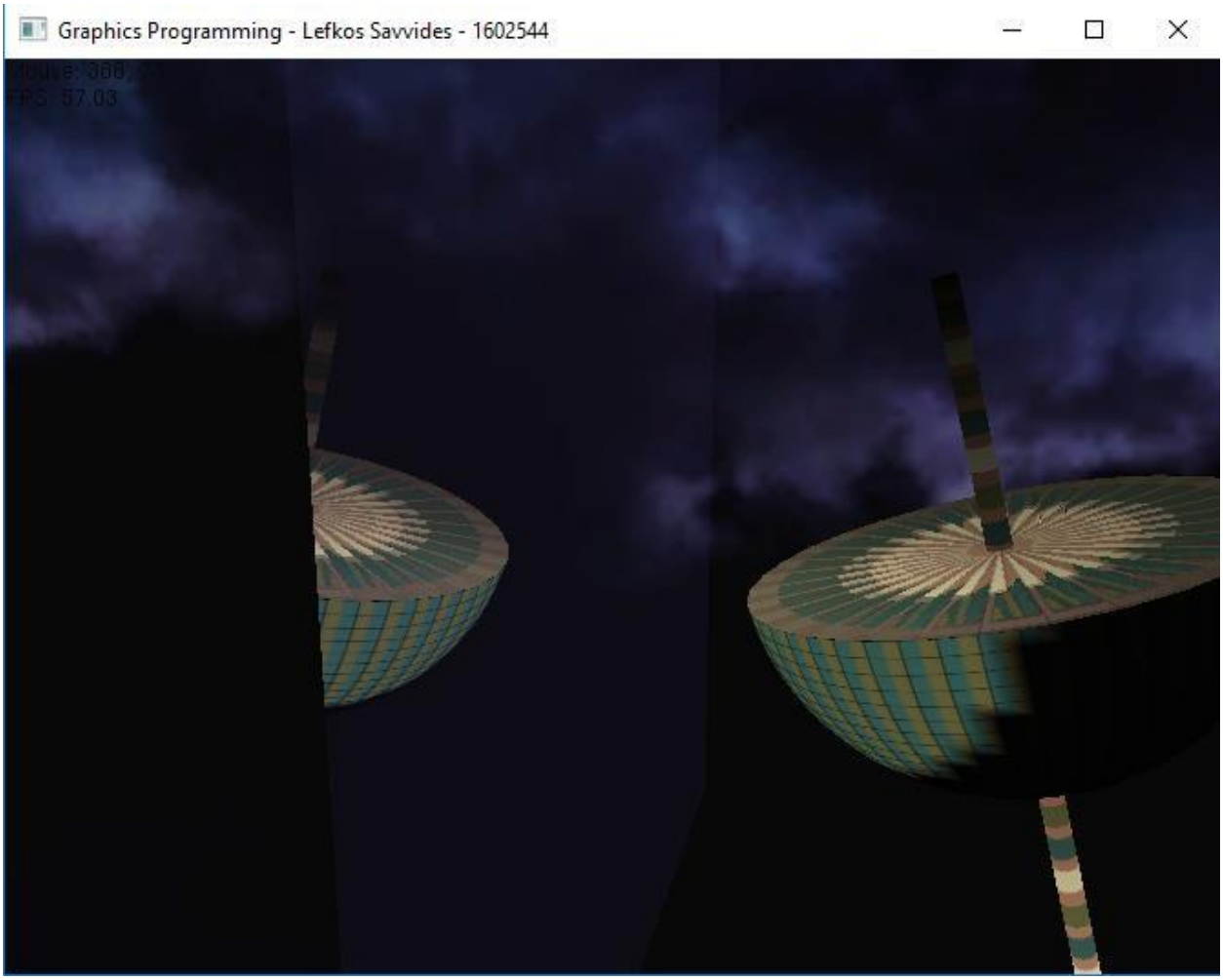

**Transparency** has been used along with **the Stencil Buffer** in order to create a mirror that **reflects** the Tournament of Power Arena Stage in all of it's entirety.

By creating a wall (Quad) with an Alpha value of 0.1, and Blending enabled where I created my Stencil, I have a surface that shows what's behind it, the second arena I am rendering.

```
//Disables the Stencil Test
glDisable(GL_STENCIL_TEST);
//Enables Blending
glEnable(GL_BLEND);
//Disables Lighting and sets the Colour for the Wall (DarkSlateBlue)
glDisable(GL_LIGHTING);

glColor4f(0.282f, 0.239f, 0.545f, 0.1f);
//Pushes the Matrix
glPushMatrix();

//Chooses where to draw the Wall
glTranslatef(-30, 0, 0);
//Rotates the Wall so it's Vertical
glRotatef(90, 0, 1, 0);

//Starts drawing the Wall
glBegin(GL_QUADS);

glVertex3f(-50, 50, 0);

glVertex3f(-50, -50, 0);

glVertex3f(50, -50, 0);
```

```
glVertex3f(50, 50, 0);
//Stops drawing the Wall
glEnd();
//Pops the Matrix
glPopMatrix();

//Enables Lighting
glEnable(GL_LIGHTING);
//Disables Blending
glDisable(GL_BLEND);
//Pushes the Matrix
glPushMatrix();
```

Using the Stencil, it creates a realistic mirror, and so when the camera moves around the Arena is not visible behind the "mirror".




**Vertex Arrays/ Procedural Generation:**

As per requested, Vertex Arrays are not only used in Model Loading. They've been used in Procedural Generation as well for a more clean and optimized coding experience. Example screenshots include my render function for Procedural Generation as well as the calculation of the First Calculation in my for loop for the generation of a Hemisphere.

```
void ProceduralGeneration::Render()
{
    //Enabling the Use of Arrays to read from them.
    glEnableClientState(GL_VERTEX_ARRAY);
    glEnableClientState(GL_NORMAL_ARRAY);
    glEnableClientState(GL_TEXTURE_COORD_ARRAY);

    //Binds the loaded in Texture (The base for the Arena)
    glBindTexture(GL_TEXTURE_2D, ArenaBase);

    //Specify which arrays will be used to Render the Model
    glVertexPointer(3, GL_FLOAT, 0, Vertex.data());
    glNormalPointer(GL_FLOAT, 0, Normals.data());
    glTexCoordPointer(2, GL_FLOAT, 0, TexCoords.data());

    //Function used to Draw the Shape
    glDrawArrays(GL_QUADS, 0, m_VertexCount);

    //Disable the Use of Arrays to read from them.
    glDisableClientState(GL_VERTEX_ARRAY);
    glDisableClientState(GL_NORMAL_ARRAY);
    glDisableClientState(GL_TEXTURE_COORD_ARRAY);

    //Remove Texture Data from being binded
    glBindTexture(GL_TEXTURE_2D, NULL);
}
```

```
//-------------------------First--------------------------

//Vertex
Vertex.push_back(Radius * cos(angle_lati) * sin(next_long));
Vertex.push_back(Radius * cos(next_long));
Vertex.push_back(Radius * sin(angle_lati) * sin(next_long));

//Normals
Normals.push_back(Radius * cos(angle_lati) * sin(next_long));
Normals.push_back(Radius * cos(next_long));
Normals.push_back(Radius * sin(angle_lati) * sin(next_long));

//Texture Coordinates
TexCoords.push_back(0.0f);
TexCoords.push_back(1.0f);

//-------------------------Second-------------------------
```

The whole of my Arena is composed of Procedurally generated shaped, of which include a Hemisphere, a Disc and a Cylinder.

- The Hemisphere is used as a base.
- The Disc is used at the Arena floor.
- And the Cylinder is them middle piece.

**By looking at the arena you can see the three different Shapes generated and differentiate them with ease.**

## Function to Generate Cylinder:

```
void ProceduralGeneration::GenerateCylinder(int r, int seg)
{
	//Variable Initialization
	Radius = r;
	Segments = seg;

	//Float Variables
	float x = 0, y = 0, Angle = 0, Angle_StepSize = 0.1;

	//Draws the Tube
glBegin(GL_QUAD_STRIP);
```

```
	//Angle initialization
	Angle = 0.0;

	//Set TexCoordinates
	glTexCoord2f(0.0f, 1.0f);

	//Loop for coordinates calculation
	while (Angle < 2 * M_PI)
	{
		//Calculates x/y coordinates for Vertex
data		x = Radius * cos(Angle);		y =
Radius * sin(Angle);


		//---------FIRST FACE----------
glNormal3f(0.0f, 0.0f, 1.0f);		glTexCoord2f(1.0f,
1.0f);
		glVertex3f(x, y, Segments);

		//---------SECOND FACE---------
glNormal3f(0.0f, 0.0f, 1.0f);		glTexCoord2f(1.0f,
0.0f);
		glVertex3f(x, y, 0.0);

		//Calculates the next Angle
		Angle = Angle + Angle_StepSize;
	}

	//---------THIRD FACE----------
glNormal3f(0.0f, 0.0f, 1.0f);
glTexCoord2f(0.0f, 1.0f);   glVertex3f(Radius, 0.0,
Segments);

	//---------FOURTH FACE----------
glNormal3f(0.0f, 0.0f, 1.0f);
glTexCoord2f(0.0f, 0.0f);   glVertex3f(Radius, 0.0,
0.0);

	glEnd();

}
```

**Due to the size of the other two, I won't include a screenshot or the code here, but there are both in the ProceduralGeneration.cpp file along with comments explaining the different parts.**

**All three shapes have Texture Coordinates and Normals applied to them for Lighting and Texturing as well.**

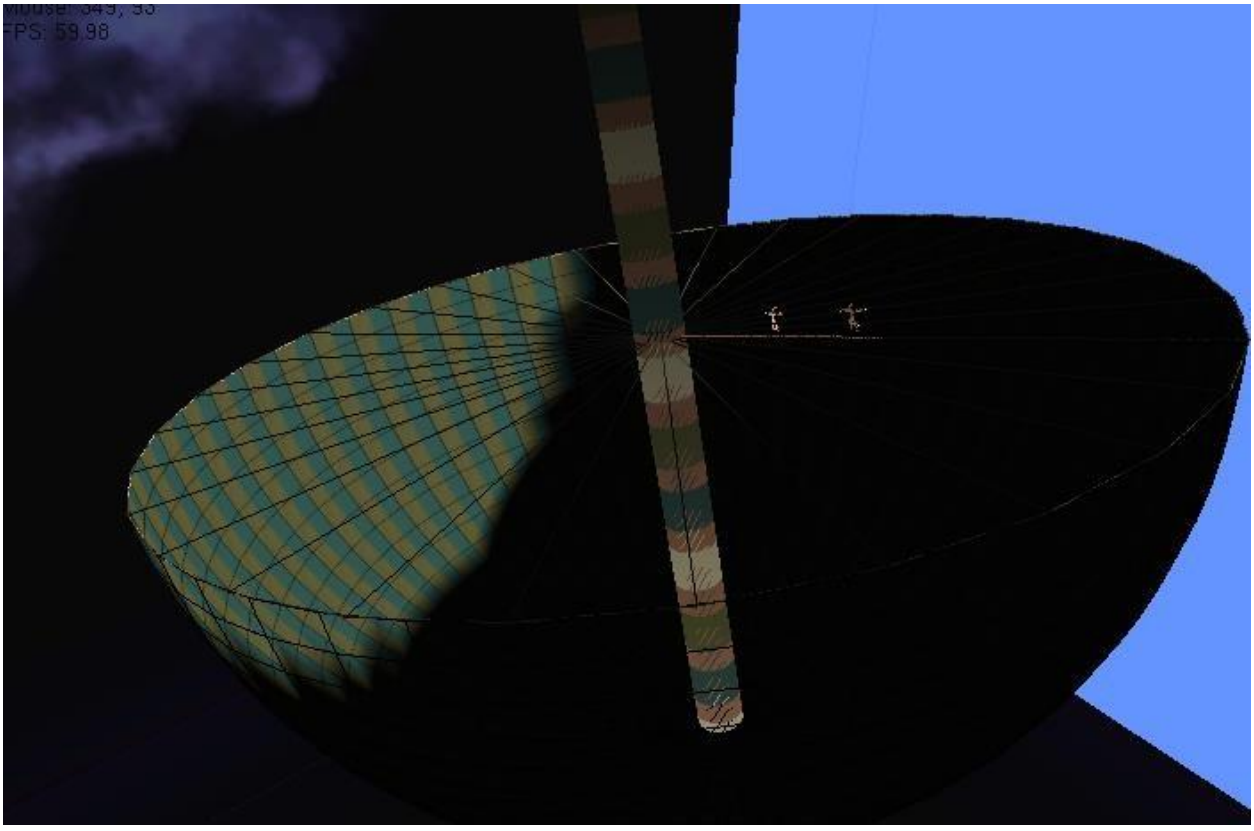
## User Interaction:

When the user presses “g” the pillar in the middle starts/ stops moving.

That dictates the colour sky, position of the pillar and the direction of movement. When it reaches its bottom position it starts going back up.

## Wireframe Mode:

By pressing “r” on the keyboard, wireframe mode is toggled on / off.



## References:

**Idea inspired by the Tournement of Power Arc of Dragonball Super, I recreated the fighting stage.**

Skybox used is the image provided in our module lab exercises but recoloured to match the referenced photo sky colour.

Textures for Arena/ Pillar are created in MSPaint.

Music is “Golden Frieza” theme from the Dragonball Super OST.

Vegeta model and texture from Budokai Tenkaichi 3, found online.

Goku model and texture from http://www.domawe.net/2014/10/gokudragonball-3d-model-free-download.html



**The project was created as part of a second year University Project.**