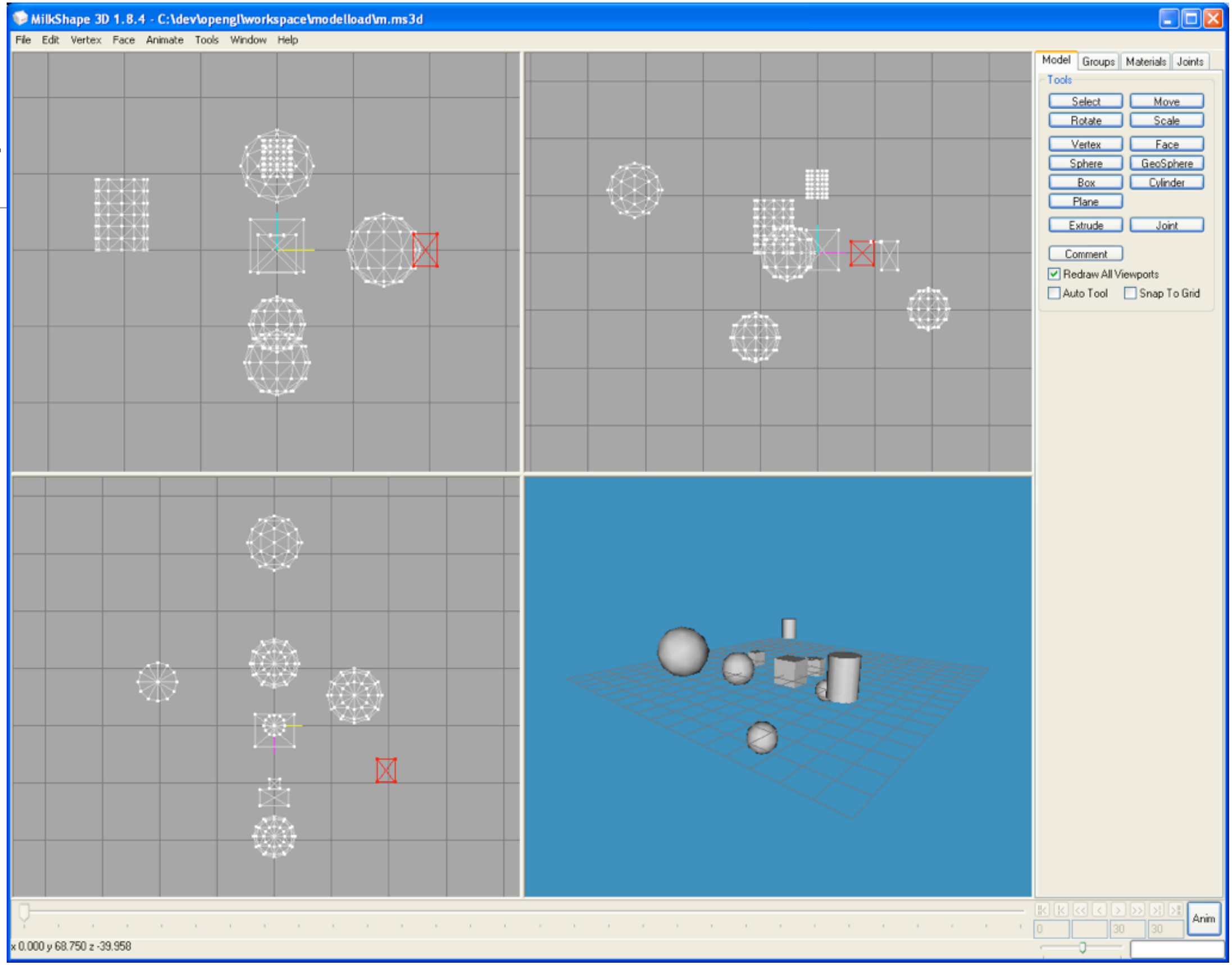
First Model Load / C++ Refresh

OpenGL

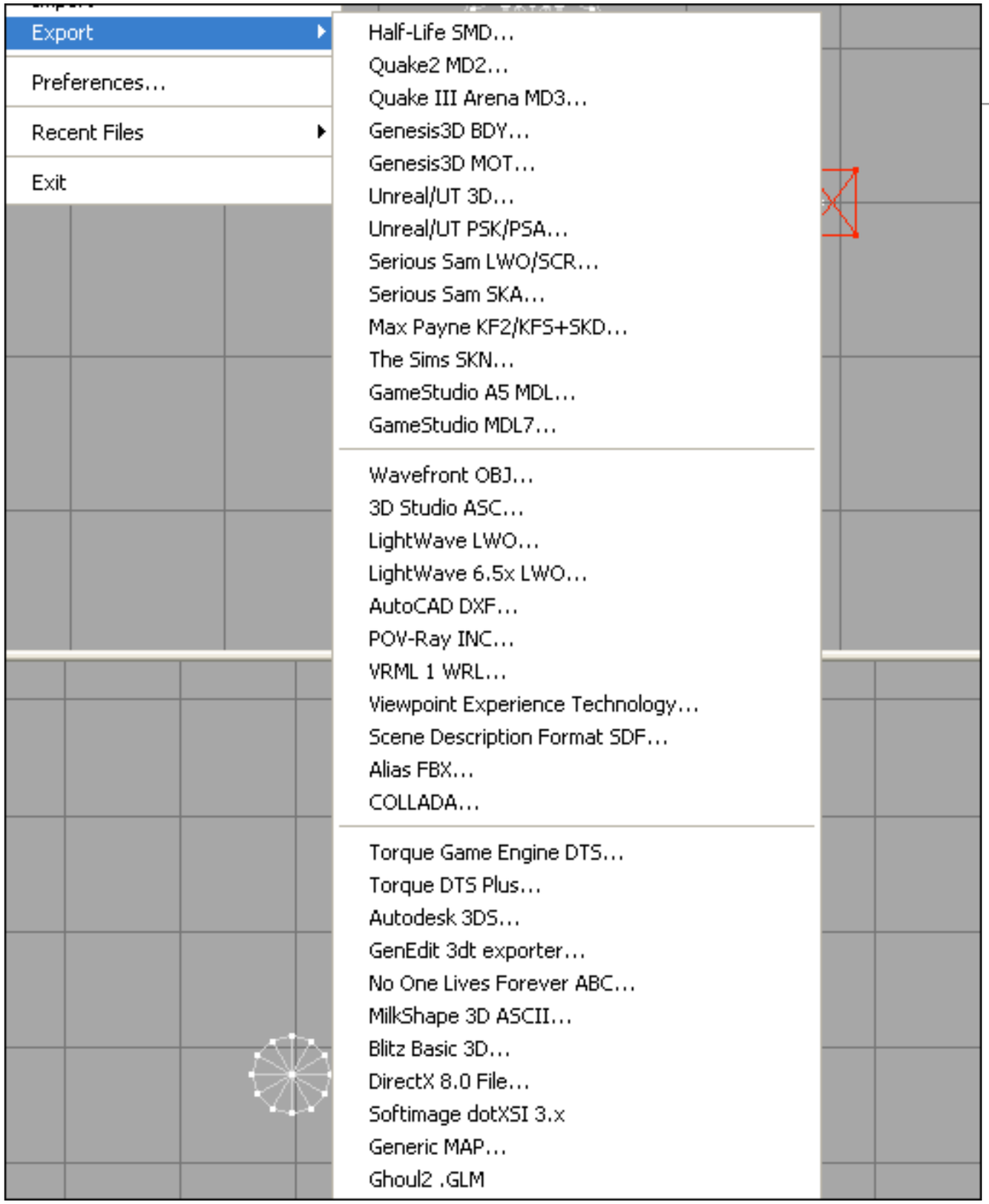
Learning Outcomes

- Review the need for a model
- Review C++ stream input classes
- Model 1: Define a simple data structure for loading a model
- Model 2: Employ STL
- Model 3: Encapsulate the data structure within a set of classes, allocating behaviors appropriately
- Model 4: Make file formal extensible

Model Creator



Model Format




```
# Wavefront OBJ exported by MilkShape 3D

mtllib t.mtl

v -11.500000 16.000000 13.000000
v -11.500000 -7.000000 13.000000
v 11.500000 16.000000 13.000000
v 11.500000 -7.000000 13.000000
v 11.500000 16.000000 -7.750000
v 11.500000 -7.000000 -7.750000
v -11.500000 16.000000 -7.750000
v -11.500000 -7.000000 -7.750000
# 8 vertices

vt 0.000000 1.000000
vt 0.000000 0.000000
vt 1.000000 1.000000
vt 1.000000 0.000000
# 4 texture coordinates

vn 0.000000 0.000000 1.000000
vn 1.000000 0.000000 0.000000
vn 0.000000 0.000000 -1.000000
vn -1.000000 0.000000 0.000000
vn 0.000000 1.000000 0.000000
vn 0.000000 -1.000000 0.000000
# 6 normals

g Box04
usemtl Material01
s 1
f 1/1/1 2/2/1 3/3/1
f 2/2/1 4/4/1 3/3/1
s 2
f 3/1/2 4/2/2 5/3/2
f 4/2/2 6/4/2 5/3/2
s 1
f 5/1/3 6/2/3 7/3/3
f 6/2/3 8/4/3 7/3/3
s 2
f 7/1/4 8/2/4 1/3/4
f 8/2/4 2/4/4 1/3/4
s 3
f 7/1/5 1/2/5 5/3/5
f 1/2/5 3/4/5 5/3/5
f 2/1/6 8/2/6 4/3/6
f 8/2/6 6/4/6 4/3/6
# 12 triangles in group

# 12 triangles total
```

Model Loader

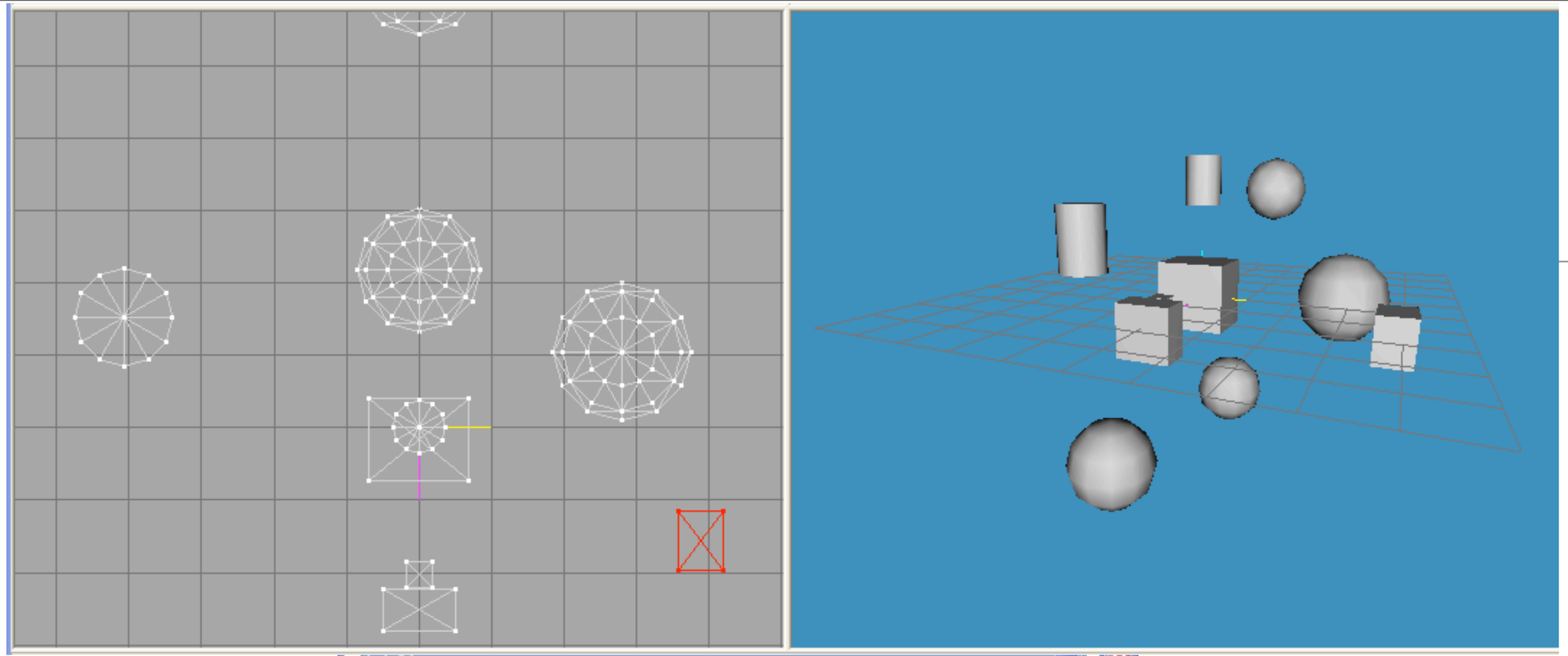
- Parse file format and load to internal data structures

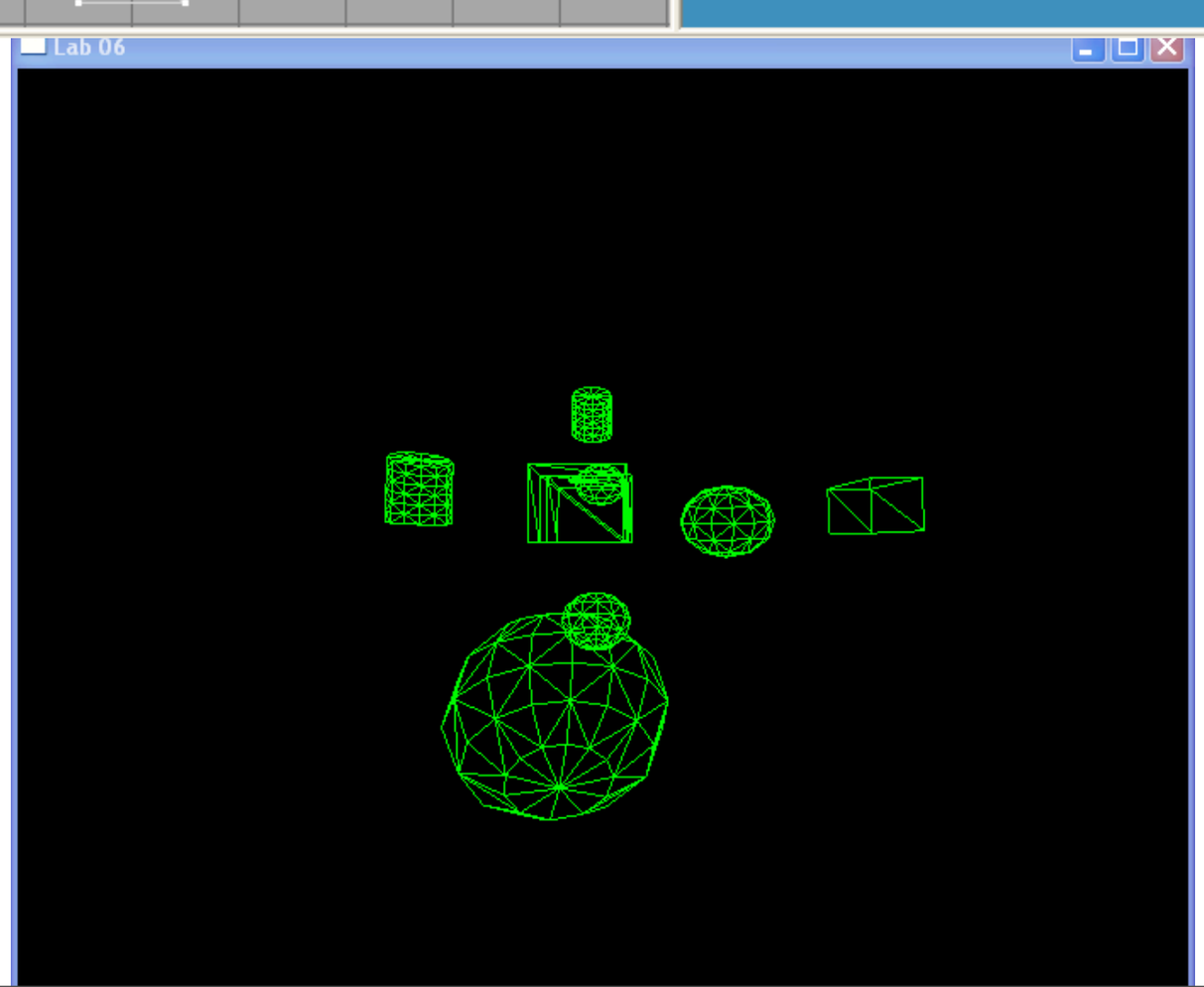
```
#define GLM_NONE     (0)            /* render with only vertices */
#define GLM_FLAT     (1 << 0)       /* render with facet normals */
#define GLM_SMOOTH   (1 << 1)       /* render with vertex normals */
#define GLM_TEXTURE  (1 << 2)       /* render with texture coords */
#define GLM_COLOR    (1 << 3)       /* render with colors */
#define GLM_MATERIAL (1 << 4)       /* render with materials */


/* GLMmaterial: Structure that defines a material in a model.
 */
typedef struct _GLMmaterial
{
  char* name;                       /* name of material */
  GLfloat diffuse[4];               /* diffuse component */
  GLfloat ambient[4];               /* ambient component */
  GLfloat specular[4];              /* specular component */
  GLfloat emmissive[4];             /* emmissive component */
  GLfloat shininess;                /* specular exponent */
//this is for textures
  GLuint IDTextura;         // ID-ul texturii difuze
} GLMmaterial;

/* GLMtriangle: Structure that defines a triangle in a model.
 */
typedef struct _GLMtriangle {
  GLuint vindices[3];               /* array of triangle vertex indices */
  GLuint nindices[3];               /* array of triangle normal indices */
  GLuint tindices[3];               /* array of triangle texcoord indices*/
  GLuint findex;                    /* index of triangle facet normal */
  //GLuint nrvecini;
  GLuint vecini[3];
  bool visible;
} GLMtriangle;

//adaugat pentru suport texturi
typedef struct _GLMtexture {
  char *name;
  GLuint id;                        /* ID-ul texturii */
  GLfloat width;      /* width and height for texture coordinates */
  GLfloat height;
} GLMtexture;

/* GLMgroup: Structure that defines a group in a model.
 */
typedef struct _GLMgroup {
  char*             name;           /* name of this group */
  GLuint            numtriangles;   /* number of triangles in this group */
  GLuint*           triangles;      /* array of triangle indices */
  GLuint            material;       /* index to material for group */
  struct _GLMgroup* next;           /* pointer to next group in model */
} GLMgroup;
```

Model Render

- Render the model using OpenGL

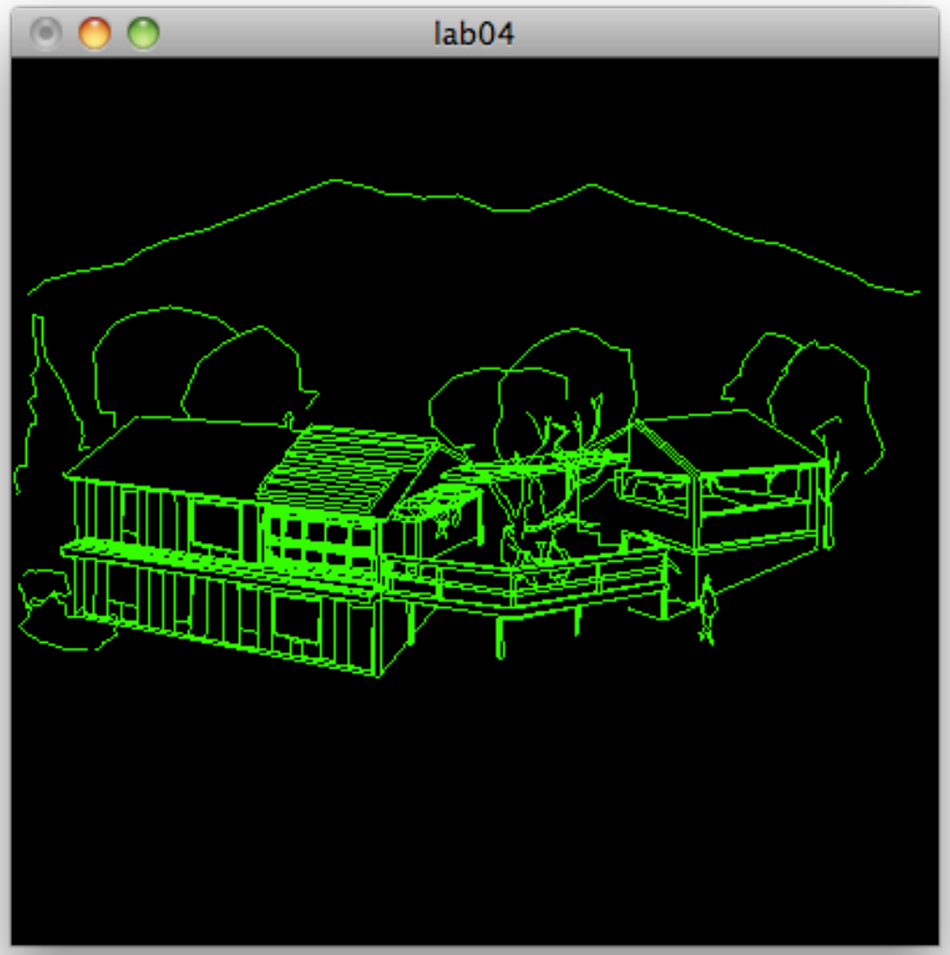
```
GLvoid glmDraw(GLMmodel* model, GLuint mode,char *drawonly)
{
    static GLuint i;
    static GLMgroup* group;
    static GLMtriangle* triangle;
    static GLMmaterial* material;
    GLuint IDTextura;

    assert(model);
    assert(model->vertices);

    /* do a bit of warning */
    if (mode & GLM_FLAT && !model->facetnorms) {
        printf("glmDraw() warning: flat render mode requested "
            "with no facet normals defined.\n");
        mode &= ~GLM_FLAT;
    }
    if (mode & GLM_SMOOTH && !model->normals) {
        printf("glmDraw() warning: smooth render mode requested "
            "with no normals defined.\n");
        mode &= ~GLM_SMOOTH;
    }
    if (mode & GLM_TEXTURE && !model->texcoords) {
        printf("glmDraw() warning: texture render mode requested "
            "with no texture coordinates defined.\n");
        mode &= ~GLM_TEXTURE;
    }
    if (mode & GLM_FLAT && mode & GLM_SMOOTH) {
        printf("glmDraw() warning: flat render mode requested "
            "and smooth render mode requested (using smooth).\n");
        mode &= ~GLM_FLAT;
    }
    if (mode & GLM_COLOR && !model->materials) {
        printf("glmDraw() warning: color render mode requested "
            "with no materials defined.\n");
        mode &= ~GLM_COLOR;
    }
    if (mode & GLM_MATERIAL && !model->materials) {
        printf("glmDraw() warning: material render mode requested "
            "with no materials defined.\n");
        mode &= ~GLM_MATERIAL;
    }
    if (mode & GLM_COLOR && mode & GLM_MATERIAL) {
        printf("glmDraw() warning: color and material render mode requested "
            "using only material mode.\n");
        mode &= ~GLM_COLOR;
    }
    if (mode & GLM_COLOR)
        glEnable(GL_COLOR_MATERIAL);
    else if (mode & GLM_MATERIAL)
        glDisable(GL_COLOR_MATERIAL);
    if (mode & GLM_TEXTURE) {
        glEnable(GL_TEXTURE_2D);
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    }
    /* perhaps this loop should be unrolled into material, color, flat,
       smooth, etc. loops?  since most cpu's have good branch prediction
       schemes (and these branches will always go one way), probably
```




Lab 06

loadAndDraw() invocation

- Load and Draw (render) combined in a single function




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

  loadAndDraw("bighouse.txt");

  glFlush();
}

void setupRC()
{
  glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

  glColor3f(0.0f, 1.0f, 0.0f);

  glOrtho(-1.0f, +1.0f, -1.0f, +1.0f, -1.0f, +1.0f);
}
```

loadAndDraw() Implementation

```
62
16
 -0.7209  0.1932
 -0.4061  0.1713
 -0.4082  0.1773
 -0.4103  0.1872
 -0.4082  0.1972
 -0.4019  0.1992
 -0.3956  0.1992
 -0.3935  0.1872
 -0.3956  0.1853
 -0.3956  0.1614
 -0.3599  0.1574
 -0.3494  0.1673
 -0.0766  0.1315
 -0.1921 -0.0398
 -0.8867  0.0578
 -0.7230  0.1972
10
 -0.3557  0.1693
 -0.4040  0.1136
 -0.4082  0.1076
 -0.1291  0.0697
 -0.0850  0.1295
 -0.0808  0.1415
 -0.1249  0.0797
 -0.4040  0.1155
 -0.3557  0.1733
 -0.0808  0.1415
```

- fstream is an standard library class to provide stream based access to a file on disk.

```
void loadAndDraw(char * fileName)
{
  fstream inStream;
  inStream.open(fileName, ios::in);
  if (inStream.fail())
    return;
  GLint numpolys, numLines;
  inStream >> numpolys;
  for (int j = 0; j < numpolys; j++)
  {
    inStream >> numLines;
    glBegin( GL_LINE_STRIP);
    for (int i = 0; i < numLines; i++)
    {
      float x, y;
      inStream >> x >> y;
      glVertex2f(x, y);
    }
    glEnd();
  }
  inStream.close();
}
```

1st Model

- Encapsulate fundamental data structure as a set of basic abstractions

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

struct LineStrip
{
  int size;
  Vertex *vertices;
};

struct Model
{
  int size;
  LineStrip* lineStrips;
};
```

```
62
16
 -0.7209  0.1932
 -0.4061  0.1713
 -0.4082  0.1773
 -0.4103  0.1872
 -0.4082  0.1972
 -0.4019  0.1992
 -0.3956  0.1992
 -0.3935  0.1872
 -0.3956  0.1853
 -0.3956  0.1614
 -0.3599  0.1574
 -0.3494  0.1673
 -0.0766  0.1315
 -0.1921 -0.0398
 -0.8867  0.0578
 -0.7230  0.1972
10
 -0.3557  0.1693
 -0.4040  0.1136
 -0.4082  0.1076
 -0.1291  0.0697
 -0.0850  0.1295
 -0.0808  0.1415
 -0.1249  0.0797
 -0.4040  0.1155
 -0.3557  0.1733
 -0.0808  0.1415
```

loadSimple() Implementation

```
Model loadSimple(char *fileName)
{
  Model model;

  fstream inStream;
  inStream.open(fileName, ios::in);
  if (inStream.fail())
    exit(1);
  inStream >> model.size;
  model.lineStrips = new LineStrip[model.size];
  for (int polyIndex = 0; polyIndex < model.size; polyIndex++)
  {
    inStream >> model.lineStrips[polyIndex].size;
    model.lineStrips[polyIndex].vertices = new Vertex[model.lineStrips[polyIndex].size];
    for (int vertexIndex = 0; vertexIndex < model.lineStrips[polyIndex].size;
                                                          vertexIndex++)
    {
      inStream >> model.lineStrips[polyIndex].vertices[vertexIndex].x
               >> model.lineStrips[polyIndex].vertices[vertexIndex].y;
    }
  }
  inStream.close();
  return model;
}
```

renderSimple() Implementation

- Essentially traversal of the data structure

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

struct LineStrip
{
  int size;
  Vertex *vertices;
};

struct Model
{
  int size;
  LineStrip* lineStrips;
};
```

```
void renderSimple(Model &model)
{
  for (int polyIndex=0; polyIndex < model.size; polyIndex++)
  {
    glBegin( GL_LINE_STRIP);
    for (int vertexIndex = 0; vertexIndex < model.lineStrips[polyIndex].size;
                                                              vertexIndex++)
    {
      glVertex2f(model.lineStrips[polyIndex].vertices[vertexIndex].x,
                                 model.lineStrips[polyIndex].vertices[vertexIndex].y);
    }
    glEnd();
  }
}
```

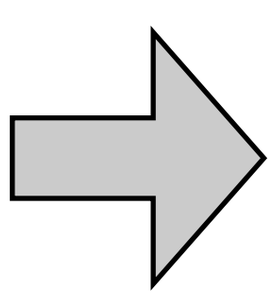
Using the model

```
Model model;

void renderScene(void)
{
  glClear( GL_COLOR_BUFFER_BIT);

  renderModel(model);

  glFlush();
}

void setupRC()
{
  glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

  glColor3f(0.0f, 1.0f, 0.0f);

  glOrtho(-1.0f, +1.0f, -1.0f, +1.0f, -1.0f, +1.0f);
  model = loadModel("bighouse.txt");
}
```

2nd Model: Simplifying the Abstractions

- Replace build in arrays with STL Vectors:

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

struct LineStrip
{
  int size;
  Vertex *vertices;
};

struct Model
{
  int size;
  LineStrip* lineStrips;
};
```



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

struct LineStrip
{
  vector<Vertex> vertices;
};

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

Revised loadModel()

```
Model loadModel(char *fileName)
{
  Model model;

  fstream inStream;
  inStream.open(fileName, ios::in);
  if (inStream.fail())
    exit(1);
  int numberPolys;
  inStream >> numberPolys;
  for (int polyIndex = 0; polyIndex <  numberPolys; polyIndex++)
  {
    LineStrip LineStrip;
    int numberVertices;
    inStream >> numberVertices;
    for (int vertexIndex = 0; vertexIndex < numberVertices; vertexIndex++)
    {
      Vertex vertex;
      inStream >> vertex.x >> vertex.y;
      LineStrip.vertices.push_back(vertex);
    }
    model.lineStrips.push_back(LineStrip);
  }
  inStream.close();
  return model;
}
```

```
62
16
 -0.7209  0.1932
 -0.4061  0.1713
 -0.4082  0.1773
 -0.4103  0.1872
 -0.4082  0.1972
 -0.4019  0.1992
 -0.3956  0.1992
 -0.3935  0.1872
 -0.3956  0.1853
 -0.3956  0.1614
 -0.3599  0.1574
 -0.3494  0.1673
 -0.0766  0.1315
 -0.1921 -0.0398
 -0.8867  0.0578
 -0.7230  0.1972
10
 -0.3557  0.1693
 -0.4040  0.1136
 -0.4082  0.1076
 -0.1291  0.0697
 -0.0850  0.1295
 -0.0808  0.1415
 -0.1249  0.0797
 -0.4040  0.1155
 -0.3557  0.1733
 -0.0808  0.1415
```

Revised render()

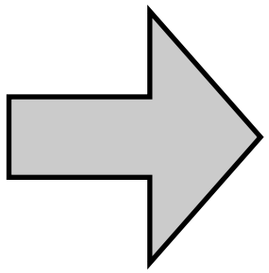
```
struct Vertex
{
  float x;
  float y;
};

struct LineStrip
{
  vector<Vertex> vertices;
};

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

```
void renderModel(Model &model)
{
  for (unsigned int polyIndex=0; polyIndex < model.lineStrips.size(); polyIndex++)
  {
    glBegin( GL_LINE_STRIP);
    for (unsigned int vertexIndex = 0;
              vertexIndex < model.lineStrips[polyIndex].vertices.size(); vertexIndex++)
    {
      glVertex2f(model.lineStrips[polyIndex].vertices[vertexIndex].x,
                                model.lineStrips[polyIndex].vertices[vertexIndex].y);
    }
    glEnd();
  }
}
```

3rd Model:Encapsulating Behaviour

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

struct LineStrip
{
  int size;
  Vertex *vertices;
};

struct Model
{
  int size;
  LineStrip* lineStrips;
};
```

- Constructor loads data from stream
- render invokes opengl methods



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

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

struct LineStrip
{
  vector<Vertex> vertices;

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

struct Model
{
  vector<LineStrip> lineStrips;

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

Vertex & LineStrip

```
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

- Constructor assumes stream already successfully opened

```
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();
  }
}
```

Using the Model

```
Model *model;

void renderScene(void)
{
  glClear( GL_COLOR_BUFFER_BIT);

  model->render();

  glFlush();
}

void setupRC()
{
  glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

  glColor3f(0.0f, 1.0f, 0.0f);

  glOrtho(-1.0f, +1.0f, -1.0f, +1.0f, -1.0f, +1.0f);

  fstream modelStream;
  modelStream.open("bighouse.txt", ios::in);
  if (modelStream.fail())
    exit(1);
  model = new Model(modelStream);
}
```

4th Model Enhancing the Model File Format

- Comments to make file readable
- Scale to influence projection
- Line strips explicitly flagged (with 0)
- Room for extensions (eg triangles)

```
#scale
150 150
#number of entities
2
# 1st entity
# 0 = line strip
0
# number of vertices
6
# vertices
40 40
40 90
70 120
100 90
100 40
40 40
# 2nd entity
# 0 = line strip
0
# number of vertices
4
# vertices
50 100
50 120
60 120
60 110
```

Skipping Comments

- Keep reading full lines if they start with #
- As soon as a line not starting with # is encountered, put the (just read) character back in the read buffer.

```
void skipComment(istream &is)
{
  char ch;
  is >> ch;
  if (ch == '#')
  {
    do
    {
      string buf;
      getline(is, buf);
      is >> ch;
    } while (ch == '#');
  }
  is.putback(ch);
}
```

Revised Vertex & LineStrip Constructors

- Every time we are about to read, call skipComment() to move passed any comments and on to the actual data

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

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

Extend Model

- New attributes for max X and Y values

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

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

Model Constructor

- Read max X & Y
- Only load a LineStrip if its type is explicitly present

```
#scale
150 150
#number of entities
2
# 1st entity
# 0 = line strip
0
# number of vertices
6
# vertices
40 40
40 90
70 120
100 90
100 40
40 40
```

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

    }
  }
}
```

Extension: Triangle

- Code 1 = LineStrip
- Code 2 = Triangle

```
#scale
150 150
#number of entities
2
# 1st entity
# 0 = line strip
0
# number of vertices
6
# vertices
40 40
40 90
70 120
100 90
100 40
40 40
# 2nd entity
# 0 = line strip
0
# number of vertices
4
# vertices
50 100
50 120
60 120
60 110
# 3rd entity
# 1 = triangle
1
# 3 vertices
20 10
100 22
22 22
# 1 = triangle
1
# 3 vertices
120 10
10 22
102 22
```

Triangle Extension

- Define new class Triangle with
  - Constructor
  - render method
- Introduce new Triangle vector in Model
  - Load this vector with triangles as encountered in file
  - Modify render method to also traverse and render this vector