VIEWER COMPONENTS

The software has five major components:

- A simple Viewer Application that allows you to open and view OBJ files. This viewer is extensible via plugins implementing a simple interface. The idea is that solving an exercise involves writing a plugin. Testing the plugin involves running the viewer and loading the plugin to check its behavior.

- A Core Library that provides classes to represent a 3D scene: Scene, Object, Face, Vertex, Camera... This library is used by the application and your plugins.

- A GLWidget Library containg a GLWidget class. The GLWidget class has little OpenGL code. Instead, most of the GLWidget implementation is devoted to enable users to load an arbitrary number of plugins that provide the typical functionalities of a 3D application: setting up the OpenGL state, loading shaders, drawing the scene by issuing OpenGL rendering calls, and enabling some user interaction (object selection, camera control...).

- A Plugin Interface that provides the base class for all plugins. A default (do nothing) implementation is provided for all methods, so most plugins need to override very few methods (typically onPluginLoad(), onObjectAdd(), and a subset of preFrame(), postFrame, drawScene() and paintGL(). Within a plugin, you can use the scene() and camera() methods to access the Scene and the Camera. You can also access the GLWidget with glwidget().

- Some Plugin examples that provide basic functionality for the viewer.

CORE LIBRARY

INTRODUCTION

The core library contains a collection of classes for representing 3D scenes.
A **Scene** represents a collection of 3D objects.
Each 3D **Object** contains a collection of vertices and a collection of faces.
A **Vertex** stores per-vertex attributes (such as vertex coordinates).
A **Face** is a collection of vertex indices. Faces do not hold directly vertex coordinates; instead, faces store vertex indices, that is, integers indicating the position of each vertex in the vector of vertices associated with every 3D Object.
The library also provides a simple **Camera** class, and basic math classes for representing **Points** and **Vectors** in 3D space.

BOX

The Box class represents a box in 3D space through its (min, max) corners. This class is suitable for representing axis-aligned boxes (but not oriented boxes). A typical use is for representing the bounding box of an object or group of objects.

#include <box.h>

Public Member Functions

	Box (const Point &point=Point())
	Box (const Point &minimum, const Point &maximum)
void	expand (const Point &p)
void	expand (const Box &b)
void	render ()
Point	center () const
float	radius () const

CAMERA

The Camera class represents a perspective camera.

#include <camera.h>

Public Member Functions

void	init (const Box &b)
void	setModelview () const
void	setProjection () const
Point	getObs () const
void	setAspectRatio (float ar)
void	updateClippingPlanes (const Box &)
void	incrementDistance (float inc)
void	incrementAngleX (float inc)
void	incrementAngleY (float inc)
void	pan (const Vector &offset)

FACE

The Face class represents a face of a 3D object. Each face keeps a list of vertices (actually vertex indices) and a normal vector. The face is assumed to be convex.

#include <face.h>

Public Member Functions

	Face ()
	Face (int i0, int i1, int i2, int i3=-1)
void	addVertexIndex (int i)
int	numVertices () const
int	vertexIndex (int i) const
Vector	normal () const
void	computeNormal (const vector< Vertex > &verts)

OBJECT

The Object class represents a 3D object. Each object consists of a collection of vertices and a collection of faces. Objects also store a bounding box.
#include <object.h>

Public Member Functions

	Object (std::string name)
void	readObj (const char *filename)
Box	boundingBox () const
const vector< Face > &	faces () const
const vector< Vertex > &	vertices () const
vector< Vertex > &	vertices ()
void	computeNormals ()
void	computeBoundingBox ()

POINT

The Point class represents a point (x,y,z) in 3D space. For the sake of simplicity, Point is implemented as a typedef of Qt's QVector3D.
#include <point.h>

Example of use:

```
#include "point.h"
#include "vector.h"
// create two points
Point p = Point(0.0, 0.0, 0.0);
Point q = Point(1.0, 0.0, 0.0);
// get coordinate values
float x = p.x();
// set coordinate values
p.setX(2.0);
// common operations
Vector v = p - q; // point substraction
Point r = 0.4*p + 0.6*q; // barycentric sum
```

SCENE

The Scene class represents a 3D scene as a flat collection of 3D objects. A scene is basically a vector of 3D objects.
#include <scene.h>

Public Member Functions

	Scene ()
const vector< Object > &	objects () const
void	addObject (Object &)
int	selectedObject () const
void	setSelectedObject (int index)
void	computeBoundingBox ()
Box	boundingBox () const

VECTOR

The Vector class represents a vector (x,y,z) in 3D space. For the sake of simplicity, Vector is implemented as a typedef of Qt's QVector3D.

#include <vector.h>

Example of use:

```
#include "point.h"
#include "vector.h"
// create two vectors
Vector u = Vector(0.0, 0.0, 0.0);
Vector v = Vector(1.0, 0.0, 0.0);

// get components
float x = u.x();

// set components
u.setX(2.0);

// get length
float len = u.length();

// normalize (in place)
u.normalize();

// get normalized copy
v = u.normalized();

// common operations
Vector w;
w = u + v; // vector addition
w = u - v; // vector substraction
w = 2.0*u; // scalar multiplication
float dot = dotProduct(u, v);  // dot product
w = crossProduct(u, v);       // cross product
```

VERTEX

The Vertex class represents a vertex with a single attribute (vertex coordinates).
#include <vertex.h>

Public Member Functions

	Vertex (const Point &coords)
Point	coord () const
void	setCoord (const Point &coord)

GLWIDGET LIBRARY

INTRODUCTION

The *GLWidget library* contains a single class: *GLWidget* . The main purpose of this class, which is derived from *QGLWidget*, is to provide a very basic implementation of the well-known methods *initializeGL()* , *paintGL()* and *resizeGL()*.

GLWidget has little OpenGL rendering code. Instead, GLWidget loads an arbitrary number of *plugins* that provide the typical functionalities of a 3D application. The *viewer application* makes use of this library. Indeed, the *plugins* you will create will also require using this library to get access to important objects such as the 3D scene and the camera.

Overview

The GLWidget class holds basically three different pieces of information: a **scene**, a **camera**, a list of **loaded plugins.**
Most of the code in GLWidget deals with invoking appropriate methods from the plugins:

- Everytime a new plugin is loaded, its **onPluginLoad**() method is called.
- Everytime a new object is added to the scene, the **onObjectAdd**() method of all loaded plugins is invoked.
- The GLWidget::paintGL() method performs three basic steps: 1) call **preFrame**() for all plugins, 2) call **paintGL**() for the plugin that implements it, and 3) call **postFrame**() for all plugins.
- Mouse and keyboard events (***keyPressEvent*** and so on) are propagated to all loaded plugins.

GLWIDGET

The GLWidget class handles OpenGL rendering through plugins.
#include <glwidget.h>

Public Slots & Member functions

void	addObject ()
void	addObjectFromFile (const QString &filename)
void	resetCamera ()
void	drawAxes () const
Box	boundingBoxIncludingAxes () const
void	loadPlugin ()
void	loadPlugins (const QStringList &list)
void	loadDefaultPlugins ()
	GLWidget (QWidget *parent)
Scene *	scene ()
Camera *	camera ()
void	setPluginPath (const QString &)

PLUGIN INTERFACE

INTRODUCTION

The *Plugin interface* contains a single class: *BasicPlugin* . All viewer plugins must derive from this class.
A default implementation is provided for all methods, so most plugins need to override very few methods, e.g. onPluginLoad(), onObjectAdd(), and a subset of preFrame(), postFrame, drawScene() and paintGL().
Within a plugin, you can use the scene() and camera() methods to access the *Scene* and the *Camera* .

BASICPLUGIN

#include <basicplugin.h>

Public Member Functions

	BasicPlugin ()
virtual void	onPluginLoad ()
virtual void	onObjectAdd ()
virtual void	preFrame ()
virtual void	postFrame ()
virtual void	keyPressEvent (QKeyEvent *)
virtual void	keyReleaseEvent (QKeyEvent *)
virtual void	mouseMoveEvent (QMouseEvent *)
virtual void	mousePressEvent (QMouseEvent *)
virtual void	mouseReleaseEvent(QMouseEvent*)
virtual void	wheelEvent (QWheelEvent *)
virtual bool	paintGL ()
virtual bool	drawScene ()
Scene *	scene ()
Camera*	camera ()
BasicPlugin*	drawPlugin ()
GLWidget *	glwidget ()