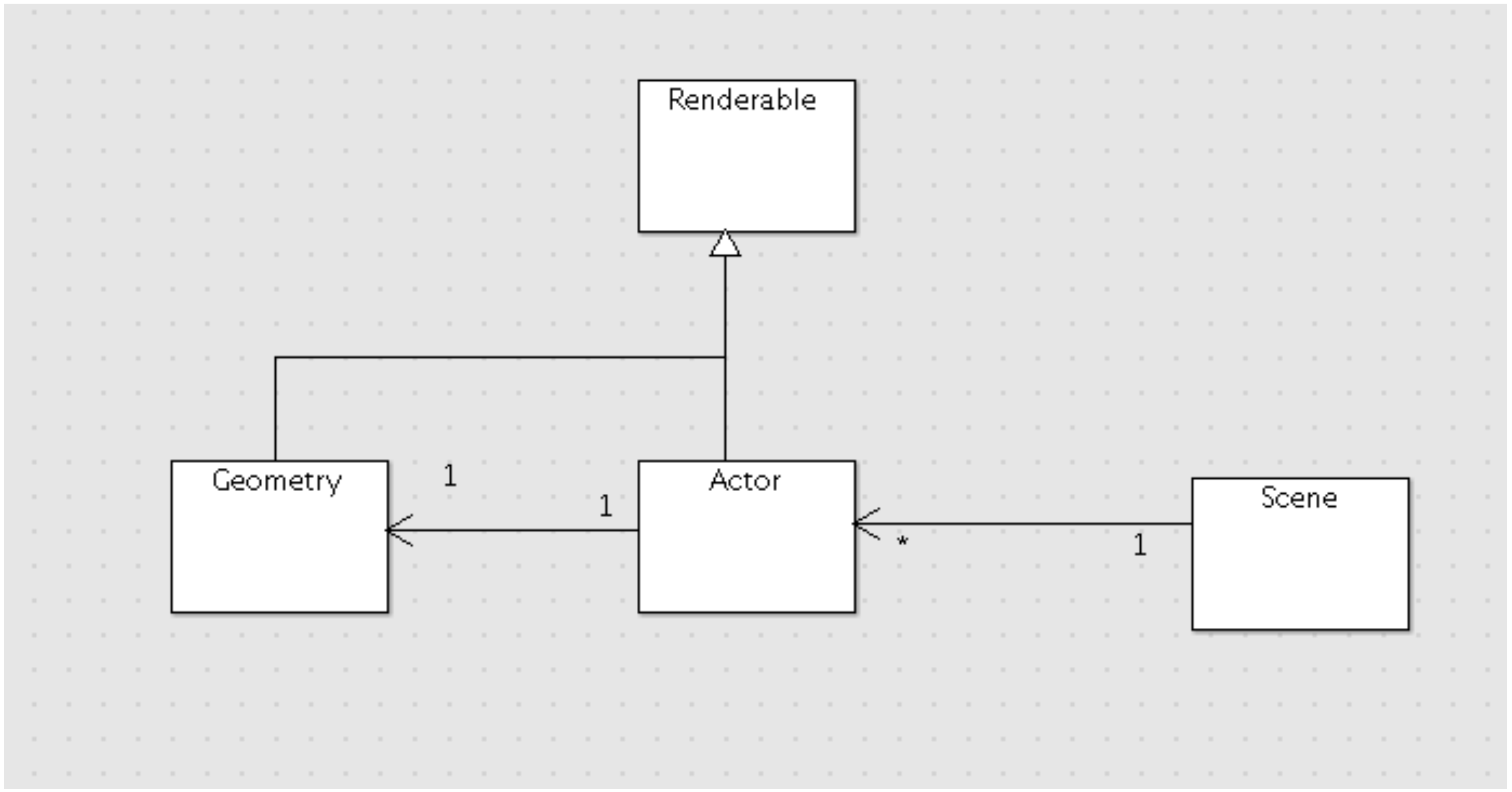
Custom Actors

OpenGL

Scene

- A Scene consists of a collection of Actors
- Actors are renderable, and have a reference to a Geometry object



“Custom” Actor

- Derive from Actor and...

  (1) Use geometry manufactured by the Model

  or

  (1) Hand coded geometry

(1) Geometry Manufactured by Model

```
struct ColourCube: public  Actor
{
  ColourCube(Geometry* g);
  void render();
};
```

```
ColourCube::ColourCube(Geometry* g) : Actor(g)
{}

void ColourCube::render()
{
  glShadeModel(GL_SMOOTH);
  glPolygonMode(GL_FRONT,GL_FILL);
  glPolygonMode(GL_BACK,GL_FILL);

  foreach (Face &face, geometry->faces)
  {
    glBegin(GL_QUADS);

    foreach (int index, face.vertexIndices)
    {
      glVertex3f( geometry->vertexGroup->vertices[index-1].X,
                  geometry->vertexGroup->vertices[index-1].Y,
                  geometry->vertexGroup->vertices[index-1].Z );
    }
    glEnd();
  }
  glPolygonMode(GL_FRONT,GL_LINE);
  Color::White.render();
}
```

(1) Scene::Scene for Loaded Geometry

- Scene constructor watches for group named ‘colourcube’
- Creates ColourCube object

```
Scene:: Scene(Model *model)
{
  foreach (GeometryMap::value_type &value, model->entities)
  {
    string name = value.first;
    Actor *actor;
    if (name == "colourcube")
    {
      actor = new ColourCube(&value.second);
    }
    else
    {
      actor = new Actor(&value.second);
    }
    actors.insert(name, actor);
  }
}
```

(2) Hand Coded Geometry

```
Color colours[][6] =
  {
    {Color::White,   Color::Yellow, Color::Red,     Color::Magenta},
    {Color::Cyan,    Color::Green,  Color::Black,   Color::Blue},
    {Color::Cyan,    Color::White,  Color::Magenta, Color::Blue},
    {Color::Green,   Color::Yellow, Color::Red,     Color::Black},
    {Color::White,   Color::Cyan,   Color::Green,   Color::Yellow},
    {Color::Magenta, Color::Blue,   Color::Black,   Color::Red}
  };

Vector3 vertices[][6] =
{
  { Vector3(-1.0f, 1.0f, 1.0f), Vector3(-1.0f, -1.0f, 1.0f), Vector3( 1.0f, -1.0f, 1.0f), Vector3( 1.0f, 1.0f, 1.0f)  },
  { Vector3( 1.0f, 1.0f,-1.0f), Vector3( 1.0f, -1.0f,-1.0f), Vector3(-1.0f, -1.0f,-1.0f), Vector3(-1.0f, 1.0f, -1.0f) },
  { Vector3(-1.0f, 1.0f,-1.0f), Vector3(-1.0f,  1.0f, 1.0f), Vector3( 1.0f,  1.0f, 1.0f), Vector3( 1.0f, 1.0f, -1.0f) },
  { Vector3( 1.0f,-1.0f,-1.0f), Vector3( 1.0f, -1.0f, 1.0f), Vector3(-1.0f, -1.0f, 1.0f), Vector3(-1.0f,-1.0f, -1.0f) },
  { Vector3( 1.0f,-1.0f, 1.0f), Vector3( 1.0f, -1.0f,-1.0f), Vector3( 1.0f,  1.0f,-1.0f), Vector3( 1.0f, 1.0f,  1.0f) },
  { Vector3(-1.0f, 1.0f, 1.0f), Vector3(-1.0f, 1.0f, -1.0f), Vector3(-1.0f, -1.0f,-1.0f), Vector3(-1.0f, -1.0f, 1.0f) }
 };
```

(2) Hand Coded Geometry

```
void drawFace(Color colours[], Vector3 vertices[])
{
  for (int i=0; i<4; i++)
  {
    colours[i].render();
    vertices[i].render();
  }
}

ColourCube::ColourCube(Geometry* g) : Actor(g)
{}

void ColourCube::render()
{

  glBegin( GL_QUADS);
  for (int i=0; i<6; i++)
  {
    drawFace(colours[i], vertices[i]);
  }
  glEnd();
}
```

Scene::Scene -

- Just insert the custom object after the model has been loaded...

```
Scene:: Scene(Model *model)
{
  foreach (GeometryMap::value_type &value, model->entities)
  {
    string name = value.first;
    Actor *actor;
    actor = new Actor(&value.second);
    actors.insert(name, actor);
  }
  string colourcube = "colourcube";

  actors.insert(colourcube, new ColourCube());
}
```

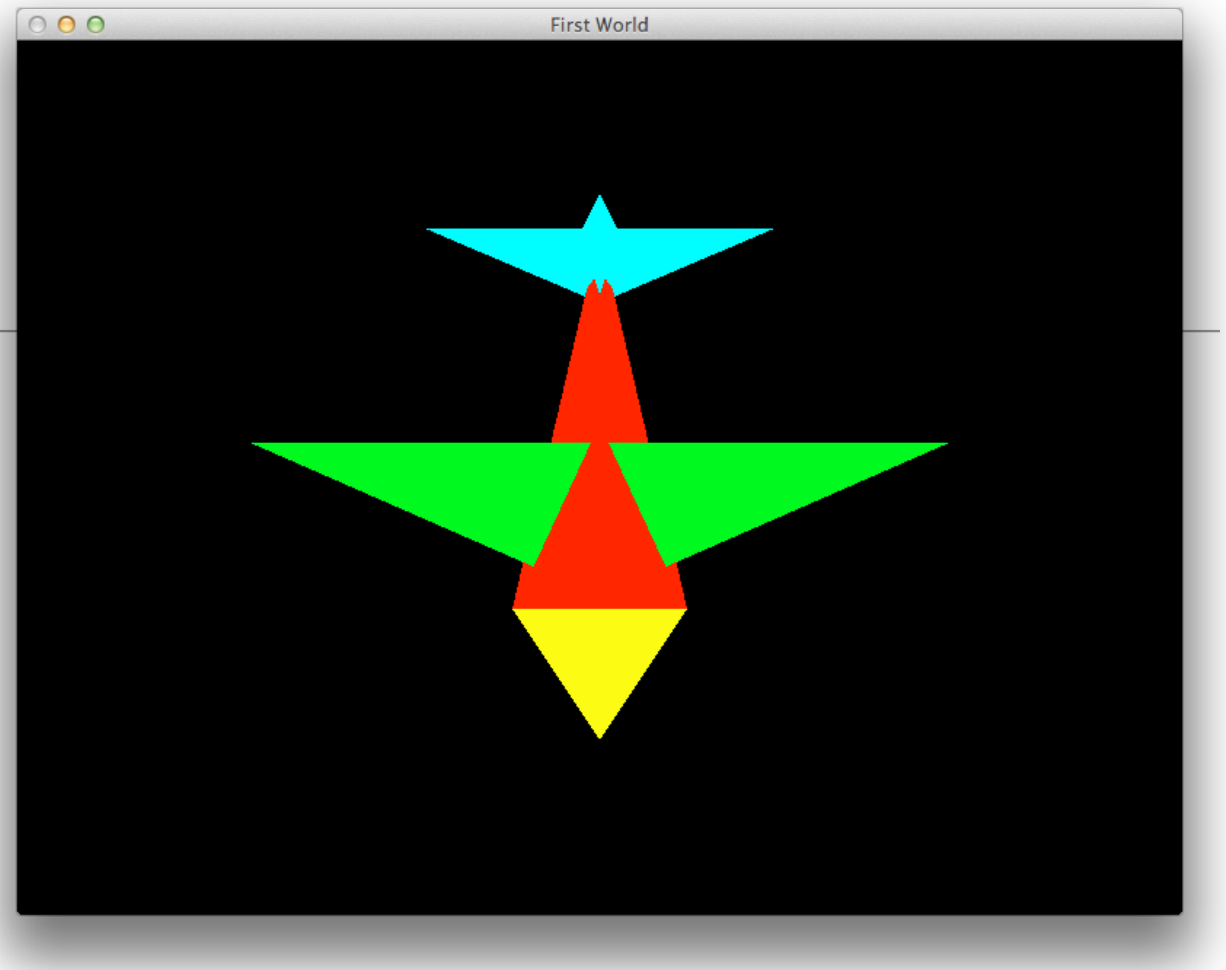
Exercise 2

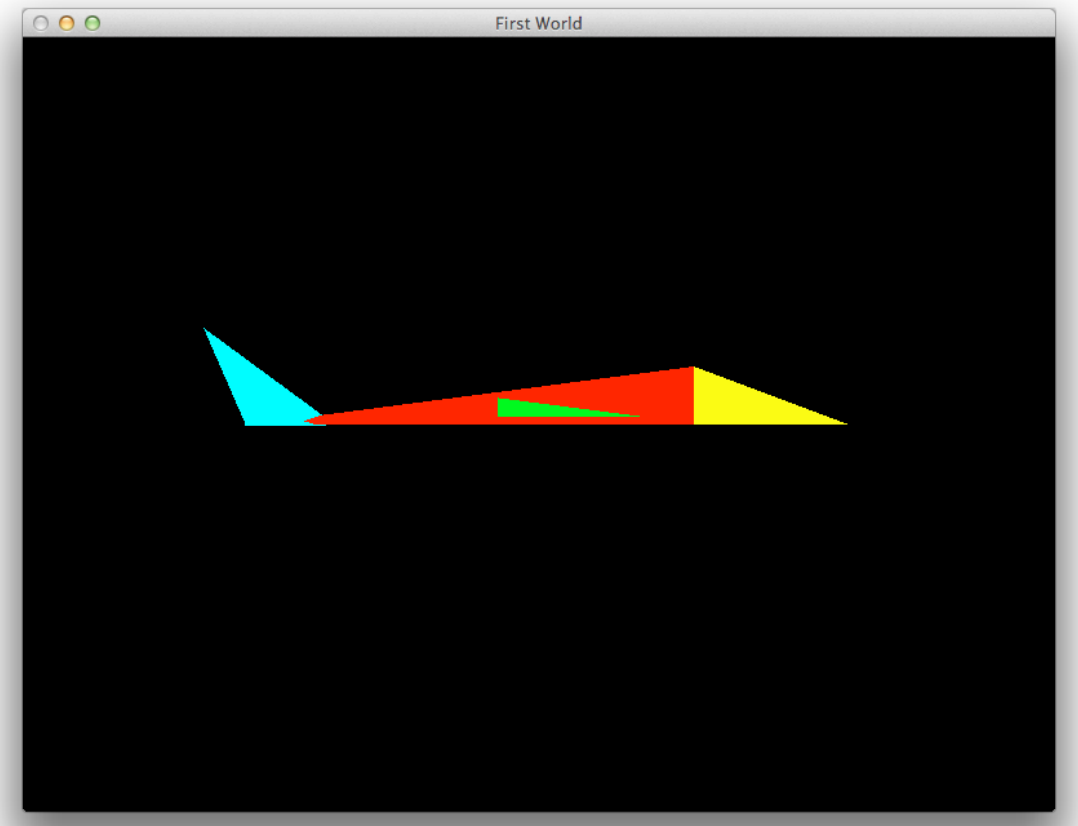
- Build another custom object - to be called "JetPlane" with this geometry
  - jetplanegeomerty.h

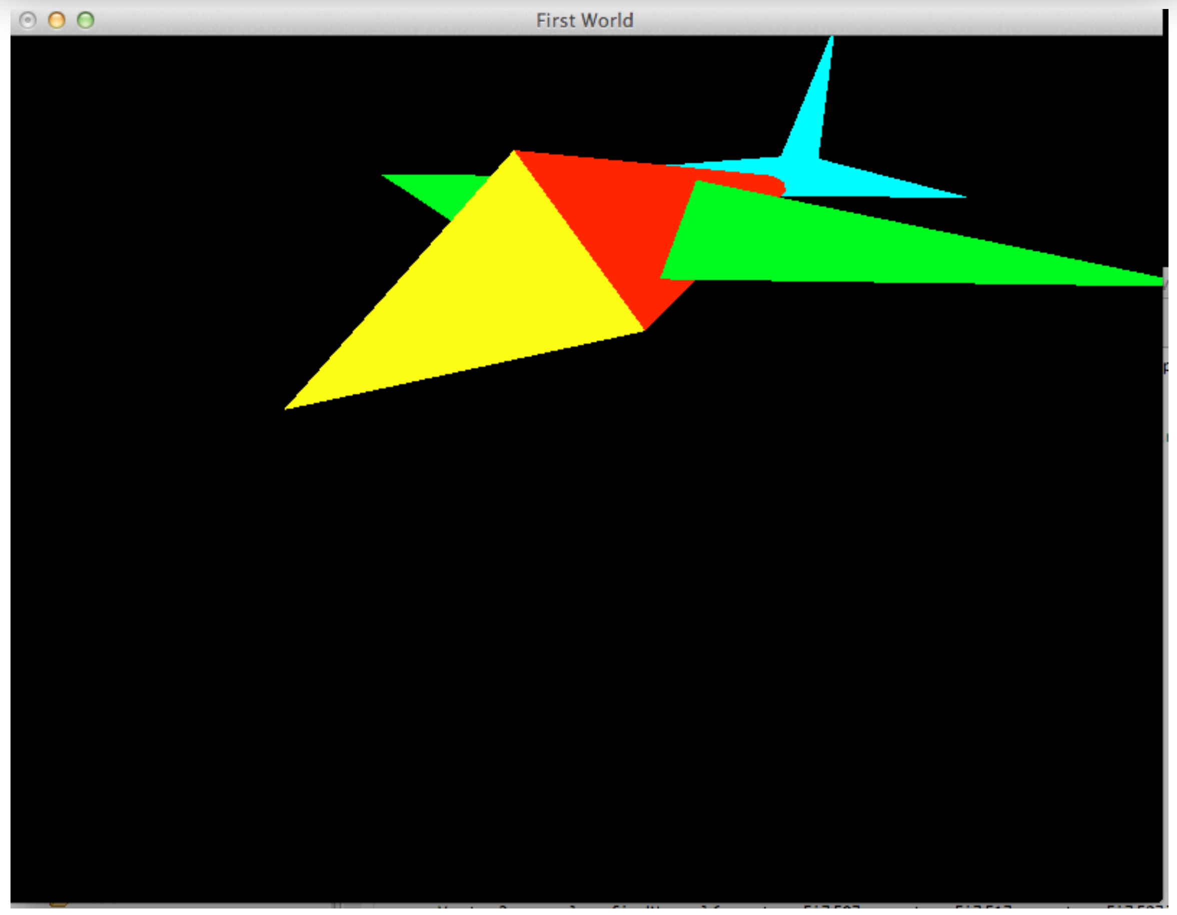
```
Vector3  noseCone[][3] =
{ { Vector3 (  0.0,  0.0,   6.0),
    Vector3 ( -1.5,  0.0,   3.0),
    Vector3 (  1.5,  0.0,   3.0)   },
  { Vector3 (  1.5,  0.0,   3.0),
    Vector3 (  0.0,  1.5 ,  3.0),
    Vector3 (  0.0,  0.0,   6.0)   },
  { Vector3 (  0.0,  0.0,   6.0),
    Vector3 (  0.0,  1.5,   3.0),
    Vector3 ( -1.5,  0.0,   3.0)   }
};

Vector3 body[][3] =
{ { Vector3 ( -1.5,  0.0,  3.0),
    Vector3 (  0.0,  1.5,  3.0),
    Vector3 (  0.0,  0.0, -5.6), },
  { Vector3 (  0.0,  0.0, -5.6),
    Vector3 (  0.0,  1.5,  3.0),
    Vector3 (  1.5,  0.0,  3.0), },
  { Vector3 (  1.5,  0.0,  3.0),
    Vector3 ( -1.5,  0.0,  3.0),
    Vector3 (  0.0,  0.0, -5.6)  }
};

Vector3 wings[][3] =
{ { Vector3 (  0.0,  .2, 2.7),
    Vector3 ( -6.0,  .2, -.8),
    Vector3 (  6.0,  .2, -.8)  },
  { Vector3 (  6.0,  .2, -.8),
    Vector3 (  0.0,  .7, -.8),
    Vector3 (  0.0,  .2, 2.7), },
  { Vector3 (  6.0,  .2, -.8),
    Vector3 ( -6.0,  .2, -.8),
    Vector3 (  0.0,  .7, -.8), },
 {  Vector3 (  0.0,  .2, 2.7),
    Vector3 (  0.0,  .7, -.8),
    Vector3 ( -6.0,  .2, -.8) }
};
```

```
Vector3 tail[][3] =
{ { Vector3 (-3.0,   -.05,  -5.7),
    Vector3 ( 3.0,   -.05,  -5.7),
    Vector3 ( 0.0,   -.05,  -4.0), },
  { Vector3 ( 0.0,   -.05,  -4.0),
    Vector3 ( 3.0,   -.05,  -5.7),
    Vector3 ( 0.0,   .40,   -5.7), },
  { Vector3 ( 0.0,   .40,   -5.7),
    Vector3 (-3.0,  -.05,   -5.7),
    Vector3 ( 0.0,  -.05,   -4.0)  },
  { Vector3 ( 3.0,   -.05,  -5.7),
    Vector3 (-3.0,  -.05,   -5.7),
    Vector3 ( 0.0,   .40,   -5.7), },
  { Vector3 ( 0.0,   .05,   -4.0),
    Vector3 (  .3,   .05,   -5.7),
    Vector3 ( 0.0,   2.50,  -6.5), },
  { Vector3 ( 0.0,   2.50,  -6.5),
    Vector3 ( -.3,   .05,   -5.7),
    Vector3 ( 0.0,   .05,   -4.0)  },
 {  Vector3 ( .3,    .05,   -5.7),
    Vector3 ( -.3,   .05,   -5.7),
    Vector3 ( 0.0,   2.5,   -6.5) }
};
```

Which is to Render as:


First World


First World


First World