Wavefront 2: Model & Code (lab07a)

OpenGL

Lab07b: Wavefront 3

- Perspective
  - Change the Projection back to orthographic. What effect does it have on the rendered models?
  - Design and implement a extension to the classes devised to date to make the perspective selectable, perhaps even without a restart. For instance a specific key combination could switch to/from orthographic/perspective
- Camera
  - In earlier labs, we implemented a rudimentary mechanism for navigation a scene. Propose and implement a design whereby this is modeled as a Camera abstraction - that is placed within a scene and can be moved around via some keyboard sequences

Perspective: Keystrokes

- ‘1’

```
gluPerspective(60.0f, 1, 1.0, 1000.0);
```

```
Vector3(0,0,-20).translate();
```

- ‘2’

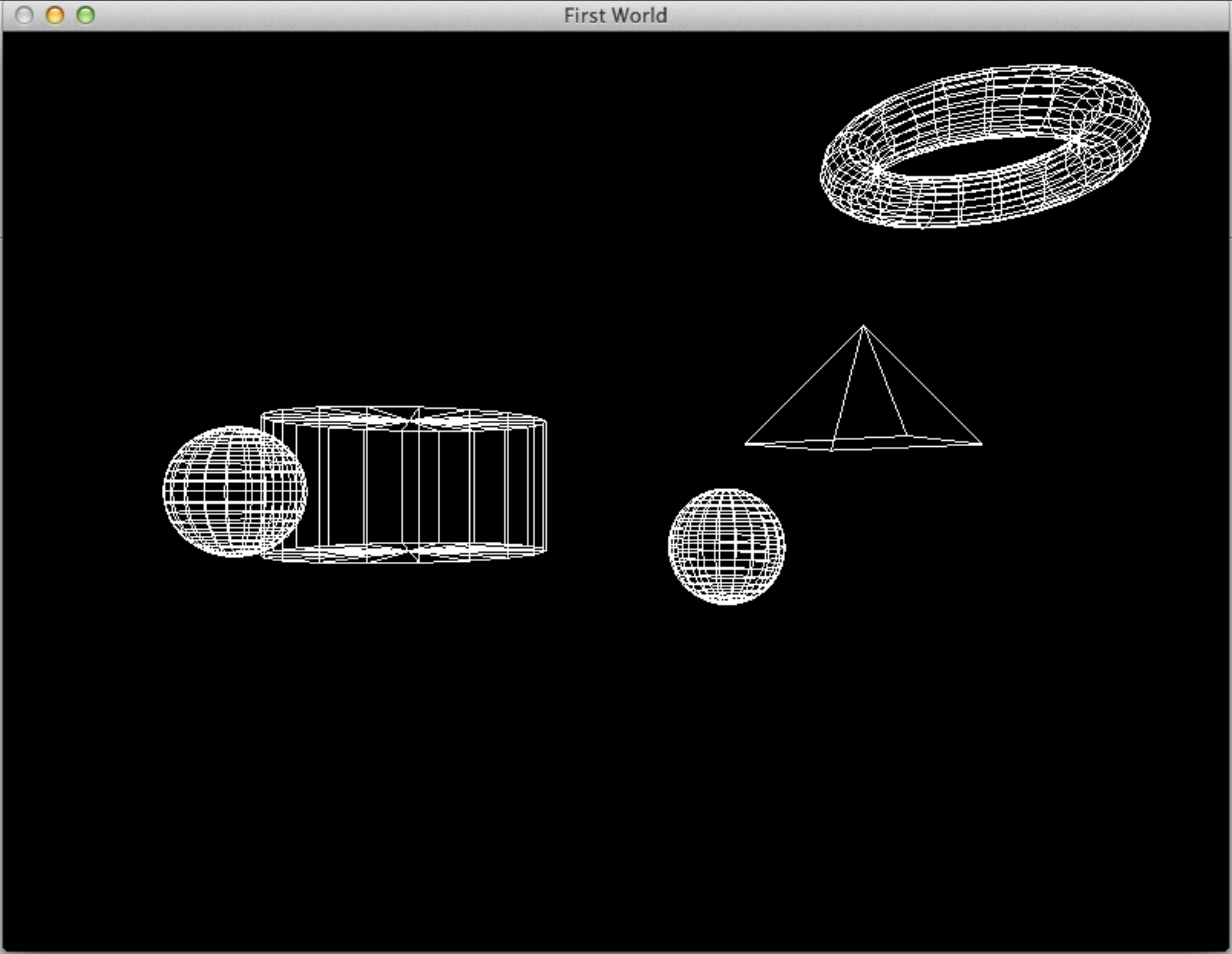
```
glOrtho(-10,10,-10,10,-10,10);
glMatrixMode ( GL_MODELVIEW);
Vector3::UnitX.rotate(angle);
```

- ‘3’

```
glOrtho(-10,10,-10,10,-10,10);
glMatrixMode ( GL_MODELVIEW);
Vector3::UnitY.rotate(angle);
```

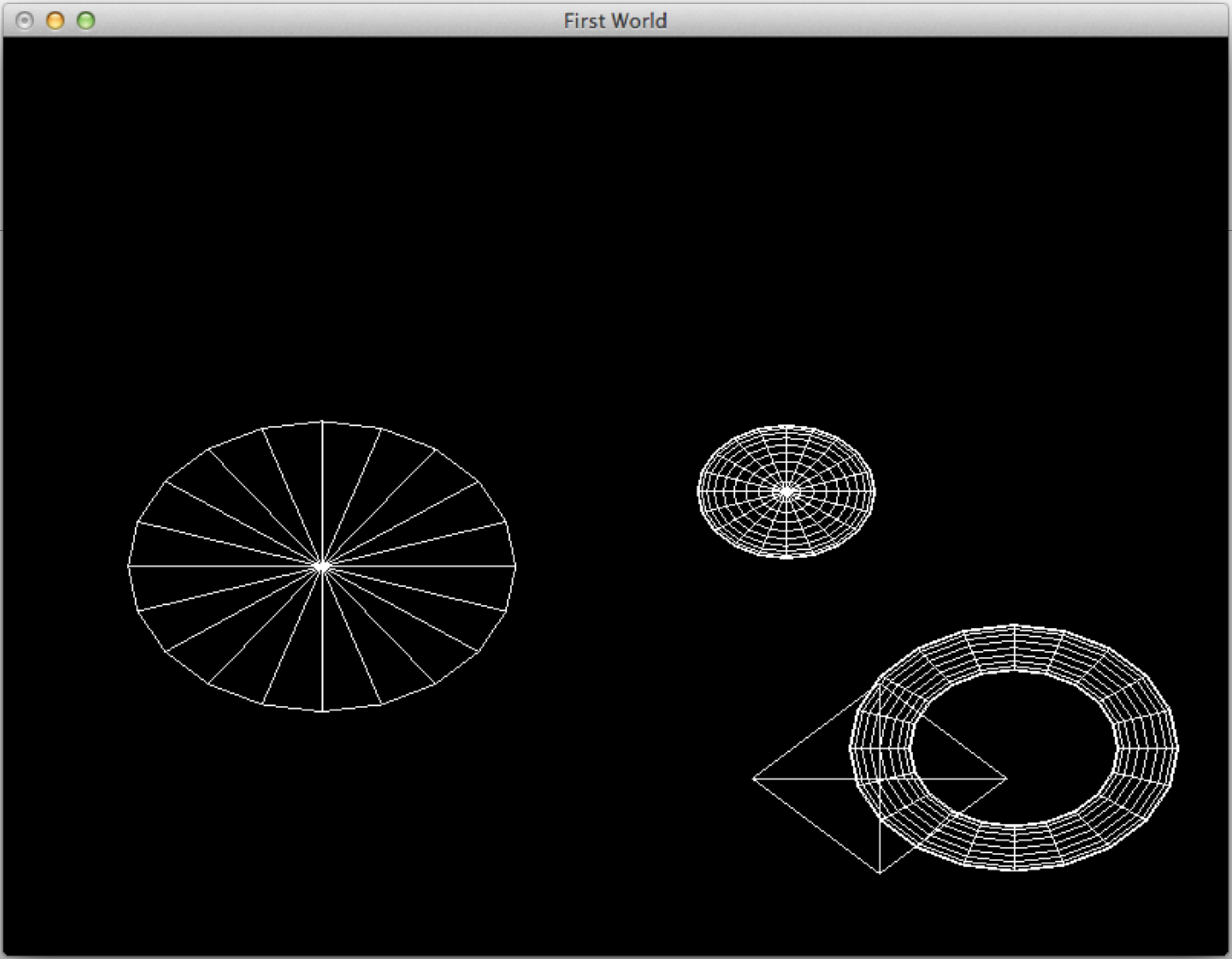
- ‘4’

```
glOrtho(-10,10,-10,10,-10,10);
glMatrixMode ( GL_MODELVIEW);
Vector3::UnitZ.rotate(angle);
```

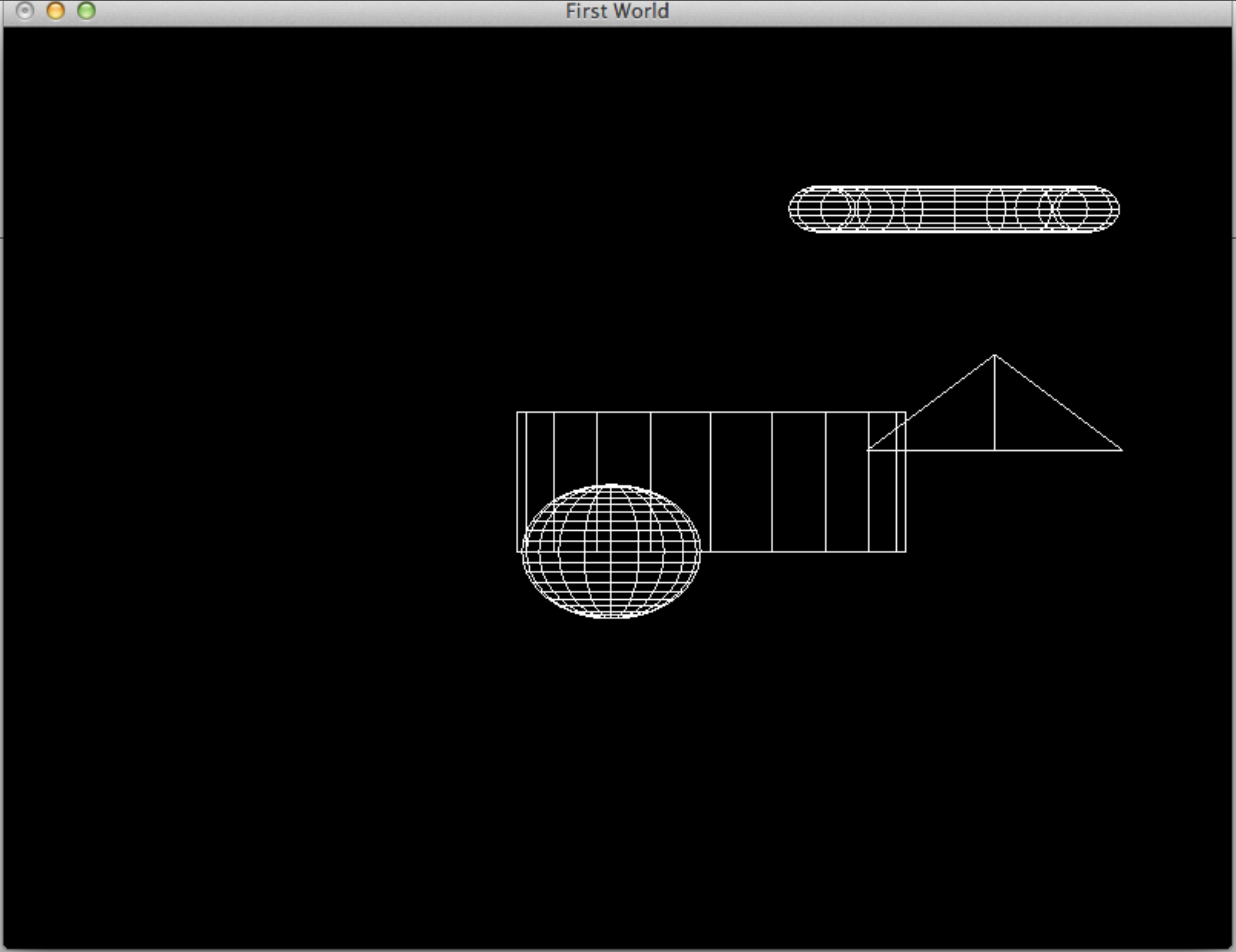


```
gluPerspective(60.0f, 1, 1.0, 1000.0);
```

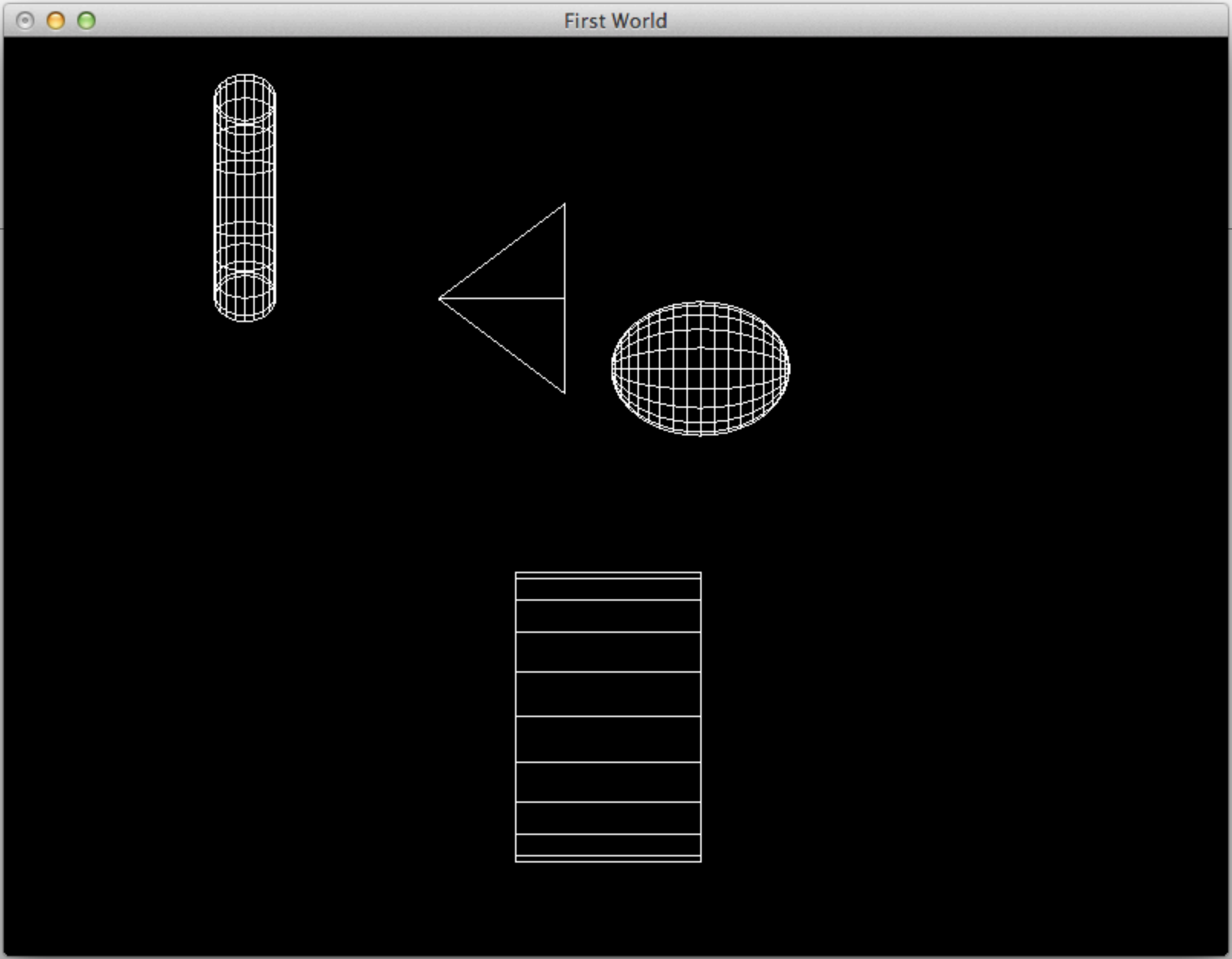
```
Vector3(0,0,-20).translate();
```



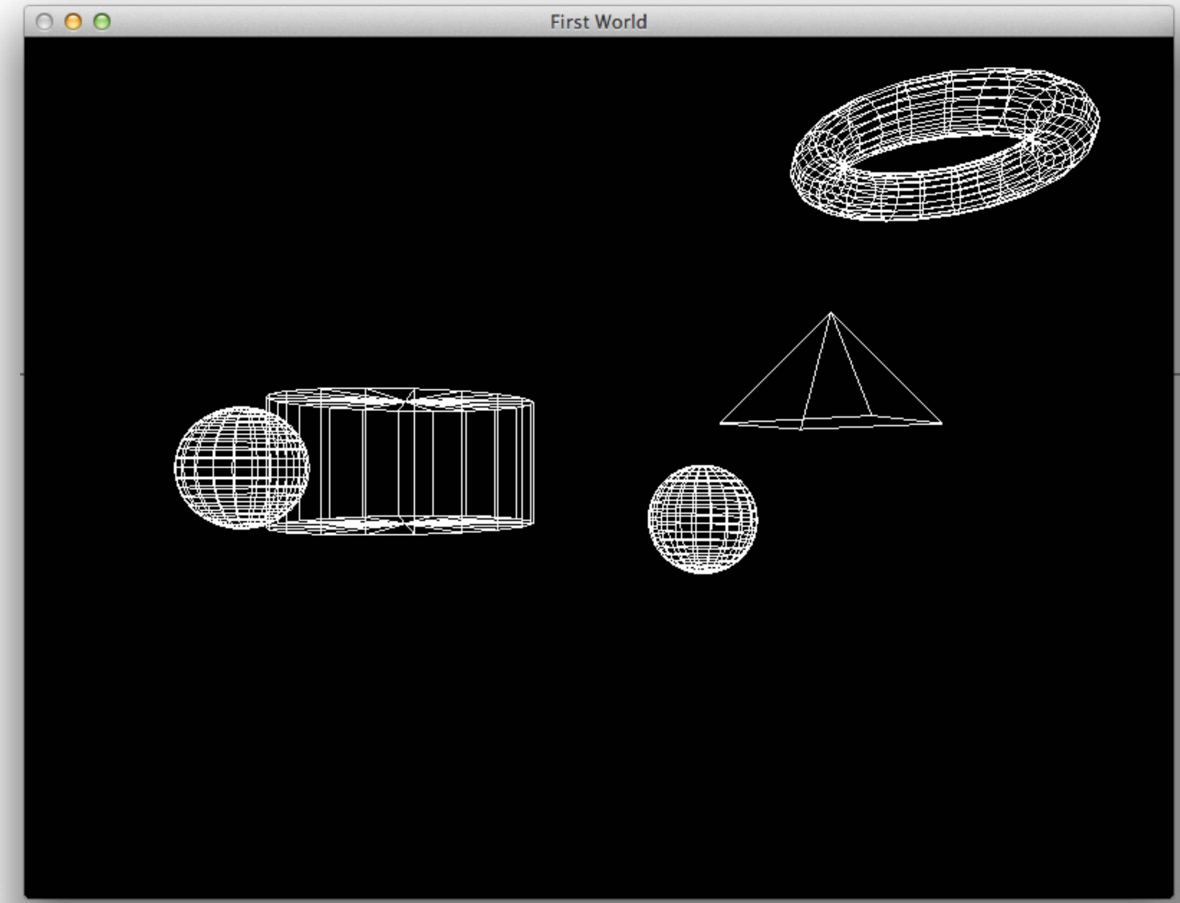
```
glOrtho(-10,10,-10,10,-10,10);
glMatrixMode ( GL_MODELVIEW);
Vector3::UnitX.rotate(angle);
```

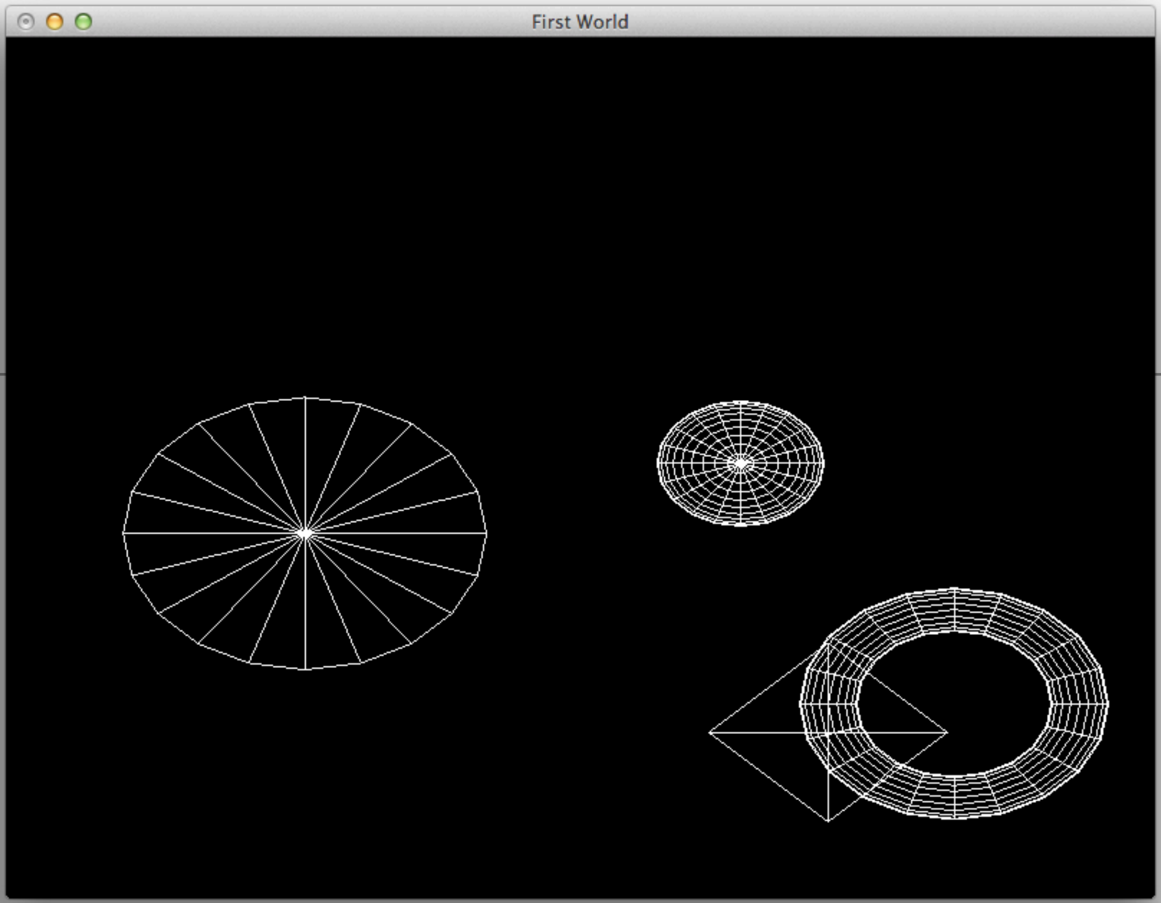


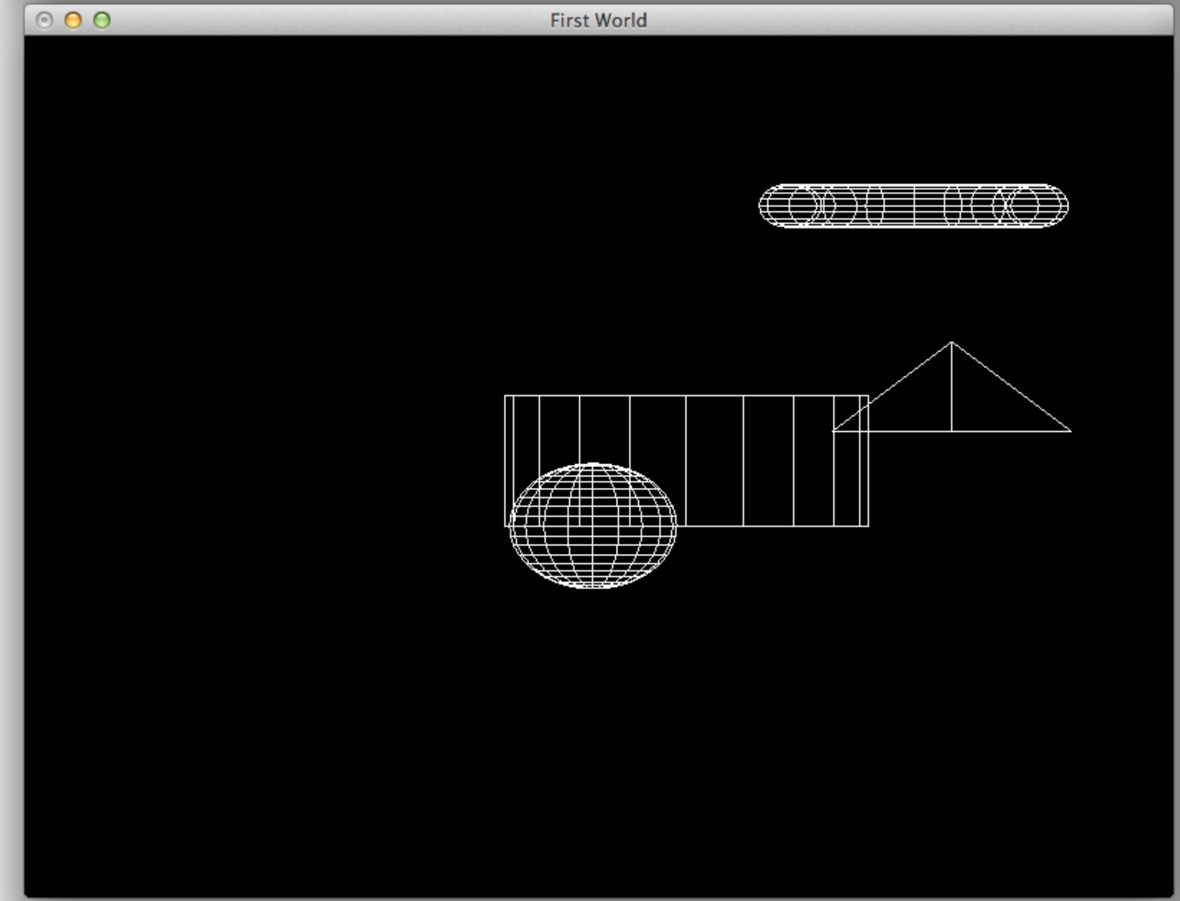
```
glOrtho(-10,10,-10,10,-10,10);
glMatrixMode ( GL_MODELVIEW);
Vector3::UnitY.rotate(angle);
```

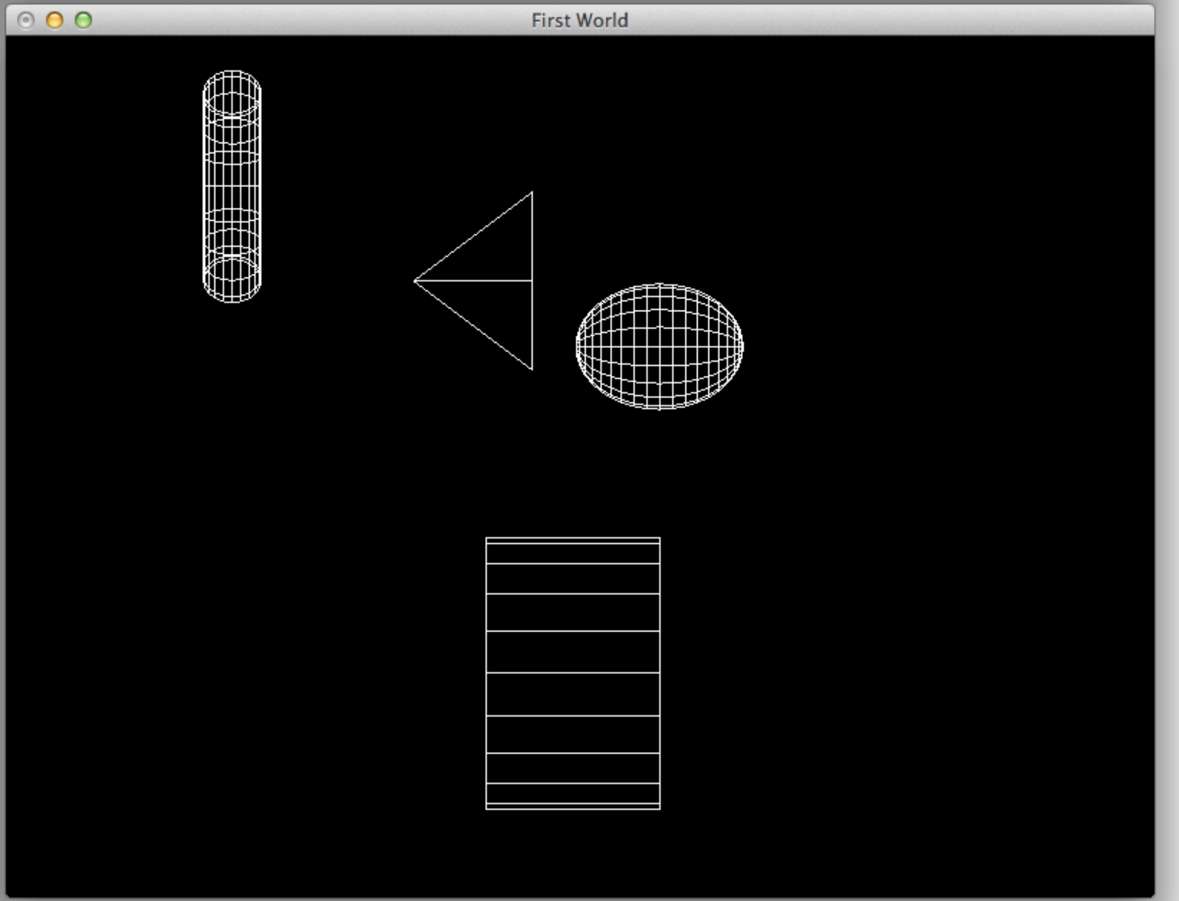


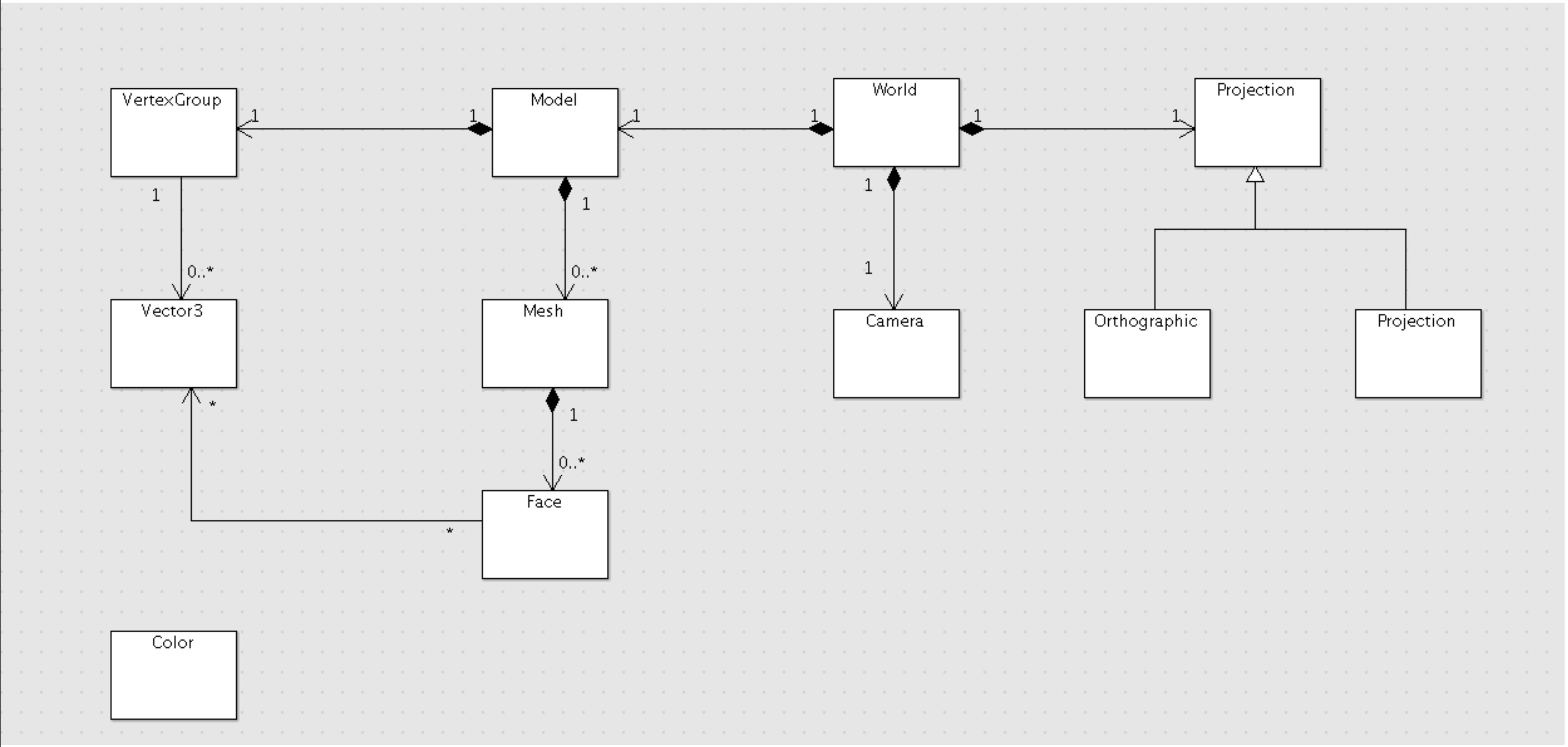
```
glOrtho(-10,10,-10,10,-10,10);
glMatrixMode ( GL_MODELVIEW);
Vector3::UnitZ.rotate(angle);
```


First World


First World


First World


First World


VertexGroup
Model
World
Projection
1
1
1
1
1
1
1
1
1
1
0..*
0..*
Vector3
Mesh
Camera
Orthographic
Projection
*
1
0..*
Face
*
Color

Projection

```
typedef std::pair<float, float> Range;

struct Projection
{
  Range windowSize;

  void resize(Range size);
  virtual void render()=0;
};
```

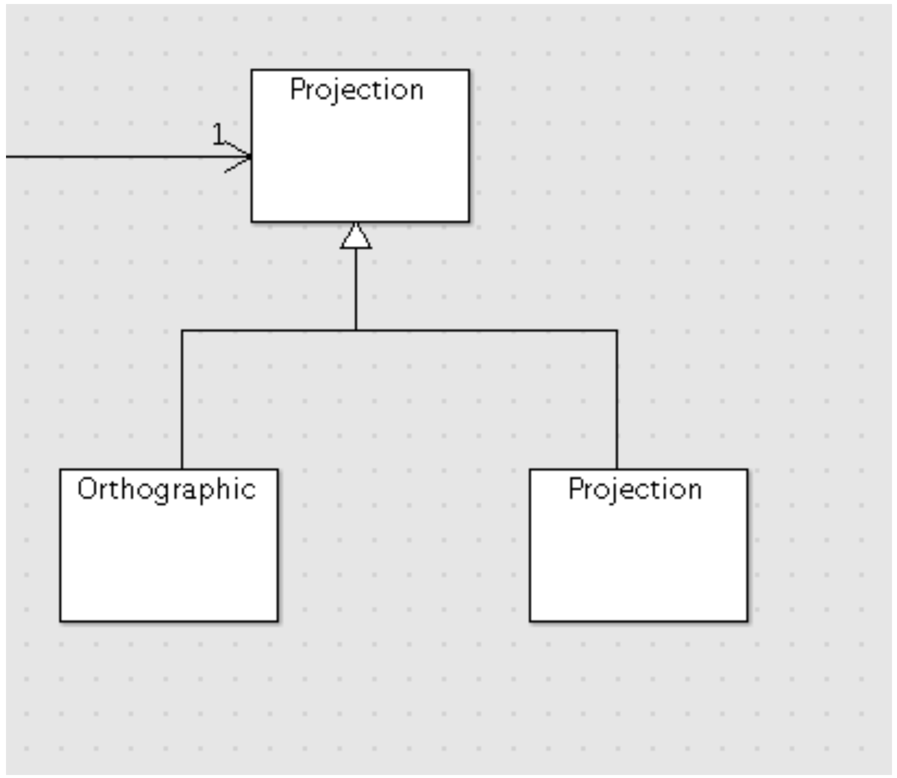
```
void Projection::resize(Range size)
{
  windowSize = size;
}
```

```
struct Orthographic: public Projection
{
  Range xRange;
  Range yRange;
  Range zRange;
  Vector3 axis;
  int angle;

  Orthographic(Range x, Range y, Range z, int angle, Vector3 axis);
  void render();
};
```

```
struct Perspective : public Projection
{
  float fovy;
  Range zRange;
  float zDistance;

  Perspective (float fovy, Range zRange, float zDistance);
  void render();
};
```



Orthographic

```
Orthographic::Orthographic(Range x, Range y, Range z, int theAngle, Vector3 theAxis)
: xRange(x), yRange(y), zRange(z), angle(theAngle), axis(theAxis)
{
}

void Orthographic::render()
{
  glLoadIdentity();
  glViewport(0, 0, windowSize.first, windowSize.second);
  glMatrixMode ( GL_PROJECTION);
  glLoadIdentity();
  glOrtho(xRange.first, xRange.second, yRange.first, yRange.second, zRange.first, zRange.second);
  glMatrixMode ( GL_MODELVIEW);

  axis.rotate(angle);
}
```

Perspective

```
Perspective::Perspective (float fovy, Range zRange, float zDistance)
: fovy(fovy), zRange(zRange), zDistance(zDistance)
{
}

void Perspective::render()
{
  glLoadIdentity();
  glViewport(0, 0, windowSize.first, windowSize.second);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(fovy, windowSize.first/windowSize.second, zRange.first, zRange.second);
  glMatrixMode (GL_MODELVIEW);
  Vector3(0,0,zDistance).translate();
}
```

World

```
struct World
{
    static World& GetInstance();

    void setCmdlineParams(int*argc, char **argv);
    void initialize(int width, int height, std::string name);
    void start();
    void loadModel (std::string modelName);

    void render();
    void keyPress(unsigned char ch);

    static World* s_World;
    Model          theModel;

    Projection   *currentProjection;


    int   *argc;
    char **argv;
};
```

main

```
int main(int argc, char* argv[])
{
  theWorld.setCmdlineParams(&argc, argv);
  theWorld.initialize(800,600, "First World");

  theWorld.loadModel("model.obj");

  Projection *projection0 = Perspective(60, Range(1,1000), -10);
  Projection *projection1 = new Orthographic (Range(-10,10), Range(-10,10), Range(-10,10), 90, Vector3::UnitX);
  Projection *projection2 = new Orthographic (Range(-10,10), Range(-10,10), Range(-10,10), 90, Vector3::UnitY);
  Projection *projection3 = new Orthographic (Range(-10,10), Range(-10,10), Range(-10,10), 90, Vector3::UnitZ);

  theWorld.currentProjection = projection1;

  theWorld.start();
  return 0;
}
```

reshape

```
void reshape(int w, int h)
{
  theWorld.currentProjection->resize(Range(w,h));
  theWorld.currentProjection->render()
}
```

```
typedef std::map <char, Projection*>  ProjectionMap
struct World
{

    static World& GetInstance();

    void setCmdlineParams(int*argc, char **argv);
    void initialize(int width, int height, std::string name);
    void start();
    void loadModel (std::string modelName);

    void render();
    void keyPress(unsigned char ch);

    static World* s_World;
    Model         theModel;
    Projection   *currentProjection;
    ProjectionMap projections;

    int   *argc;
    char **argv;
};
```

```
int main(int argc, char* argv[])
{
  theWorld.setCmdlineParams(&argc, argv);
  theWorld.initialize(800,600, "First World");

  theWorld.loadModel("model.obj");

  // put projections into map, and initialise the currentProjection


  theWorld.start();
  return 0;
}
```

```
void World::keyPress(unsigned char ch)
{
  //Select and apply the correct projection

}
```