Extending the Model

OpenGL

Vertex & LineStrip

```
struct Vertex
{
  float x;
  float y;

  Vertex(istream& is);
  void render();
};

struct LineStrip
{
  vector<Vertex> vertices;

  LineStrip(istream& is);
  void render();
};
```

```
Vertex::Vertex(istream &is)
{
  is >> x >> y;
}

void Vertex::render()
{
  glVertex2f(x, y);
}

LineStrip::LineStrip(istream &is)
{
  int size;
  is >> size;
  for (int i = 0; i < size; i++)
  {
    Vertex vertex(is);
    vertices.push_back(vertex);
  }
}

void LineStrip::render()
{
  glBegin( GL_LINE_STRIP);
  for (unsigned int i = 0; i < vertices.size(); i++)
  {
    vertices[i].render();
  }
  glEnd();
}
```

Model

```
struct Model
{
  vector<LineStrip> lineStrips;

  Model(istream& is);
  void render();
};
```

```
Model::Model(istream &is)
{
  int size;
  is >> size;
  for (int i = 0; i < size; i++)
  {
    LineStrip LineStrip(is);
    lineStrips.push_back(LineStrip);
  }
}

void Model::render()
{
  for (unsigned int i = 0; i < lineStrips.size(); i++)
  {
    lineStrips[i].render();
  }
}
```

New Entity: Triangle

```
struct Triangle
{
  Vertex p1, p2, p3;

  Triangle(istream& is);
  void render();
};
```

```
Triangle::Triangle(istream &is)
: p1(is),p2(is),p3(is)
{
}

void Triangle::render()
{
  glBegin( GL_TRIANGLES);
    p1.render();
    p2.render();
    p3.render();
  glEnd();
}
```

Augmenting Model (1)

```
struct Model
{
  int maxX, maxY;
  vector<LineStrip> lines;
  vector<Triangle>  triangles;

  Model(istream& is);
  ~Model();
  void render();
};
```

Augmenting Model (2)

```
Model::Model(istream &is)
{
  int size;
  is >> maxX >> maxY;
  is >> size;
  for (int i = 0; i < size; i++)
  {
    int typeId;
    is >> typeId;
    switch (typeId)
    {
      case LineStripId: { LineStrip line(is);
                          lines.push_back(line);
                          break;
                        }
      case TriangleId:  { Triangle triangle(is);
                          triangles.push_back(triangle);
                          break;
                        }
    }
  }
}
```

Augmenting Model (3)

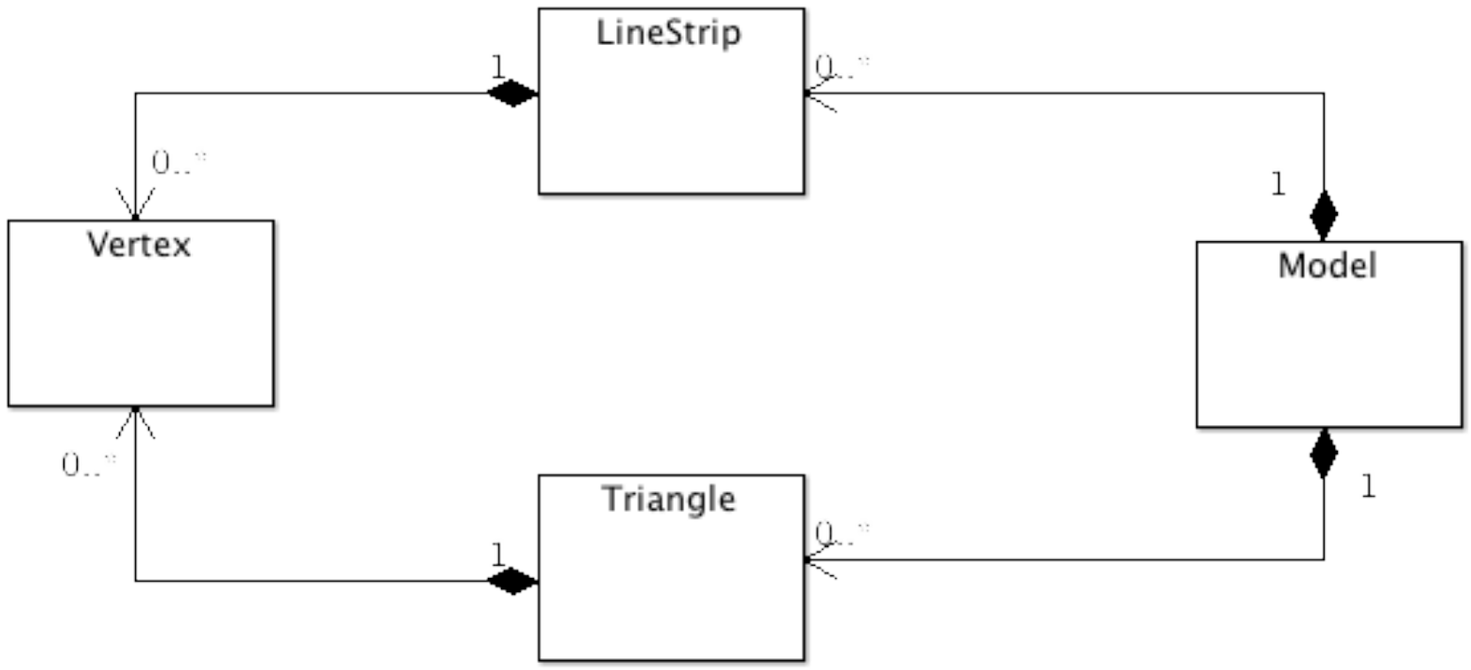
```
void Model::render()
{
  for (unsigned int i = 0; i < lines.size(); i++)
  {
    lines[i].render();
  }
  for (unsigned int i = 0; i < triangles.size(); i++)
  {
    triangles[i].render();
  }
}
```

```
struct Vertex
{
  float x;
  float y;

  Vertex(istream& is);
  void render();
};
```

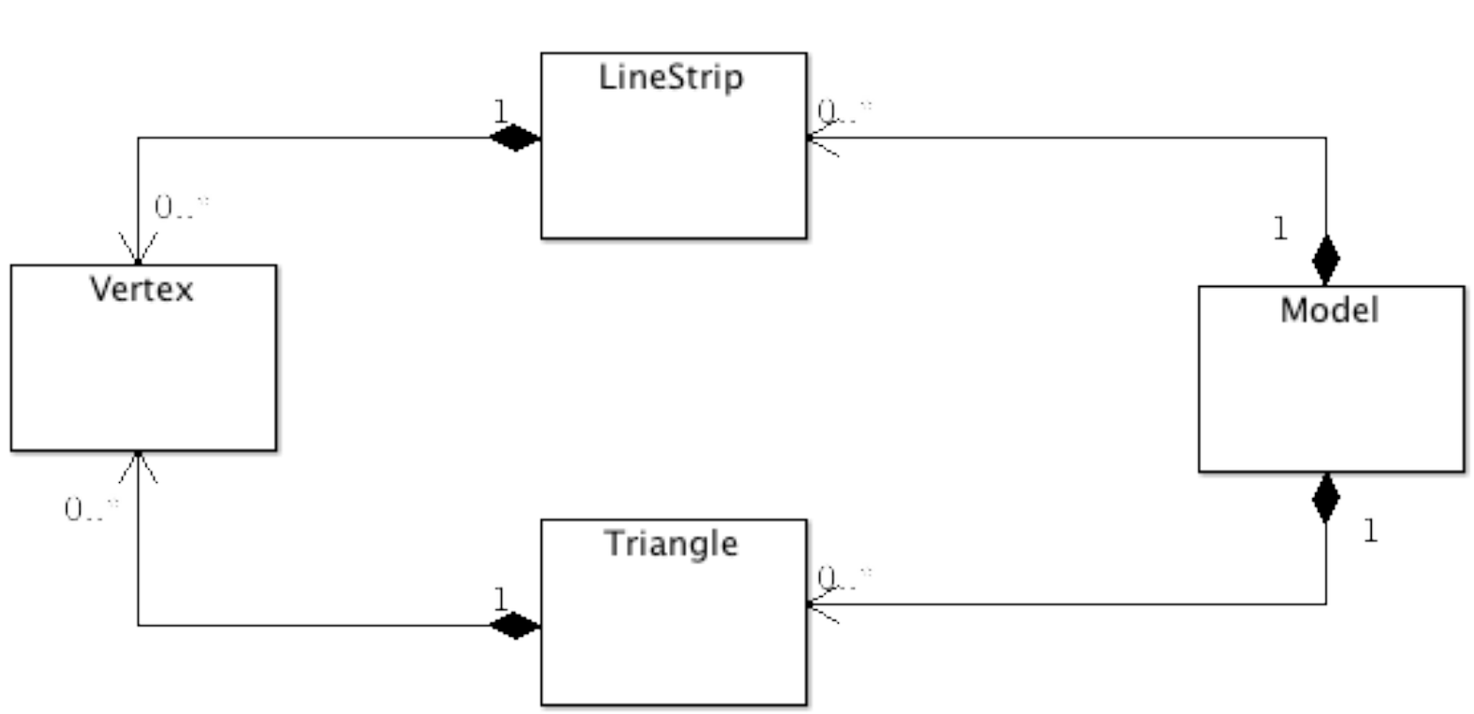
```
struct Triangle
{
  Vertex p1, p2, p3;

  Triangle(istream& is);
  void render();
};
```

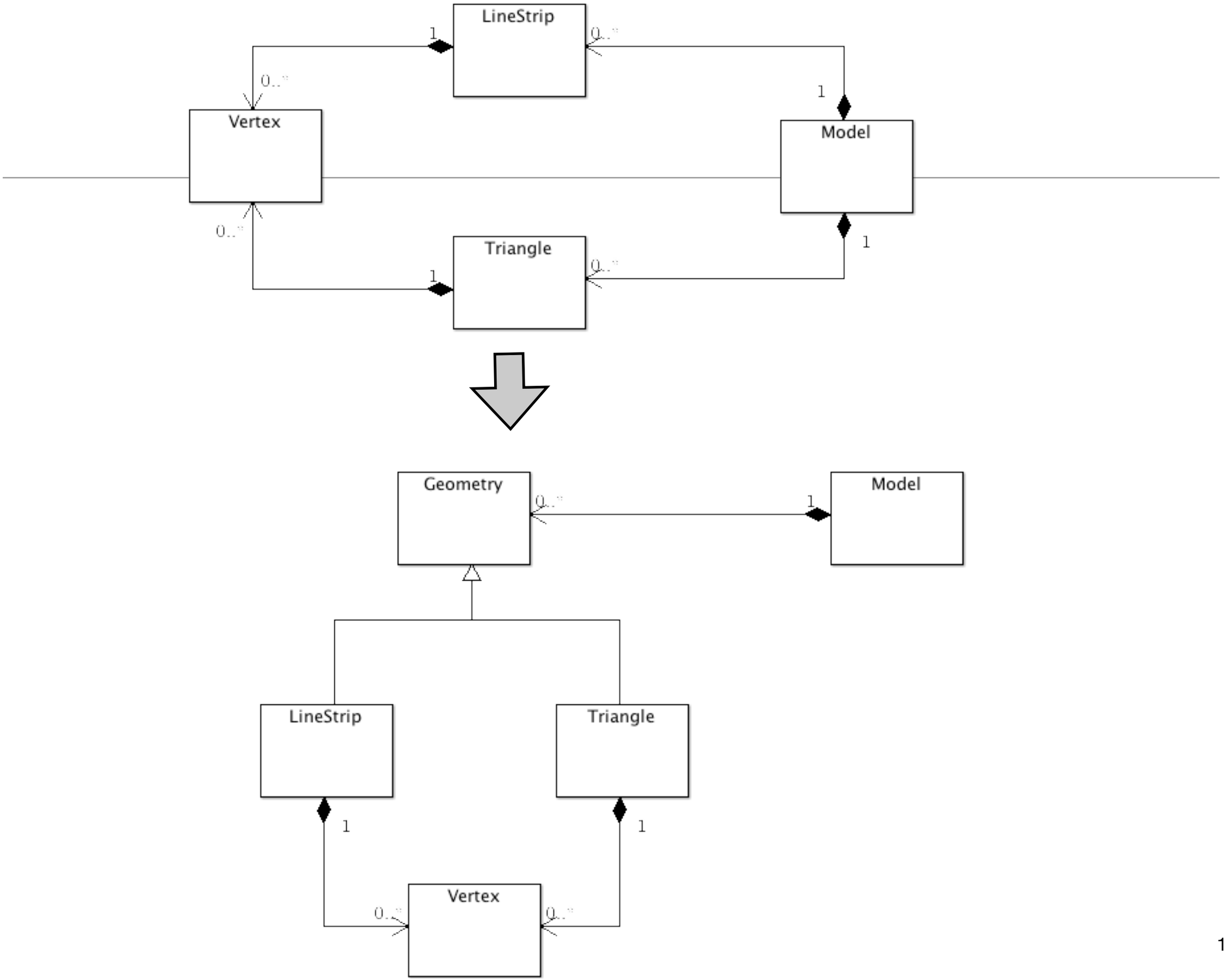
```
struct LineStrip
{
  vector<Vertex> vertices;

  LineStrip(istream& is);
  void render();
};
```

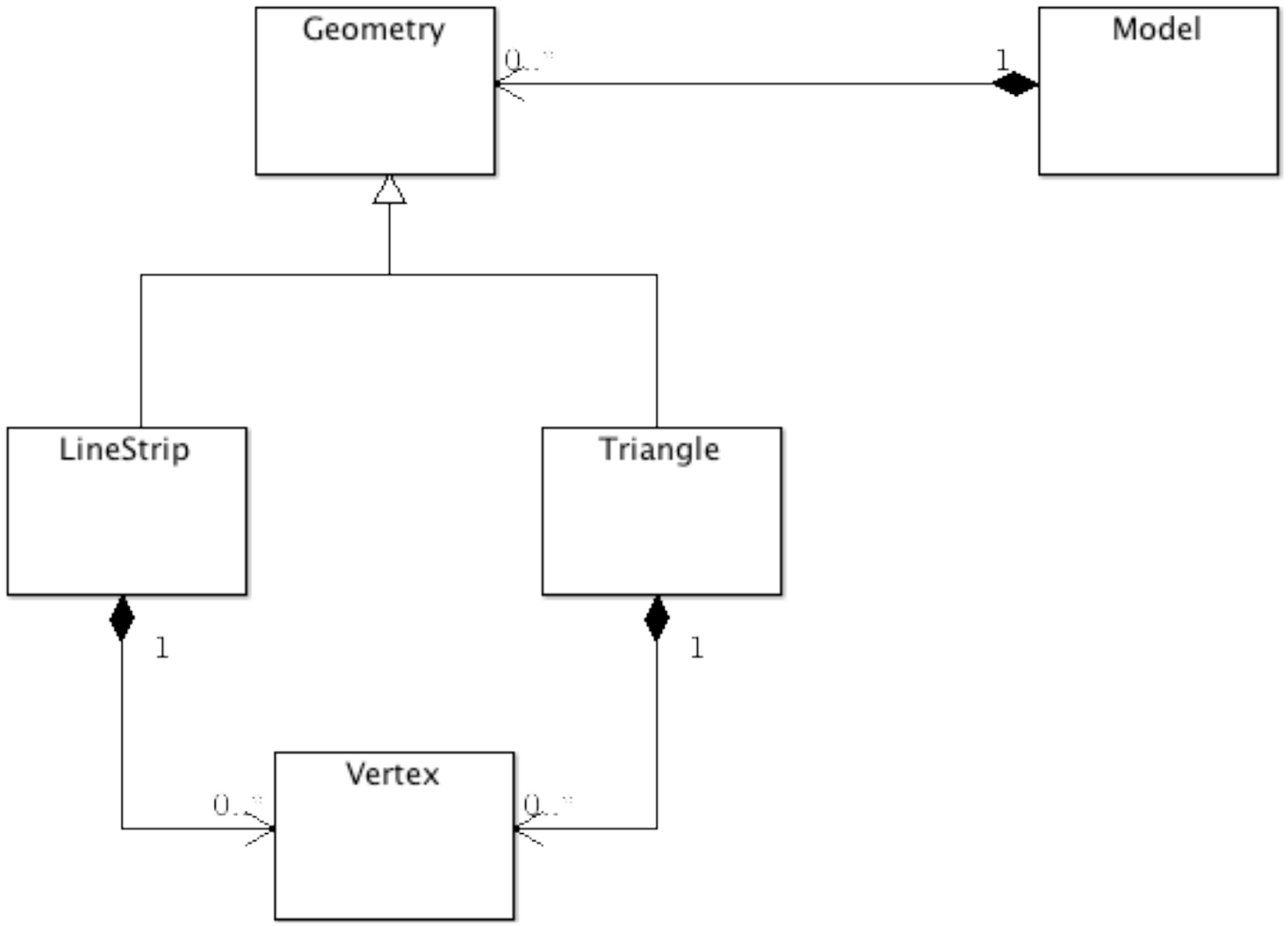
```
struct Model
{
  int maxX, maxY;
  vector<LineStrip> lines;
  vector<Triangle>  triangles;

  Model(istream& is);
  ~Model();
  void render();
};
```



```
void Model::render()
{
  for (unsigned int i = 0; i < lines.size(); i++)
  {
    lines[i].render();
  }
  for (unsigned int i = 0; i < triangles.size(); i++)
  {
    triangles[i].render();
  }
}
```




LineStrip
1
0..*
0..*
1
Vertex
Model
0..*
1
Triangle
1
0..*
Geometry
0..*
1
Model
LineStrip
Triangle
1
1
Vertex
0..*
0..*




```
struct Vertex
{
  float x;
  float y;

  Vertex(istream& is);
  void render();
};

struct Geometry
{
  virtual void render()=0;
};

struct LineStrip : public Geometry
{
  vector<Vertex> vertices;

  LineStrip(istream& is);
  void render();
};

struct Triangle : public Geometry
{
  Vertex p1, p2, p3;

  Triangle(istream& is);
  void render();
};

struct Model
{
  int maxX, maxY;
  vector <Geometry*> entities;

  Model(istream& is);
  ~Model();
  void render();
};
```

```
Model::Model(istream &is)
{
  int size;
  is >> maxX >> maxY;
  is >> size;
  for (int i = 0; i < size; i++)
  {
    int typeId;
    is >> typeId;
    Geometry *entity;
    switch (typeId)
    {
      case LineStripId: {
                          entity = new LineStrip(is);
                          entities.push_back(entity);
                          break;
                        }
      case TriangleId:  {
                          entity = new Triangle(is);
                          entities.push_back(entity);
                          break;
                        }
    }
  }
}
```

```
Model::~Model()
{
  for (unsigned int i=0; i<entities.size(); i++)
  {
    delete entities[i];
  }
}

void Model::render()
{
  for (unsigned int i=0; i<entities.size(); i++)
  {
    entities[i]->render();
  }
}
```