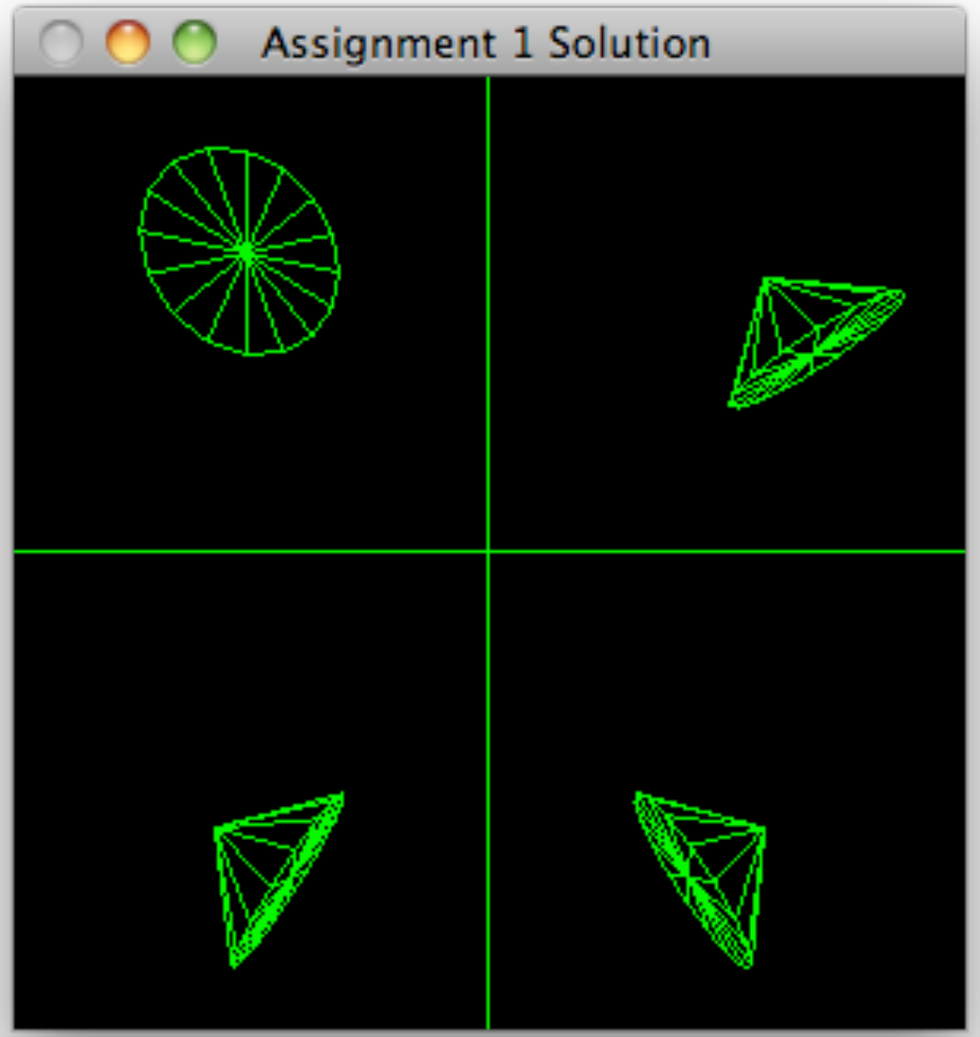
Assignment 1 Solution

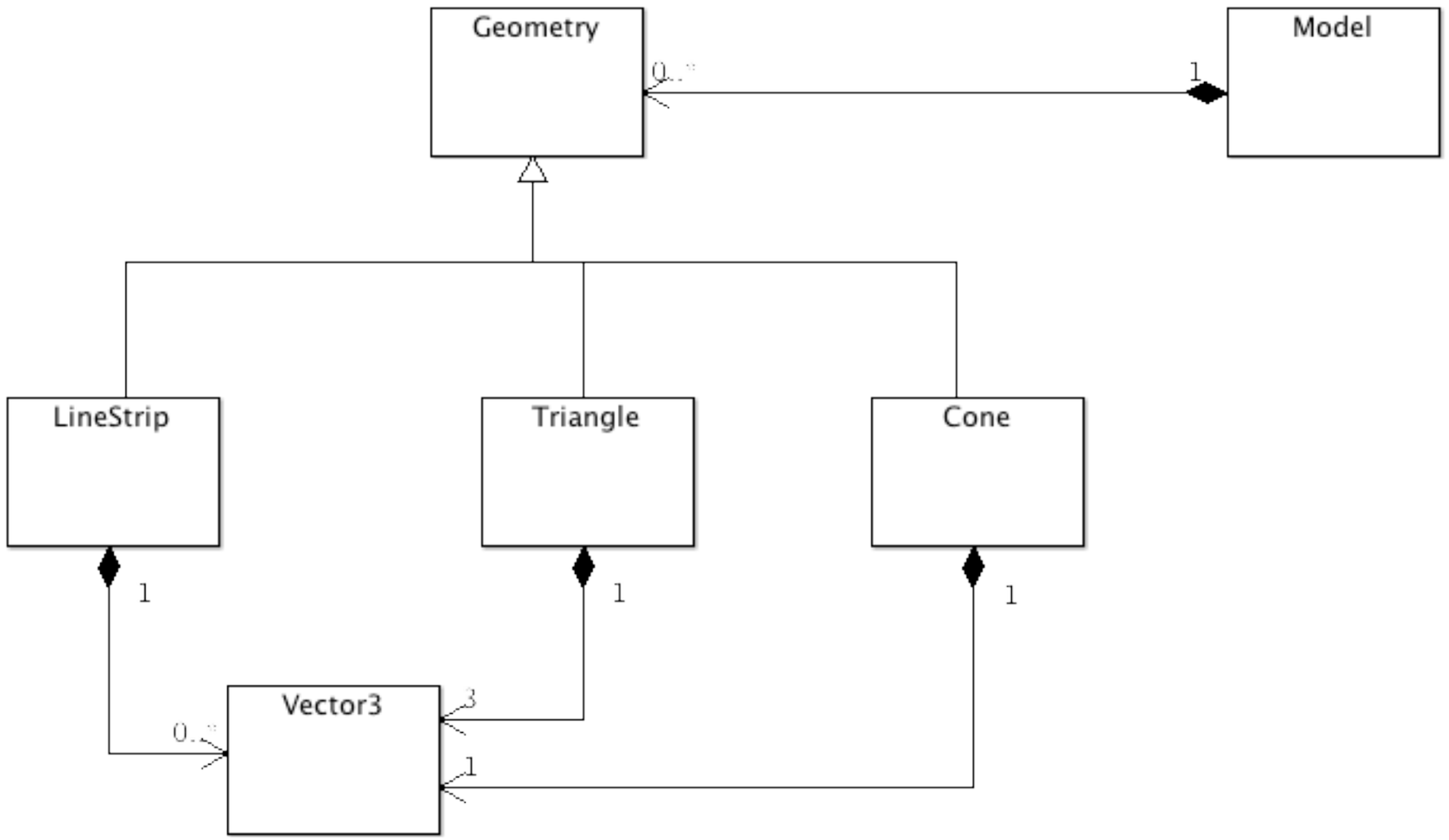
OpenGL

Specification

- Develop a working OpenGL eclipse project to display 1 or more 3D cones, animated to rotate around one of their axis with single or varying frequency.



Design



Geometry

- Abstract class
- Represents an entity that can be rendered
- Has a unique name

```
struct Geometry
{
  Geometry(std::istream& is);

  virtual void render()=0;

  std::string name;
};
```

Enhanced Geometry

- Extend Geometry to include rotation axis + angle.

- When rotate is called, it rotates around the given axis by the initial rotation + the angle

```
struct Geometry
{
  Geometry(std::istream& is);
  void rotate(int increment, const Vector3& axis);
  virtual void render()=0;

  std::string name;
  int angle;
  Vector3 rotationAxis;
};
```

```
void Geometry::rotate(int increment, const Vector3& axis)
{
  angle = (angle + increment) % 360;
  rotationAxis = axis;
}
```

Cone

- Draw a cone using triangle fans

```
struct Cone : public Geometry
{
  Vector3 origin;
  float radius;
  void drawTriangleFan(Vector3 v, float radius);

  Cone(std::istream&);
  void render();
};
```

```
Cone::Cone(istream &is)
: Geometry(is), origin(is)
{
  skipComment(is);
  is >> radius;
}

void Cone::drawTriangleFan(Vector3 centre, float radius)
{
  glBegin(GL_TRIANGLE_FAN);
    centre.render();
    float angle;
    for(angle = 0.0f; angle < (2.0f*GL_PI); angle += (GL_PI/8.0f))
    {
      float x = centre.X+radius*sin(angle);
      float y = centre.Y+radius*cos(angle);
      glVertex2f(x, y);
    }
  glEnd();
}
```

Cone render()

- Rotate by the latest angle/axis set in rotate() method

```
void Cone::render()
{
  glPushMatrix();
    glRotatef(angle, rotationAxis.X, rotationAxis.Y, rotationAxis.Z);
    glFrontFace(GL_CW);
    drawTriangleFan(origin, radius);
    glFrontFace(GL_CCW);
    drawTriangleFan(Vector3(origin.X, origin.Y, 0), radius);
  glPopMatrix();
}
```

Model File

```
-300 300 -300 300 -300 300
#number of entities
6
# 1st entity
# 0 = line
0
# name
line1
# number of vertices
2
0 -300  0
0  300  0
# 1st entity
# 0 = line
0
# name
line2
# number of vertices
2
-300 0  0
300  0  0
```

```
# 2 = cone
2
#name
cone1
# origin
150 150 50
# radius
50
# 2nd entity
# 2 = cone
2
#name
cone2
# origin
-150 -150 50
#radius
50
# 2 = cone
2
#name
cone3
# origin
-150 +150 50
#radius
50
# 2 = cone
2
#name
cone4
# origin
150 -150 50
#radius
50
```

Model Load

```
enum EntityType {LineStripId, TriangleId, ConeId};

Model::Model(istream &is)
{
    //..

    Geometry *entity=0;
    switch (typeId)
    {
      case LineStripId: {
                          entity = new LineStrip(is);
                          break;
                        }
      case TriangleId:  {
                          entity = new Triangle(is);
                          break;
                        }
        case ConeId:    {
                          entity = new Cone(is);
                          break;
                        }
    }
    if (entity != 0)
    {
        entities[entity->name] = entity;
    }
  }
}
```

Animation

- Hand coded.
- Retrieve named entities
- Rotate them by some increment.

```
void timerFunction(int value)
{
  Cone *cone1 = (Cone*) model->get("cone1");
  cone1->rotate(2, cone1->origin);

  Cone *cone2 =(Cone*) model->get("cone2");
  cone2->rotate(5, cone2->origin);

  Cone *cone3 = (Cone*)model->get("cone3");
  cone3->rotate(3, cone3->origin);

  Cone *cone4 = (Cone*)model->get("cone4");
  cone4->rotate(2, cone4->origin);

  glutPostRedisplay();
  glutTimerFunc(10, timerFunction, 1);
}
```

```
Model *model;

void setupRC()
{
  glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
  glColor3f(0.0f, 1.0f, 0.0f);
  glPolygonMode(GL_FRONT,GL_LINE);
  glPolygonMode(GL_BACK,GL_LINE);
  glFrontFace(GL_CW);;
  glShadeModel(GL_FLAT);
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_CULL_FACE);

  model = loadModel("model.txt");
  if (model != 0)
  {
    glOrtho(model->minX, model->maxX,
            model->minY, model->maxY,
            model->minZ, model->maxZ);
  }
}

void timerFunction(int value)
{
  Cone *cone1 = (Cone*) model->get("cone1");
  cone1->rotate(2, cone1->origin);

  Cone *cone2 =(Cone*) model->get("cone2");
  cone2->rotate(5, cone2->origin);

  Cone *cone3 = (Cone*)model->get("cone3");
  cone3->rotate(3, cone3->origin);

  Cone *cone4 = (Cone*)model->get("cone4");
  cone4->rotate(2, cone4->origin);

  glutPostRedisplay();
  glutTimerFunc(10, timerFunction, 1);
}
```

Full Main Program

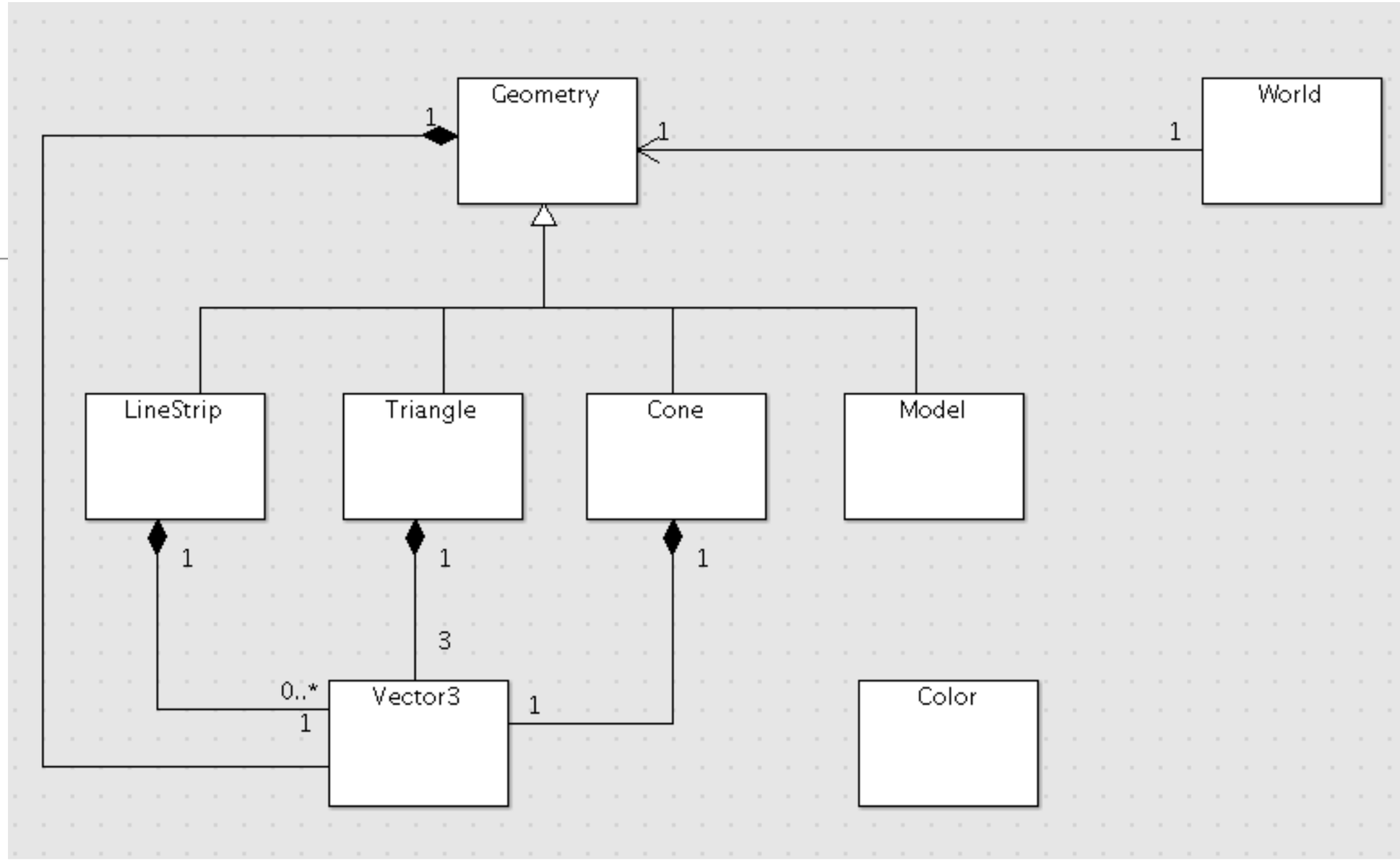
```
void renderScene(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  model->render();

  glFlush();
}

int main(int argc, char* argv[])
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
  glutCreateWindow("Assignment 1 Solution");
  glutInitWindowSize(800, 600);
  glutDisplayFunc(renderScene);
  setupRC();
  glutSpecialFunc(specialKeys);
  glutTimerFunc(33, timerFunction, 1);
  if (model != 0)
  {
    glutMainLoop();
  }
  else
  {
    cout << "Failure to load model" << endl;
  }
  return 0;
}
```

Extended Model



- Word represents encapsulation of the entire scene + animations
- Model is a type of Geometry - the Composite Pattern

World

- There is only one world - Singleton Pattern
- Encapsulates all aspects of world including:
- GLUT and OpenGL initialisation
- Suitable defaults for state machine
- Callbacks for rendering, timers and keyboard

```
#define theWorld World::GetInstance()

struct World
{
  public:
    static World& GetInstance();

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

    void add(Geometry* renderable);
    void render();
    void keyPress(unsigned char ch);

    void tickAndRender();
    void setOrthoProjection(int minX, int maxX,
                            int minY, int maxY, int minZ, int maxZ);

    static World* s_World;
    std::vector<Geometry*> renderables;
    int minX, maxX, minY, maxY, minZ, maxZ;
    int   *argc;
    char **argv;
};
```

Revised Main Program - With World Abstraction

```
Model* loadModel(const char *filename)
{
  fstream modelStream;
  modelStream.open(filename, ios::in);
  if (!modelStream.fail())
  {
    model = new Model(modelStream);
  }
  return model;
}

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

  Model *model = loadModel("model.txt");
  if (model)
  {
    theWorld.setOrthoProjection(model->minX, model->maxX,
                                model->minY, model->maxY, model->minZ, model->maxZ);
    theWorld.add(model);
  }
  theWorld.start();
  return 0;
}
```

Animation

```
void World::tickAndRender()
{
  Cone *cone1 = (Cone*) model->get("cone1");
  cone1->rotate(2, cone1->origin);

  Cone *cone2 =(Cone*) model->get("cone2");
  cone2->rotate(5, cone2->origin);

  Cone *cone3 = (Cone*)model->get("cone3");
  cone3->rotate(3, cone3->origin);

  Cone *cone4 = (Cone*)model->get("cone4");
  cone4->rotate(2, cone4->origin);

  glutPostRedisplay();
}
```

World Implementation

- Global GLUT functions:

```
World* World::s_World = NULL;

void reshape(int w, int h)
{
  glViewport(0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode ( GL_PROJECTION);

  glLoadIdentity();
  glOrtho(theWorld.minX, theWorld.maxX,
          theWorld.minY, theWorld.maxY, theWorld.minZ, theWorld.maxZ);
  glMatrixMode ( GL_MODELVIEW);
}

void renderScene(void)
{
  World::GetInstance().render();
}

void keyboard(unsigned char key, int x, int y)
{
  World::GetInstance().keyPress(key);
}

void timerFunc(int value)
{
  theWorld.tickAndRender();
  glutTimerFunc(50, timerFunc, 1);
}
```

World Implementation

- General Setup

```
World& World::GetInstance()
{
  if (s_World == NULL)
  {
    s_World = new World();
  }
  return *s_World;
}

void World::setCmdlineParams(int*argc, char **argv)
{
  this->argc = argc;
  this->argv = argv;
}

void World::setOrthoProjection(int minX, int maxX,
                               int minY, int maxY, int minZ, int maxZ)
{
  this->minX = minY;
  this->maxX = maxX;
  this->minY = minY;
  this->maxY = maxY;
  this->minZ = minZ;
  this->maxZ = maxZ;
}

void World::initialize(int width, int height, std::string name)
{
  glutInit(argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowSize(width, height);
  glutCreateWindow(name.c_str());

  Color::Black.renderClear();
  glEnable(GL_DEPTH_TEST);
  glFrontFace(GL_CCW);
  glPolygonMode(GL_FRONT,GL_LINE);
  glPolygonMode(GL_BACK,GL_LINE);
}
```

World Implementation

```
void World::add(Geometry* renderable)
{
  renderables.push_back(renderable);
}

void World::start()
{
  glutKeyboardFunc(keyboard);
  glutReshapeFunc(reshape);
  glutDisplayFunc(renderScene);
  timerFunc(0);
  glutMainLoop();
}

void World::render()
{
  Color::Black.renderClear();
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();

  for (unsigned int i=0; i<renderables.size(); i++)
  {
    renderables[i]->render();
  }
  glutSwapBuffers();
}
```