OpenGL

OpenGL : The origin

In the 80's, developing software for a wide range of graphic hardware was a pain. There weren't any APIs.

In the early 90's, Silicon Graphics Inc. (SGI) was the leader in 3D graphics and had developed their own API to render them more easily: *IRIS GL*.

OpenGL : The origin

Due to hard competition and IrisGL growing too fast, they decided to make IrisGL into an Open Graphics Library.

An Open GL ARB (Architectural Review Board) was created to establish and maintain its software specifications.

Today, Khronos group is responsible of OpenGL maintenance.

OpenGL support

Windows
Mac
Linux
PS3
PS4
Wii
iPhone
Android
DS
PSP
Wii
Switch
etc

Pretty much every decent gaming device

✓

XBox

Direct X

✗

New (3D) perspective

- ✗ We still will use SDL, but now we draw onto a OpenGL context
- ✗ Immediate Mode will be used for drawing.
  - ✗ Lets forget about retained mode… that's for "Game engines".
- ✗ Render module will adapt to OpenGL (see *ModuleRenderer3D* class)
- ✗ A Camera module will be necessary (see *ModuleCamera3D* class)
- ✗ The racing game will be formed by simple primitives (see *Primitives* class)
- ✗ The math library will include more functionality

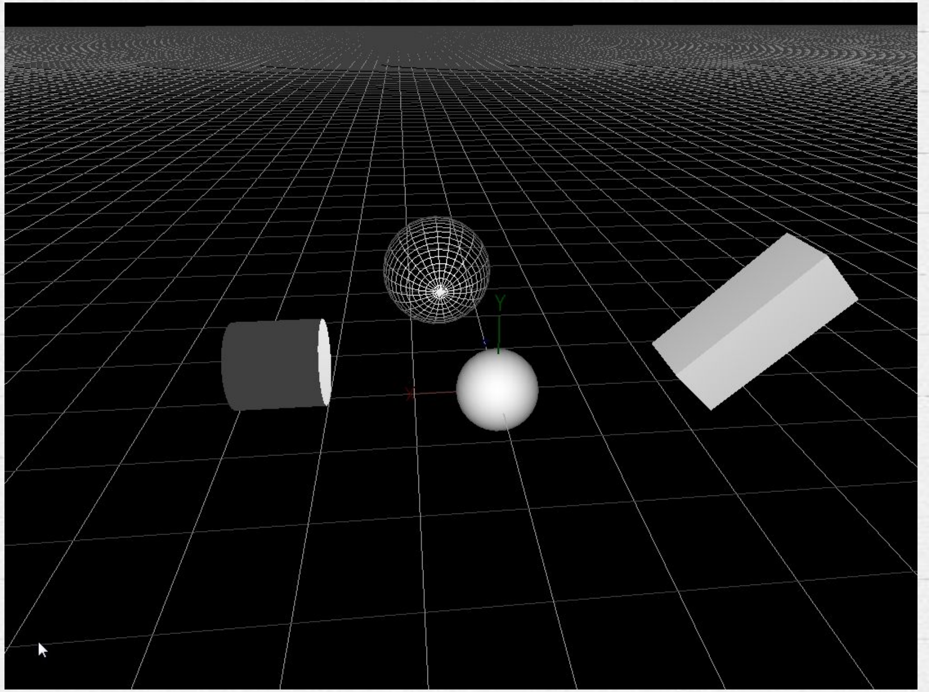
What you already have

- ✗ OpenGL context initialized
- ✗ Some lights illuminating the scene
- ✗ Some primitives ready to Render (Line, Plane, Cube, Sphere, Cylinder)
    - ✗ See Primitive.h
- ✗ Updated Camera module!
- ✗ Extended math library (glmath.h)

What you will have

By the end of the class

Hopefully



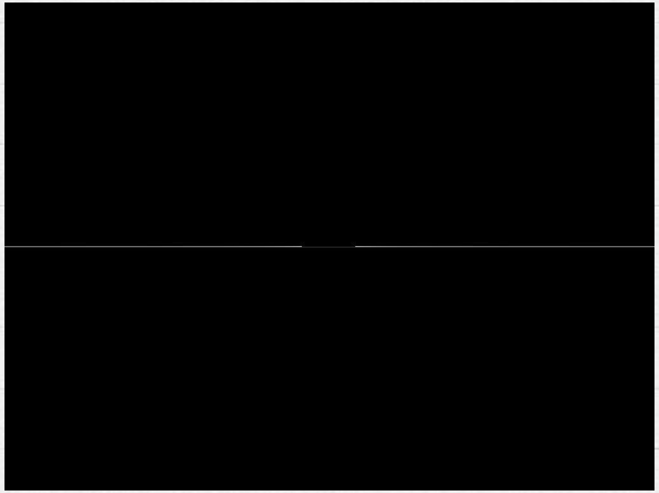
YOUR TURN !

TODO 1

```
// TODO 1: Create a Plane primitive. This uses the plane formula
// so you have to express the normal of the plane to create
// a plane centered around 0,0. Make that it draw the axis for reference
```

- ✗ Theoretically, a plane is infinite, but drawing infinite lines isn't a good plan.
- ✗ Don't forget to call Render()
- ✗ Enable "axis" as a reference

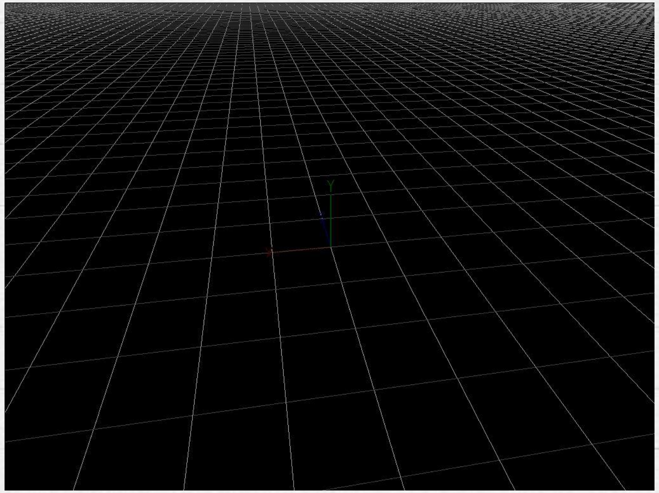
Result:



TODO 2

```
// TODO 2: Place the camera one unit up in Y and one unit to the right
// experiment with different camera placements, then use LookAt()
// to make it look at the center
```

Result:

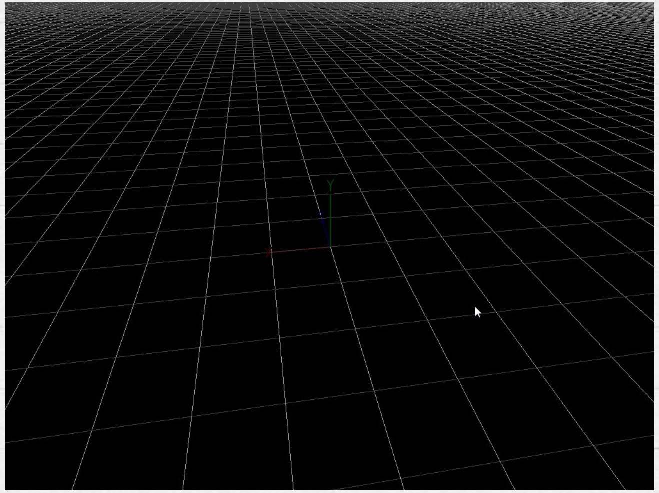


- ✗ Here's where having the axis will come in handy
- ✗ Set Position
- ✗ Call LookAt(vec3) to rotate the camera easily

TODO 3

```
// TODO 3: Make the camera go up/down when pressing R (up) F(down)
```

Result:

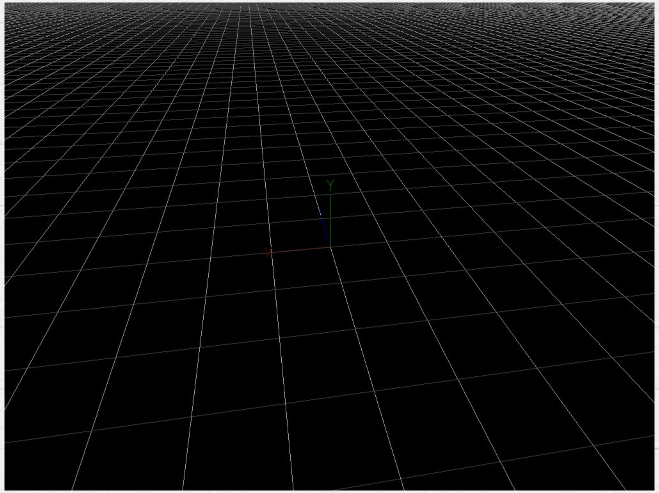


- ✗ Use **KEY_REPEAT** to add small increments to the camera position
- ✗ Around **0.05f** per frame should do it
- ✗ Update **Reference** as well (for the future)

TODO 4

```
// TODO 4: Make the camera go forward (w) and backward with (s)
// Note that the vectors X/Y/Z contain the current axis of the camera
// you can read them to modify Position
```

Result:



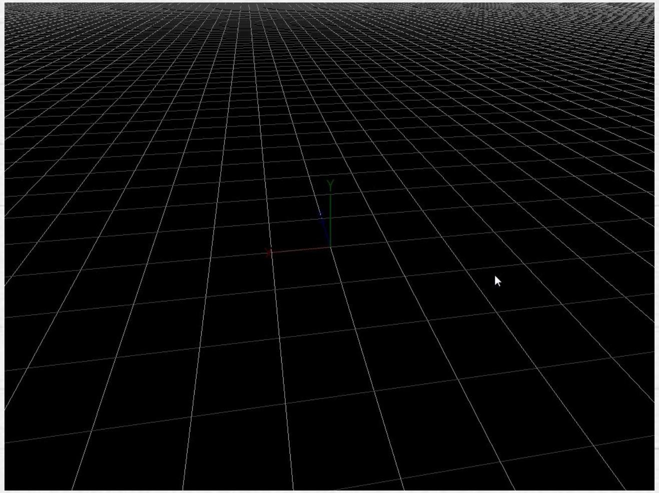
- Similar to the previous TODO
- Update camera's position
- **Read from** vectors X, Y, Z. You don't need to change them
- Update **Reference** as well!

TODO 5

```
// TODO 5: Make the camera go left (a) and right with (d)
// Note that the vectors X/Y/Z contain the current axis of the camera
// you can read them to modify Position
```

- ✗ Similar to the previous TODO
- ✗ Update camera's position
- ✗ You'll need to access the **vectors X, Y, Z** of the camera
- ✗ Update **Reference** as well!

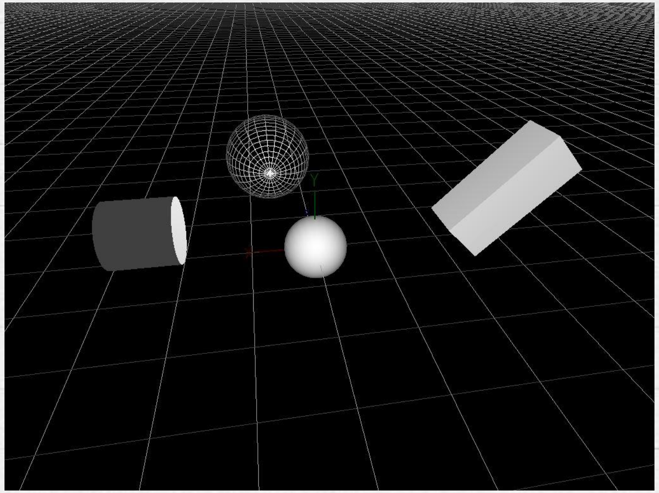
Result:



TODO 6

```
// TODO 6: Draw a sphere of 0.5f radius around the center
// Draw somewhere else a cube and a cylinder in wireframe
```

Result:



- ✗ Add some more primitives to the scene
- ✗ Don't forget to **Render()**!
- ✗ Try enabling "wireframe"
- ✗ Try rotating them around!

HOMEWORK

- Hardest task from this presentation!
- Next class some "voluntary" will share with us the solution :)

Homework

```
// TODO (Homework): Rotate the camera with the mouse
```

There are different ways to do that.

The easiest one may involve "LookAt(vec3)"

You may need:

```
// "u" is the vector to rotate
// "angle" is the value to rotate
// "v" is the axis of rotation
vec3 rotate(const vec3 &u, float angle, const vec3 &v);
```

```
int GetMouseXMotion();
int GetMouseYMotion();
```

Next Week . . .

Bullet Integration