INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Design of 3D Graphic Editor

Zhaoxia Liu

A Major Report
In
The Department
Of
Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

August 2002

Abstract

Design of 3D Graphic Editor

Zhaoxia Liu

In this report, the design of a 3D graphic editor is provided with object-oriented technology and OpenGL GLUT framework. The functionalities and features in the system of this project contain documenting 3D graphic objects, dynamically creating multiple windows and subwindows, and manipulating OpenGL features, such as, lighting, colors, solid and wire mesh states. A hierarchical data structure is built to enable import and export assembled object data. The application of building graphic objects shows that the system performs its functionalities.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisor, Prof. Peter Grogono, for his kind agreement to my initial motivation and proposal. Subsequently his enthusiastic support and valuable guidance gave me an excellent chance to explore the state of the art technology for this project. Without his kind support I could not finish it.

Second I would like to express my sincere thanks to my partner, Mr. Shuli Yang, for his contribution to this project -- described in the Implement of 3D Graphics Editor.

Finally, I would like to thank all people who provided help and support for my project.

Table of contents

Chapter 1: Introduction

OpenGL has widely been used to implement computer graphics and animation applications [PG98], CAD engineering application, game development, virtual physical reality and real-time visual systems [RW96]. It is powerful for rendering computer graphics in various system platforms. However, the libraries in OpenGL are written by C language and not object-oriented (OO). Object-Oriented technology has dominated industry software development for many years, OO technology gives many benefits to human to develop a software product, and it contains effective methodologies to build software products, such as modeling, analysis, design and implementation. Therefore, in this project, some functions in OpenGl are realized by OOP so that the functions can be easily reused, inherited and modified. The C++ is a popular OO Language and is chosen to implement the project.

This project is designed as a system of computer graphics editor for creating and editing 3D graphics, creating multiple windows and performing OpenGL features. In this report, we present the design of a 3D graphics editor. The system design is based on object-oriented techniques, and it is implemented by C++ language [HS98] and documented in the report [SLY]: "The Implementation of 3D Graphics Editor".

1.1 Project Aim

The project is to develop 3D computer graphic editor tools with multiple windows. This system is an OpenGL framework application, and it is implemented with object-oriented technology, that is, the system analysis, architecture, design and implementation follow this technology. As we consider the system emphasizing graphics edition, we have had

the functionalities to import and export the graphics object data. Therefore, some primitive 3D graphics components are built, and their shapes and positions can be changed so that complex 3D scenes can be built conveniently.

1.2 Project Tasks

The project tasks consist of the technologies used and the functionalities implemented.

- The project will be designed and implemented by using object-oriented technology and object-oriented programming language, C++.
- The system of the project is based on OpenGL API.
- The system is able to create and close windows dynamically.
- Ten solid primitive graphics components are built.
- Ten wire primitive graphics components are built.
- The system is able to import and export graphics object data files.
- The system can perform graphics features, such as, lighting, transformations, etc.
- Combining with the above functionalities, the system realizes its function to edit 3D computer graphics.

1.3 The Organization of The Project

This project is for building a 3D graphics editor. It is designed and implemented by using object-oriented technology and C++ language with OpenGL API. The user can use the application to create and edit 3D graphics objects and view the objects from multiple windows. The graphics objects can be created with the assemblies in a hierarchical

structure. The consideration of the project is to present an Object-Oriented approach to build the composite and complex graphical objects by using the primitive graphics components. We classify the OpenGL, GLU, GLAUX and GLUT functionalities on various aspects in the system of the project, such as GLUT window functions, GLU view function, AUX for 3D graphics representation, and OpenGL basic functions for global environment features and individual object features, such as, lighting, color, transformation, etc.

Chapter 2: Object-Oriented Technology

Object-oriented technology can roughly be classified to object-oriented modeling, object-oriented design and object-oriented implementation. It provides a practical, productive way to develop software for most practical application projects, and it makes software products that are easily reused and maintained. Traditional software development is mainly about functions and procedures, and it is not concerned with real world object structure. Object-oriented technology presents a powerful approach to real world objects, such as object concepts, attributes and behaviors, and makes software development more objective and effective.

2.1 Object-Oriented Modeling

Modeling is a critical part in software development [BRJ99]. OO technology can control a complexity of large-scale software system because of the modeling principle. The principle of modularity ensures that a complex and large system should be decomposed into many modules, in which a tight cohesion between components in a module and a loose coupling between modules has to be ensured. According to the modeling principle, we break a complex and large system into a set of modules so that each module is relatively small and simple, and the interactions among modules are relatively simple.

Object-oriented modeling is more concerned with aspects of the entire system and also the components used for building systems. Object-oriented modeling contains structural modeling, behavioral modeling and architectural modeling.

2.2 Object-Oriented Design

From a design point of view, there are several aspects, data abstraction, encapsulation, inheritance and polymorphism, to be considered with object-oriented technology [BRJ99]. The processes for working on object-oriented design are the analysis of software requirements, system architecture design for the whole system, subsystem design and detail design for the individual components.

- Abstraction models the system and consists of making the essential, inherent aspects of an entity, and it separates the essential from the nonessential characteristics of an entity. Abstraction classifies and groups the data of the system into the objective type; the result is a simpler but sufficiently accurate approximation of the original entity, obtained by removing or ignoring the nonessential characteristics.
- Encapsulation realize information hiding, a client doesn't need know how a service is done but the service contract while using the service. If the clients know nothing beyond the contractual interface, implementation can be modified without affecting the clients, so long as the contractual interface remains the same. Tightly cohesive components should be encapsulated as a module, and a loose coupling between modules is necessary.

- Inheritance is also the essential part to build a system with object-oriented technology, and it reuses the system components in a hierarchical structure. It gets the encapsulation of data and behaviors to be more refined in multiple levels. Inheritance

catches up with the real world object feature and makes object-oriented technology to be better understandable for the software development of the software industry.

- Polymorphism means that the same operation may behave differently on different modules and two operations have the same syntax in one module.

2.3 Object-Oriented Implementation

Object-oriented programming (OOP) dominates software development in the software industry [BRJ99]. The reason for this is that the real software projects are becoming more and more complex and large, and method of procedure programming cannot fit for such software, but OOP provides an organizational method for developing the complex and large computer systems. The framework in object-oriented design and programming is a new technique for developing extensible systems. It makes it easier to design reliable and reusable application system. Such frameworks can also improve the documentation and maintenance of existing systems.

Object-oriented program has the facilities to develop a modern software product.

- Object-oriented programs tend to be written in terms of real-world objects, not internal data structures. This makes them somewhat easier to understand by maintainers and the people who have to read your code -- but it may make it harder for you as the initial designer. Identifying objects in a problem is a challenge.

- Object-oriented programs encourage *encapsulation* -- details of an objects implementation are hidden from other objects. This keeps a change in one part of the program from affecting other parts, making the program easier to debug and maintain.

- Object-oriented programs encourage modularity. This means that pieces of the program do not depend on other pieces of the program. Those pieces can be reused in future projects, making the new projects easier to build.

  Corresponding to the object-oriented design, OOP in C++ language also has the features of abstraction, encapsulation, inheritance and polymorphism.

a. In OOP, we use *struct* and *class* to define the abstraction types. It aggregates the data to be one data type and builds the type to be reused conveniently.
b. Encapsulation is the basic feature to implement OOP codes. We realize the implementation of software product in module form. Its mechanism binds together code and the data it manipulates, and keeps both safe from outside interference and misuse. When code and data are linked together in this fashion, an object is created. In other words, an object is the device that supports encapsulation. Within an object, code, data, or both may be private to that object or public. Private code or data is known to and accessible only by another part of the object. That is, private code or data may not be accessed by a piece of the program that exists outside the object. When code or data is public, other parts of program can access it even though it is defined within an object. Typically, the public parts of an object are used to provide a controlled interface to the private elements of the object.
c. Inheritance is the process by which one object can acquire the properties of another object. This is important because it supports the concept of classification. If we think about it, most knowledge is made manageable by hierarchical classifications. For example, a red apple is part of the classification apple, which in turn is part of fruit

class, which is under the large class food. Without the use of classifications, each object would have to define explicitly all of its characteristics. However, through the use of classifications, an object need only define those qualities that make it unique within its class. It is the inheritance mechanism that makes it possible for one object to be a specific instance of a more general case. Inheritance is an important aspect of object-oriented programming.

d. Object-oriented programming languages support polymorphism. Polymorphism is the ability by which a method can be executed in more than one way, depending on some arguments and returns. When a client class sends a message, the client class doesn't need to know the class of the receiving instance. The client class for a specific event only provides a request, while the receiver knows how to perform this event. The polymorphism characteristic sometimes makes it uncertain at compile time, to determine which class an instance belongs to and thus to decide which operation to perform. Polymorphism allows a programmer to provide the same interface to different objects.

Chapter 3: OpenGL API

OpenGL is not a programming language. It is a C runtime library, which provides some prepackaged functionality. We classify OpenGL API into the OpenGL, GLU, GLAUX and GLUT functionalities on various aspects in the system of the project, such as, GLUT window functions, GLU view function, AUX for 3D graphics representation [MJT99].

3.1 GL Library

OpenGL GL library provides the basic functionalities for displaying and manipulating 2D and 3D graphic objects, such as, defining light, glLight(); object color, glColor(); and transformations, glTranslate(), glRotate() and glScale().

- OpenGL data types

  To make OpenGL code more portable for various platforms, OpenGL defines its own data types. These data types map to normal C language data types that people can use instead, if they want [RW96].

OpenGL Data Type	Internal Representation	Defined as C type
Glbyte	8-bit integer	Signed char
Glshort	16-bit integer	Short
Glint, Glsizi	32-bit integer	Long
Glfloat	32-bit floating	Float
Gldouble	64-bit floating	Double

Glubyte	16-bit unsigned Integer	Unsigned char
Glboolean	16-bit unsigned Integer	Unsigned char
Glushort	32-bit unsigned Integer	Unsigned short
Gluint, Glenum	32-bit unsigned Integer	Unsigned long
Glbitfield	32-bit unsigned Integer	Unsigned long

- Function naming convention

OpenGL functions follow following naming convention:

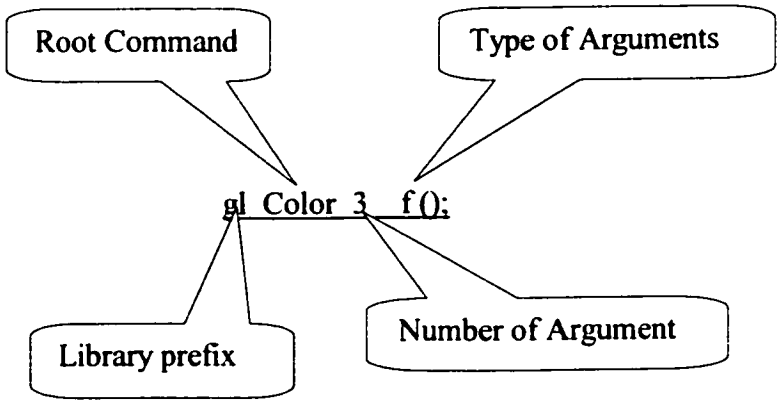
a. First which library the function - library prefix

b. Second all functions have a root – root command

c. Third pair number to specify – number of arguments

d. Fourth pair type to specify – type of arguments

For example,



- GL library functionalities

The OpenGL GL library has the basic graphic environment and display functions:

a. Lighting item: enable light, call glEnable(GL_LIGHTING) and glEnable(GL_LIGHT0), and set the light properties, such as, ambient, diffuse, specular, spot light parameters.
b. Transformations: defining translation, rotation and scale.
c. Specifying the functionalities for geometric rendering.
d. Setting window view port.
e. Setting window client environments, such as, object color, background color and other attributes.
f. Texture mapping process command functions.
g. OpenGL text displaying.
h. Clearing buffers, such as, color, depth, accumulation and stencil.

3.2 GLU Library

GLU is the OpenGL Utility Library. It is an extension of OpenGL and supports higher-level operations. GLU functions are used to manipulate the transformation matrices of model, view and projection, surface tessellations, quadratic surface rendering and etc.

- Mipmapping and image scaling with the functions, gluBuild1Dmipmaps(), gluBuild2Dmipmaps() and gluScaleImage()
- Matrix transformations, gluOrth2D(), gluPerspective(), gluLookAt(), gluProject(), gluUnProject() and etc.
- Nurbs surfaces and polygon tessellations, gluNurbsSurface(), gluTessCallback(), and etc.

- Quadratic surfaces, gluCylinder(), gluShpere(), gluDisk(), and etc.

3.3 GLUT Library

GLUT (pronounced like the glut in gluttony) is the OpenGL Utility Toolkit, a window system independent toolkit for writing OpenGL programs [RW96]. It is an extension of OpenGL and implements a simple windowing application-programming interface (API) for OpenGL. GLUT makes it considerably easier to learn about and explore OpenGL programming. GLUT provides a portable API so you can write a single OpenGL program that runs on several platforms. The following sections describe the features provided by GLUT.

3.3.1 Initialize and Create a Window

For GLUT window management, you must specify the window environment characteristics, single-buffered or double-buffered, and the color types, RGGA or color indices. The functions of GLUT window application can be used to run the application.

- Initialize GLUT library, first call the function, glutInit(). The function also processes command line options.
- Specify the buffer state, call the function, glutDisplayMode() to setup the display mode, single-buffered or double-buffered.
- Create window by calling glutCreateWindow() or glutCreateSubWindow() to open a window or a subwindow in its parent window. Before creating a window, you can set the window size and location by calling glutInitWindowSize() and glutWindowPosition().

- Further manipulation by calling glutGetWindow(), glutSetWindow(), glutSetWindow Title(), glutSetIconTitle(), glutPositionWindow(), glutReshapeWindow(), glutPop Window(), glutPushWindow(void), glutIconifyWindow(), glutShowWindow(), glutHideWindow().

3.3.2 Handle Window and Input Events

For GLUT framework application system to interact with the user, it handles the window and input events by using the callback functionalities.

- The glutDisplayFunc() procedure is the first and most important event callback function. A callback function is one where a programmer-specified routine can be registered to be called in response to a specific type of event.
- Window resizing and moving, we need to call the function, glutReshapeFunc() to reset the viewport.
- Mouse event by calling glutMouseFunc() and glutMotionFunc() to handle the mouse events.
- For keyboard events, the system calls the function, glutKeyboardFunc() to handle to keyboard key events, and glutSpecialFunc() to handle keyboard special key events.
- Redraw window graphics, the system always uses the function, glutPostRedisplay().

3.3.3 Load the Color Map

GLUT provides a routine to load a single index color with an RGB value, the system uses the function, glutSetColor().

3.3.4 Initialize and Draw 3D Graphic Objects

For drawing some primitive graphic objects, GLUT library provides some objects, such as, cube, cone, sphere, torus, teapot, and etc.

3.3.5 Manage a Background Process

For making an animation by a background process, the function glutIdleFunc() registers a callback function provided by the user.

3.3.6 Run the program

GLUT enters a loop in which events are processed as they occur until the user signals that the program is to be terminated.

Chapter 4: System Design

For the application of OpenGL framework, the system is designed by View-Document-Control (from MFC – Microsoft Foundation Class) architecture and three classes of items are considered to describe the system: constants, structures and classes. The OpenGL API is classified based on its functions. The system will be implemented by C++ language and designed by using object-oriented technology. Base classes are built for user to create further functionality. In order to create multiple window objects in the GLUT framework application dynamically, we create a class to contain all the GLUT callback functions, as we should put the callback functions as static member functions.

4.1 System Architecture

The system design is object-oriented. The internal functionalities are implemented with OpenGL API. For setting window, OpenGL GLUT API functions are used to GLUT Framework; OpenGL GLU API functions are used to set model view transformation; OpenGL GL API functions are used to implement the basic functionalities; OpenGL Glaux API functions are used to create primitive objects.

4.1.1 System Design Strategy

The main functionalities of the system are as follows: create multiple windows including windows and subwindows; documenting graphic objects including editing and storing graphic objects; classifying OpenGL graphics rendering features including global

environment features and individual object features. The system is designed based on View-Document-Control architecture and implemented by C++ language.

4.1.2 Class Diagram

We have the items to be considered for building the system, CGLApp, CGLWindow, CGLGlutFramework, CGLView, CGLLight, CGLCtrl, CGLDoc, CGLObject, CGLCube, CGLCone, CGLSphere, CGLCylinder, CGLTorus, CGLTeapot, CGLDodecahedron, CGLIcosahedron, CGLOctahedron, CGLTetrahedron, CGLAssembly. The class diagram is described as:

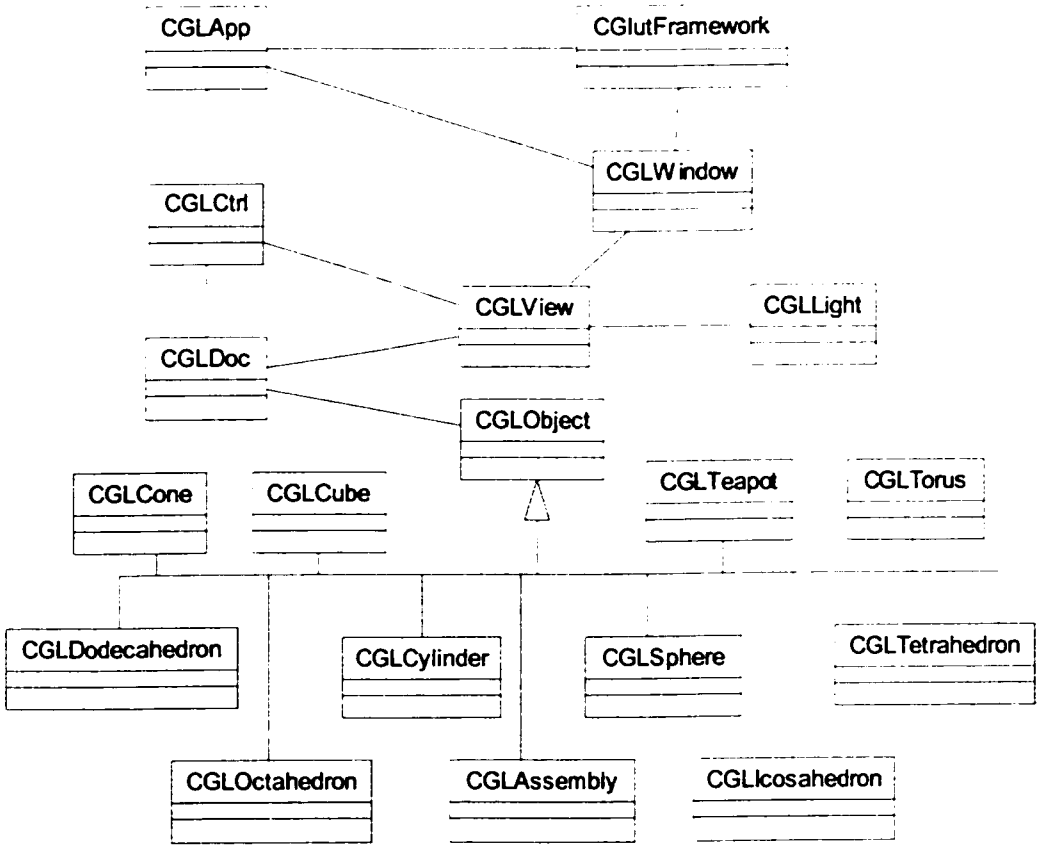


Figure 4.1 System Class Diagram

From the class diagram, we can see that there are three parts, GLUT Application Framework, View-Document-Control and Object Data.

- GLUT Application Framework: The classes, CGLApp, CGlutFramework and CGLWindow, become the system application framework. User can create new windows and new subwindows dynamically.
- View-Document-Control: The classes, CGLView, CGLDoc and CGLCtrl, become the internal core components. They handle the events coming from user GUI input display and process graphics object data.
- Object Data: The classes, CGLObject and its children classes, build the graphics object data components.

4.1.3 System Process Structure

To simplify the system structure, we consider component-based architecture. There are four parts of components to compose the system. Such as, window framework application, core components, data management and database.

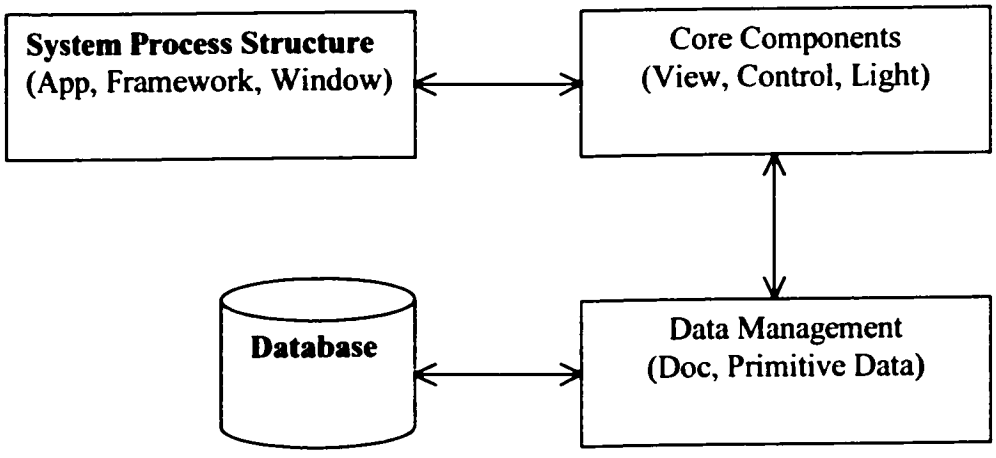


Figure 4.2 System Process Structure

The window framework is for starting the system, interacting with both user and core component parts; the core component part is for handling window messages and commands, and interacting with data management to request and get object data; the data management part is for managing and producing object data, and interacting with both core components and database; the database part is for storing object data and responding requests of data management.

4.1.4 System Interactions

In this subsection, we present interaction diagrams to describe the system process interactions for several process tasks, such as, open and close window and subwindow, import and export assembly data, model object data and perform global environment and object features.

4.1.4.1 Open and Close Windows

For opening and closing windows, the process involves CGLCtrl, CGLApp, CGLWindow, CGlutFramework objects.

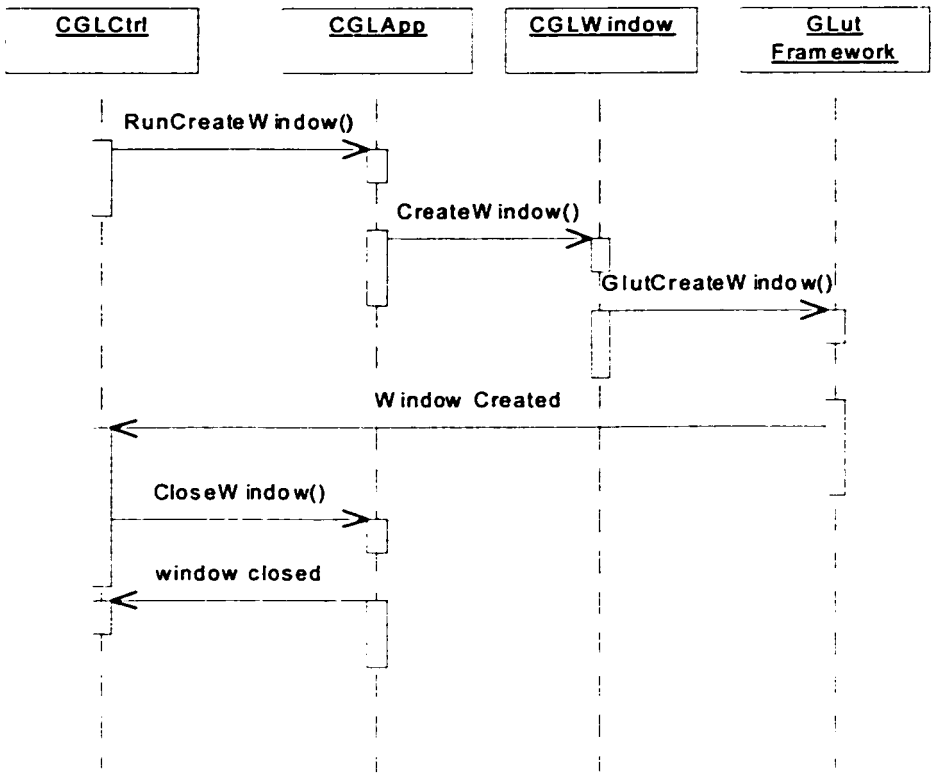


Figure 4.3 Interaction Diagram for Creating and Closing Windows

4.1.4.2 Import and Export Assembly Data

For documenting graphics objects, the system needs ways of loading and saving object data. The process involves CGLCtrl, CGLDoc, CGLObject and CGLAssembly class objects.

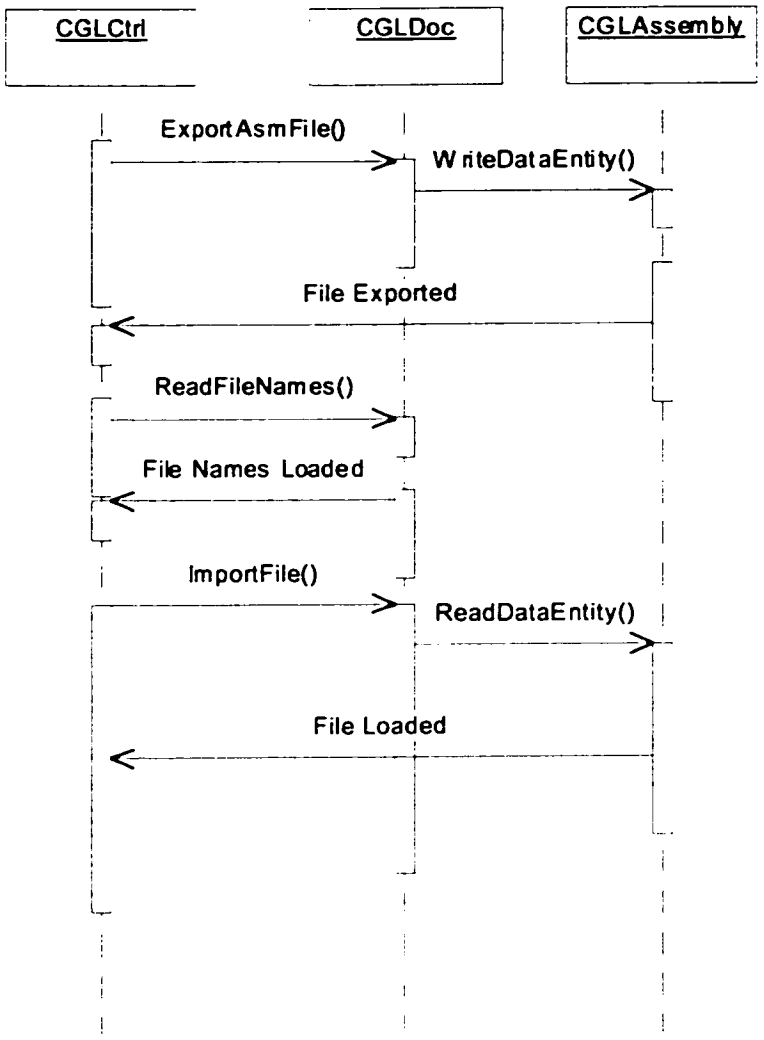


Figure 4.4 Interaction Diagram for Importing and Exporting Data Files

4.1.4.3 Model Graphics Objects and Manipulate Features

For modeling graphics objects, the system need the functionalities to process rendering primitive 3D graphics objects and manipulate the global environment and individual object features. An interaction diagram is given for loading one object and changing its features. The process involves CGLCtrl, CGLView, CGLDoc, CGLObject, CGLCube.

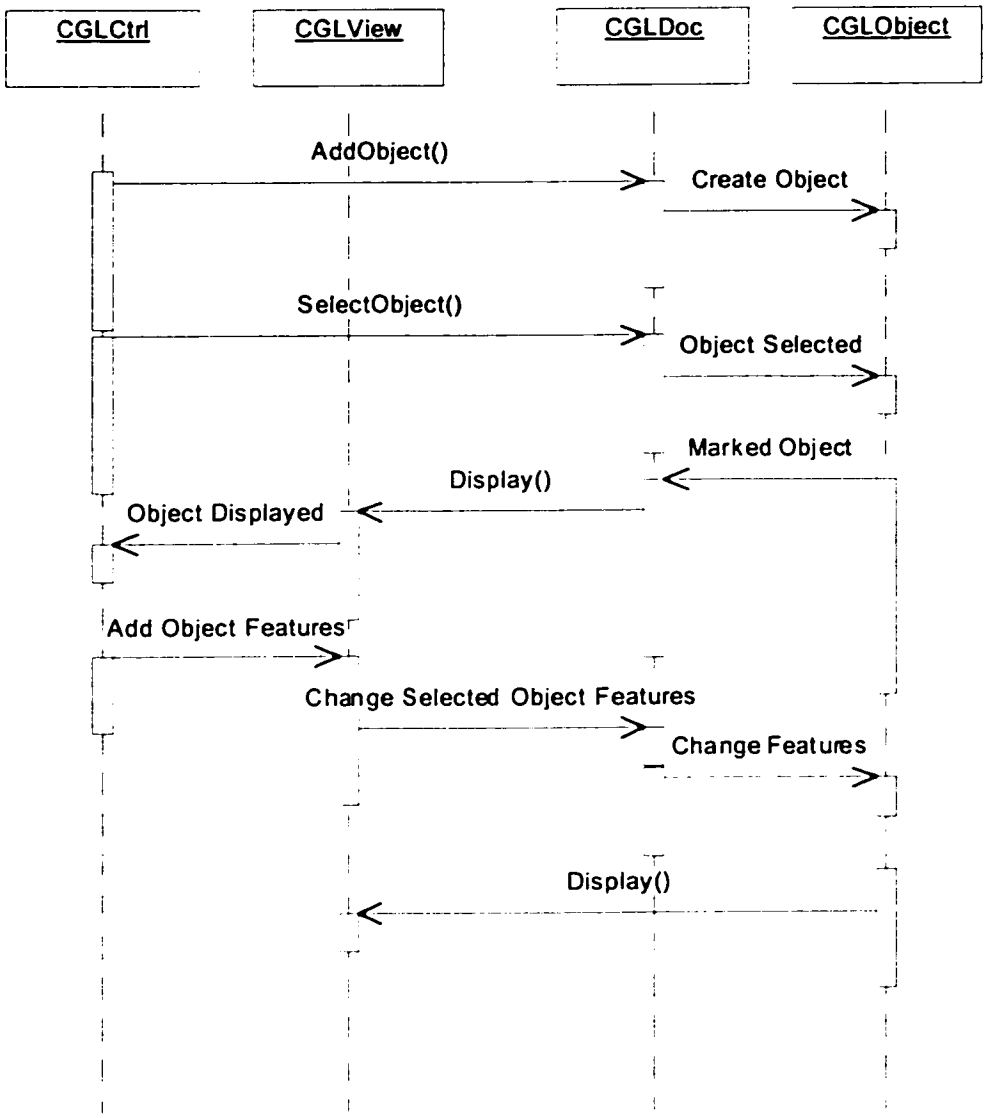


Figure 4.5 Model Object and Change Its Features

4.2 Object Data Structure and Data Storages

For documenting and editing graphics object data, we need to consider building an object data structure and data files to store data that corresponds to the data structure.

4.2.1 Object Data Structure

For creating a complicated graphics object, data structure and data storage need to be provided. In this design section, we have the hierarchical data structure to build a complicated graphics object and store the data to a data file for reuse. The structure of the data is:

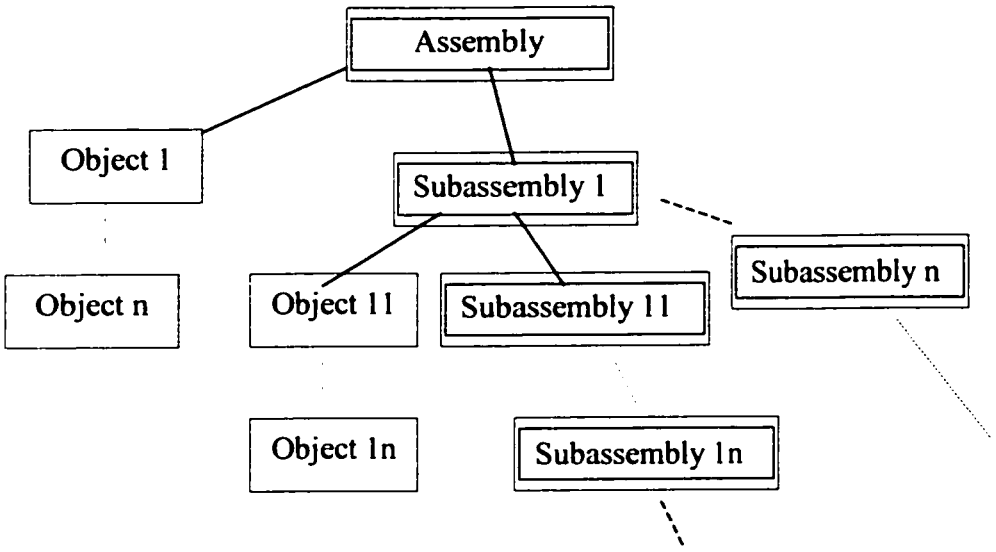


Figure 4.6 Object Data Structure

From the object data structure, we know that the assembly object can hold several primitive objects and assembly objects. It builds a hierarchical data structure.

4.2.2 Data Storages

According to GLUT framework application, we have two kinds of data file formats to consider:

- Data file for listing file names, **filenames.con,** the data format is:

Line number in data file	data
1	N, Number of assembly files
2	Index of first assembly file
3	Index of second assembly file
.	.
.	.
.	.
N+1	Index of Nth assembly file

We use the index to the file with assembly plus the index; the third line stores the index of second assembly file, etc.

- Assembly data file, **filename.dat,** the data format is:

Line number in data file	data
1	N, number of entities
2	Entity type
3	Object color, three float points
4	Object – translation, three float points
5	Object –rotation, three float points
6	Object-scale, three float points

The assembly data file stores the new object information for user to edit complicate graphics objects.

4.3 Individual Class Design

For the detail design of the system, we provide the individual class information, such as attributes and operations, and we can know more detail functionalities of the system.

4.3.1 Application Class

In the system, the application class provides the functions and we name the class as CGLApp.

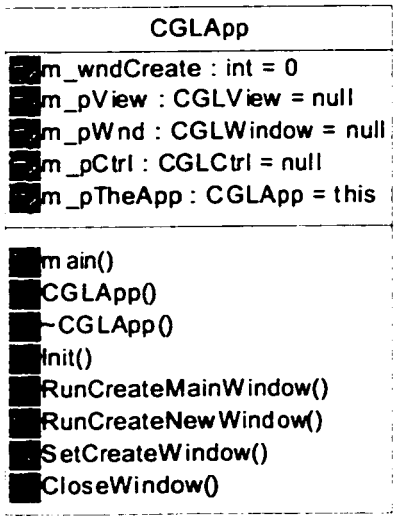


Figure 4.7 Class CGLApp

- Init() for initializing GLUT Framework, View and Control.
- RunCreateMainWindow() for creating main window object and running the main window frame application.
- RunCreateNewWindow() for creating and running new windows and new subwindows.
- CloseWindow() for terminating the individual windows.

The CGLApp class starts the application; initialize main window size and location, and classes CGlutFramework, CGLView, CGLWindow, CGLLight objects and setup to create main OpenGL windows and subwindows.

4.3.2 GLUT Framework Class

For the GLUT framework application, we consider it to be implemented by object-oriented technology. A class should be designed to enable all GLUT callback functions. We name this class as CGlutFramework that contains all the operations display, mouse, keyboard, special key, idle function, reshape callback functions.

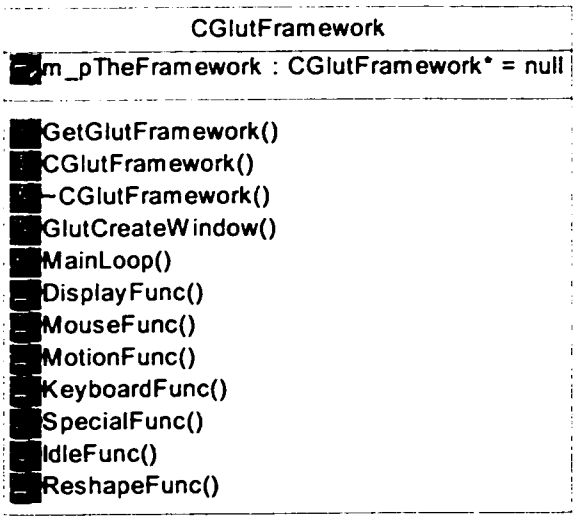


Figure 4.8 Class CGlutFramework

- DisplayFunc() the callback function for calling all the entrances of graphics representation and environment control.
- IdleFunc() for animation of the operations and actions.
- KeyboardFunc() for keyboard even control.

- MotionFunc() for action of mouse cursor moving.
- MouseFunc() for mouse even control.
- ReshapeFunc() for setting window view port, modelview and projection environment.
- SpecialFunc() for special key control.

4.3.3 Window Class

In the system, Window class provides the function to pass all the callbacks, so that we can simply create multiple windows, views, documents and controls. We name the class as CGLWindow that contains the members as CGlutFramework class.

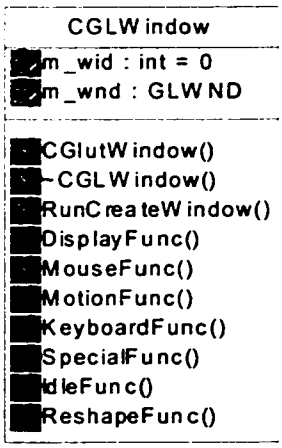


Figure 4.9 Class CGLWindow

It also has RunCreateWindow() function to call create window function in CGlutFramework. Therefore by this structure, we can create multiple windows based on object-oriented technology.

4.3.4 View Class

In the system, the view class provides the environment for the windows and is named as CGLView:

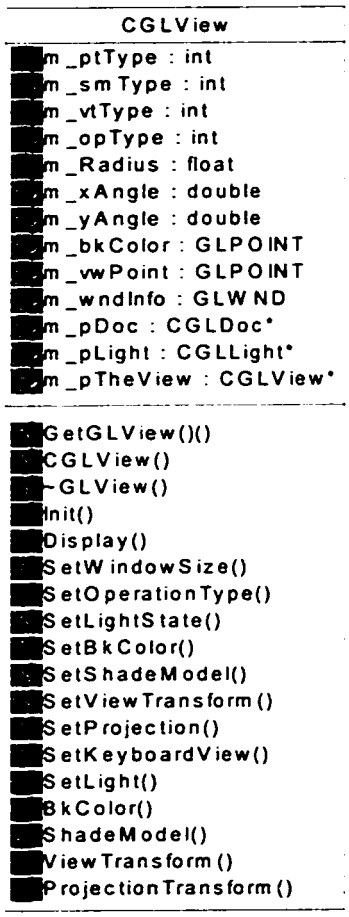


Figure 4.10 Class CGLView

It contains the functions:

- Display() for the entrance to display 3D graphics objects. The CGLWindow will call the function to enable all the functions in OpenGL GLUT window environment.
- Init() and SetWindowSize() for passing window information.
- SetBkColor() and BkColor() for setting and changing window background color.

- SetShadeModel() and ShadeModel() for setting and changing shade model, such as, flat and smooth models.
- SetProjection() and ProjectionTransform() for setting and changing projection state, such as, orthography and perspective.
- SetViewTransform() and ViewTransform() for setting and enabling model and view.
- SetLightState() and SetLight() for setting and enabling light. Let the user to change light position.
- SetKeyboardView() for setting specific view points to view the objects from keyboard input.
- SetOperationType() for selecting mouse control operations, such as, moving light, look around and zoom.

The CGLView class creates and initializes CGLDoc and CGLLight class objects, and manipulates the global environment features.

4.3.5 Control Class

In this system, the control class provides the action control for window menu, mouse button action and mouse motion, keyboard and special key controls. It is named as CGLCtrl:

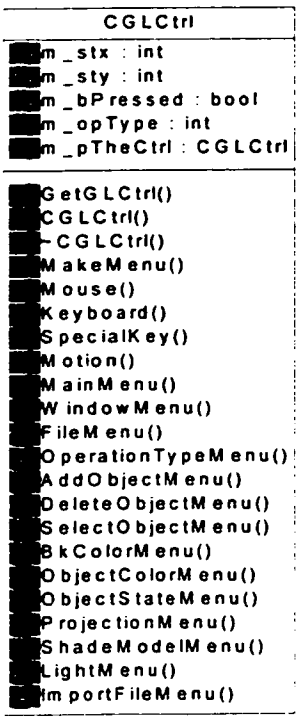


Figure 4.11 Class CGLCtrl

- MakeMenu() for the entrance of setting and calling window menu.
- Mouse() for mouse button control, mainly control moving light, view position and creating windows and subwindows.
- Keyboard() for keyboard control, mainly control specific view positions.
- SpecialKey() for special key control, mainly control object transformations.
- Motion() for mouse motion action to control global environment, such as, view position, light motion and zoom.

4.3.6 Light Class

In this system, the light class provides the functions to initialize and set light properties, enable and disable light.

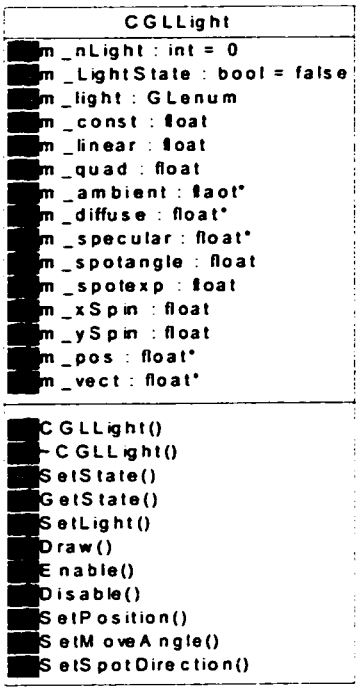


Figure 4.12 Class CGLLight

4.3.7 Document Class

In this system, the document class provides the functions to manage objects. It is named as CGLDoc and also does the operations to read and write data files.

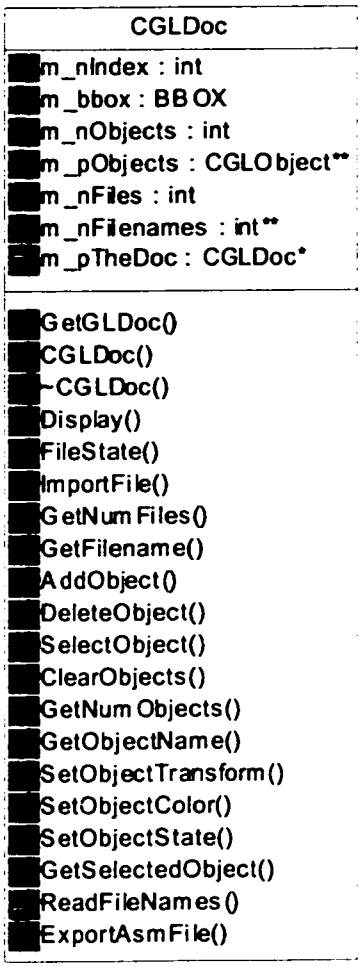


Figure 4.13 Class CGLDoc

- Display() for passing the display from CGLView class.

File import and export operations:

- FileState() for passing Window Menu events to enable import or export data files.
- GetNumFiles() for passing number of data files loaded to CGLCtrl class.
- GetFilename() for passing individual file name to CGLCtrl class.

Object management operations:

- AddObject() for adding object to document object list.

- DeleteObject() for deleting object from document object list.
- SelectedObject() for selecting object from document object list and manipulating the individual object features.
- GetNumObjects() for passing number of objects to CGLCtrl class.
- GetObjectName() for passing individual object name to window menu in CGLCtrl class.
- SetObjectTransform() for setting object transformations from CGLCtrl class keyboard control.
- SetObjectColor() for setting object color from CGLCtrl class window menu.
- SetObjectState() for setting object rendering state, solid or wire mesh, provided by CGLCtrl class window menu.
- GetSelectedObject() for passing selected object.

4.3.8 Geometric Object Class

The object class is a base class of geometric graphic object and contains material feature information, object color, shade model, show status and material properties. It is named as CGLObject.

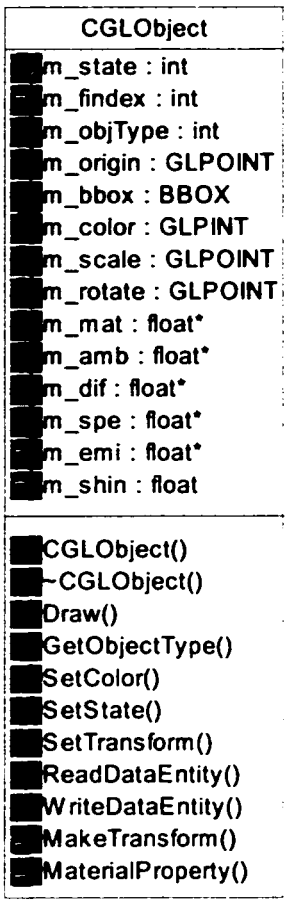


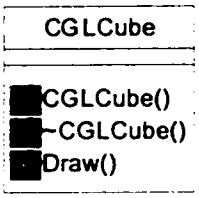
Figure 4.14 Class CGLObject

- Draw() function for rendering the individual object;
- GetObjectType() function for passing the object type to window menu.
- SetColor() function for changing color by window menu call.
- SetState() function for switching rendering states between solid and wire.
- SetTransform() function for passing transformation data from window menu call.
- ReadDataEntity(0 function for reading entity data from data file.
- WriteDataEntity() function for writing entity data to data file.

The class CGLObject is a base class and has the general attributes and functions, for children classes, they have more refined attributes and operations.

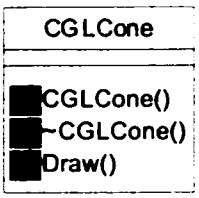
- **Cube**

The cube object is designed as class CGLObject's child class. It inherits the attributes and operations from CGLObject, CGLCube class has its own features, such as, length, width and height.



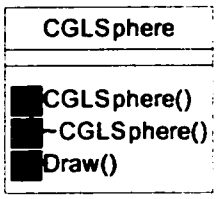
- **Cone**

The cone object is designed as class CGLObject's child class. It inherits the attributes and operations from CGLObject, CGLCone class has its own features, such as, radius and height.



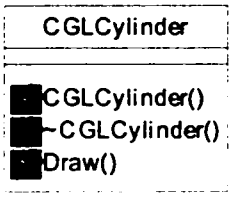
- **Sphere**

The shpere object is designed as class CGLObject's child class. It inherits the attributes and operations from CGLObject, CGLSphere class has its own features, such as, radius.



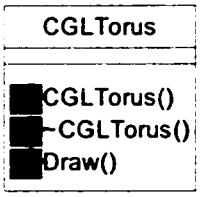
- **Cylinder**

The cylinder object is designed as class CGLObject's child class. It inherits the attributes and operations from CGLObject, CGLCylinder class has its own features, such as, radius and height.



- **Torus**

The torus object is designed as class CGLObject's child class. It inherits the attributes and operations from CGLObject, CGLTorus class has its own features, such as, inner radius and outer radius.



- **Teapot, Dodecahedron, Octahedron, Tetrahedron and Icosahedron**

The Teapot, Dodecahedron, Octahedron, Tetrahedron and Icosahedron objects are children classes of CGLObject, they all inherit some attributes and operations from CGLOBject, but also have size attribute need to be defined.

- **Assembly**

The assembly object is a child class of CGLObject, and it inherits some attributes and operations from CGLObject, but class CGLAssembly has a special rule to build its object. It groups the primitive objects and other assembly objects to build a new assembly object.

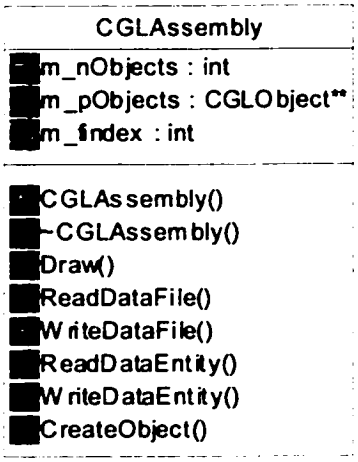


Figure 4.15 Class CGLAssembly

From CGLAssembly class, besides it has same structure with class CGLObject, it has additional operations and attributes.

- ReadDataFile() for reading the containment assembly data file.
- WriteDataFile() for writing a new assembly data file to database.
- ReadDataEntity() for reading assembly entity that is different from the other kind of objects.
- WriteDataEntity(0 for writing assembly entity that is different from the other kind of objects.

- CreateObject() for checking the object to be created and creating the object.

4.4 Structure Abstract Data Type

In this system, some structure abstract data types are provided for simplifying class attribute definition and easily implementing various class member functions and function calls.

4.4.1 3D Point Struct

For 3D point, we present GLPOINT structure for defining 3D point, vector, color, and transformation variables.

```
typedef struct _tagGLPOINT
{
    float x;
    float y;
    float z;
} GLPOINT;
```

4.4.2 Bounding Box Struct

For geometric object measurement, we need a structure to define bounding box.

```
typedef struct _tagBBOX
{
    GLPOINT min;
    GLPOINT max;
```

```
} BBOX;
```

4.4.3 Window Information Struct

In this system, we need pass the window information, such as, window size and position, to various class objects. For convenience, we define a structure to store the window information.

```
typedef struct _tagGLWND
{
    int w;
    int h;
    int x;
    int y;
} GLWND;
```

4.5 Structure Abstract Data Type

For managing system message and window message even passing, we need enumerate various message items. In this system, we define integer constants to define the messages.

Chapter 5: 3D Graphics Editor – Application Result

In this section, a sample application is provided with the 3D graphics editor. The editor builds the graphic objects by assembling the primitive objects, such as, cube, cone, sphere, torus, cylinder, teapot, and etc. The processes for building the 3D graphic objects demonstrate all the functionalities of the system, such as, multi-objects assembly editing, multi-window creation and manipulation and dynamic assembly objects storage and loading. The system is a GLUT framework application, has the primitive objects let the user to document and edit new complex objects, and provides the functions to manipulate the objects globally and individually.

The 3D graphic object is named to " family kitchen Table and Chairs". It is designed by several stages, system start, creating a table, chair, cup, light, their assembly, multiple windows, etc.

5.1 Start System

Before creating a 3D graphic object, we start the system by executing the exe file, openglapp.exe. An OpenGL Framework window is created.

Figure 5.1 Open OpenGL Window

From the figure, the window is an empty window. When we press the right mouse button, we see a window menu, which contains the items, window, file, model, view, object, exit.

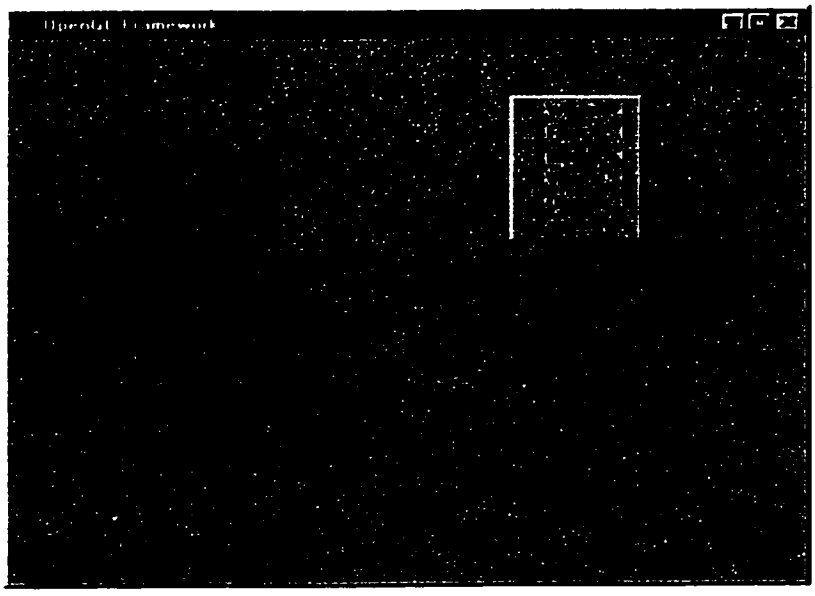


Figure 5.2 OpenGL Window Menu

In order to see the 3D graphic environment, we need to enable the light.

5.2 Creating A Bed

From section 5.1, we already start the system and open the window menu. Now we create a bed in the window and do the following processes.

- Select model item and add object item under the model item, and create a cube.
    a. After creating a cube, select model item and select object item under the model item, and choose the cube item.
    b. Press the special keys to reshape the cube by the transformations.
        - Press F1 to narrow the cube in the direction x-axis (1, 0, 0);
        - Press F2 to enlarge the cube in the direction x-axis (1, 0, 0);
        - Press F3 to narrow the cube in the direction y-axis (0, 1, 0);
        - Press F4 to enlarge the cube in the direction y-axis (0, 1, 0);
        - Press F5 to narrow the cube in the direction z-axis (0, 0, 1);
        - Press F6 to enlarge the cube in the direction z-axis (0, 0, 1);
        - Press F7 to rotate the cube in clockwise around x-axis;
        - Press F8 to rotate the cube in counter-clockwise around x-axis;
        - Press F9 to rotate the cube in clockwise around y-axis;
        - Press F10 to rotate the cube in counter-clockwise around y-axis;
        - Press F11 to rotate the cube in clock-wise around z-axis;
        - Press F12 to rotate the cube in counter-clockwise around z-axis;
        - Press PageUP to scale the cube in small;
        - Press PageDown to enlarge the cube in large;
        - Press Insert key to recover the cube to the original shape.

c. Press the keyboard keys, 'u', 'v', 'l', 'r', 'f', 'b' to view the object from specific directions:

- Press key 'u', view the object from direction (0, 1, 0);
- Press key 'v', view the object from direction (0, -1, 0);
- Press key 'l', view the object from direction (-1, 0, 0);
- Press key 'r', view the object from direction (1, 0, 0);
- Press key 'f', view the object from direction (0, 0, 1);
- Press key 'b', view the object from direction (0, 0, -1);

- Select model item and add object item under the model item, and create four cubes for the legs of the bed, another two cubes for the bed board and front fence, and one torus for the bed fence, and a cylinder and a sphere. Then select each object from the menu object list, and reshape the each object by controlling special keys.

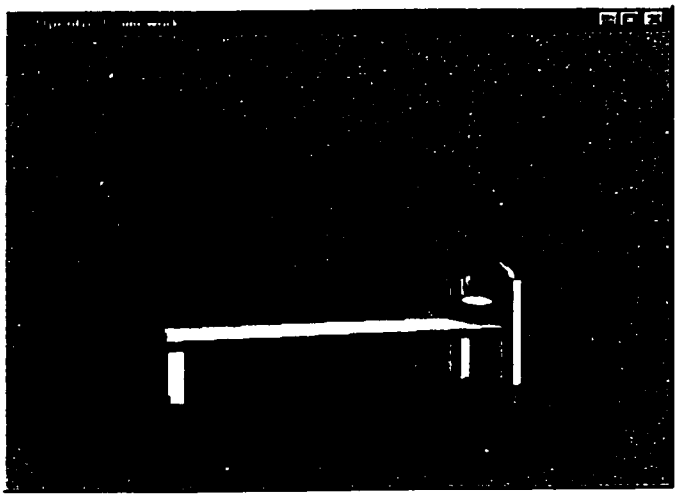


Figure 5.3 Create a Bed

After the table is created, select file item and export file item under file item to save the table to a data file as the assembly data.

5.3 Creating A Chair

As the process for creating a table, we can load a sequence of cubes to create a chair.

Figure 5.4 Create a Chair

The chair is created and, the graphic object is exported and stored in an assembly data file.

5.4 Creating Living Room Light

For creating a living room light graphical object, we can use the primitive objects, such as, cylinder, cone, sphere, and export the assembly graphic object and store in an assembly data file.

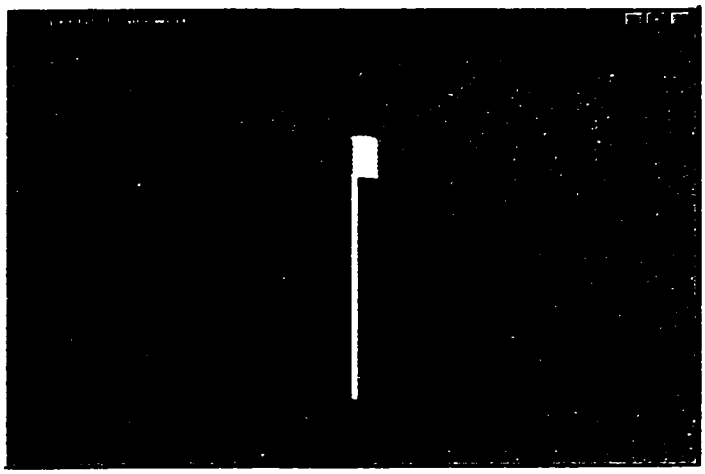


Figure 5.5 Create Living Room Light

For creating the light graphical object, we use the system functions, all the transformations, moving system light, manipulating global view positions, look around and zoom. All the global environmental controls are with left mouse button and its motion.

5.5 Create a Table Light

For creating a table light, we can use the primitive objects, such as, sphere, cylinder, cone, and export the assembly graphic object and store in an assembly data file.

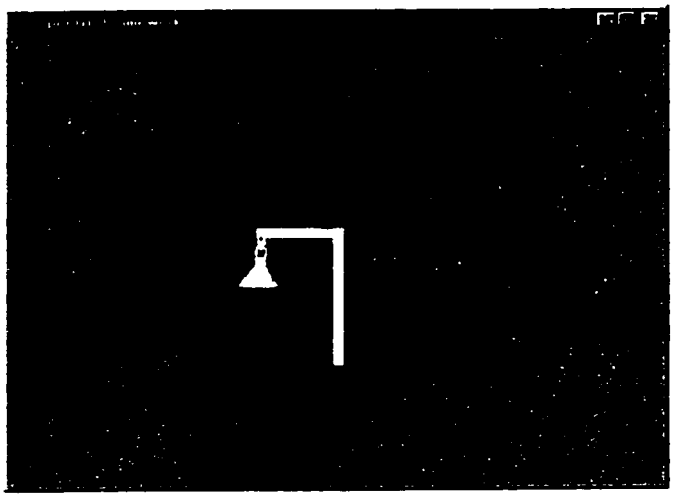


Figure 5.6 Create Table Light

For creating the light, we also use the system functions, all the transformations, moving system light, manipulating global view positions, look around and zoom. All the global environmental controls are with left mouse button and its motion.

5.6 Create Bookshelf

For creating a bookshelf, we just need load cubes and reshape them to a shelf and books.

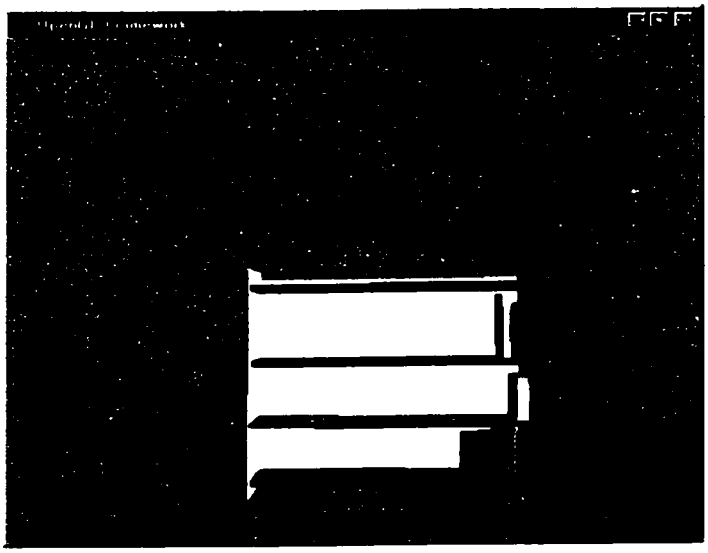


Figure 5.7 Create Bookshelf

5.7 Create A Wardrobe

For creating a wardrobe, we need load the objects, cubes, torus, and reshape them to the frame, withdraws and the bars.

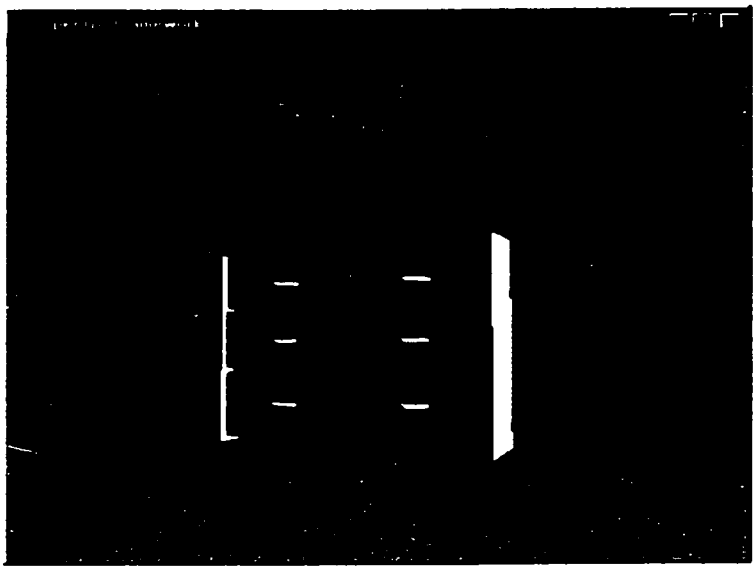


Figure 5.8 Create Wardrobe

5.8 Create A Desk

For creating a desk, we also need load the objects, cubes, torus, and reshape them to the frame, withdraws and the bars.

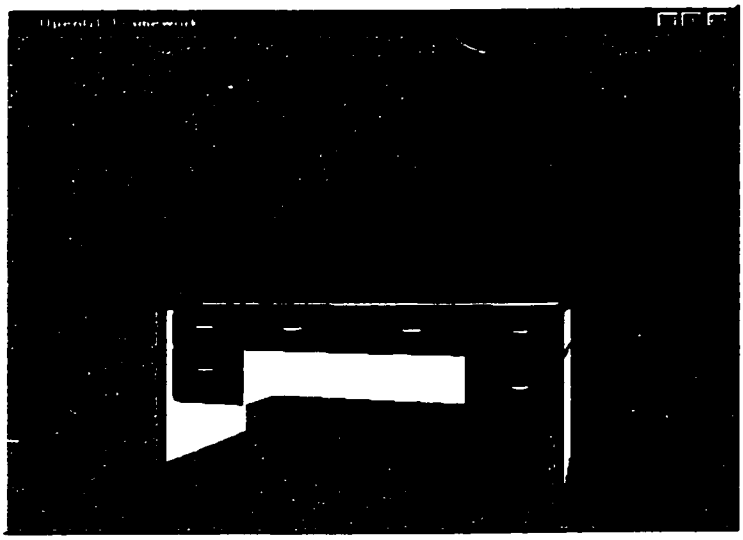


Figure 5.9 Create A Desk

5.9 Create A Telephone

For creating a telephone, we load a cube, 13 spheres, 2 torus, and 2 cones, and reshape the objects to form the telephone assembly object.

Figure 5.10 Create A Telephone

5.10 Create A Clock

For creating a clock, we need load torus, cubes and sphere, and reshape them to the right sizes and locations.

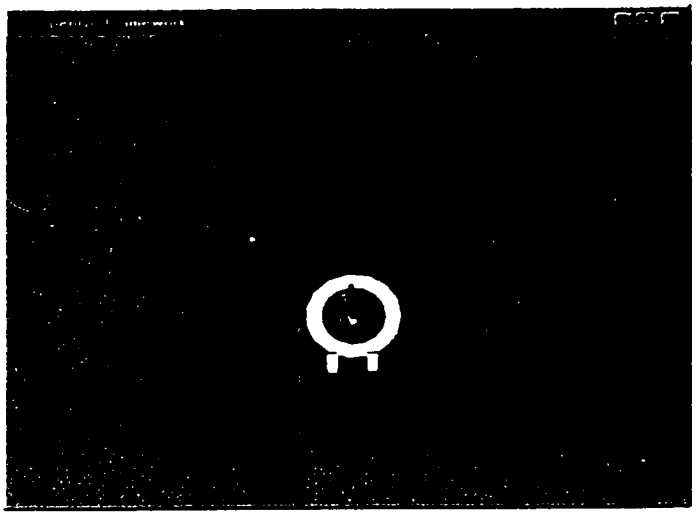
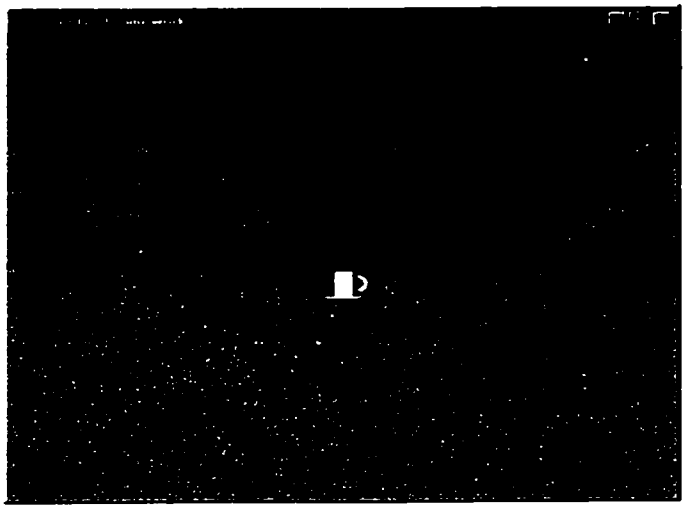


Figure 5.11 Create A Clock

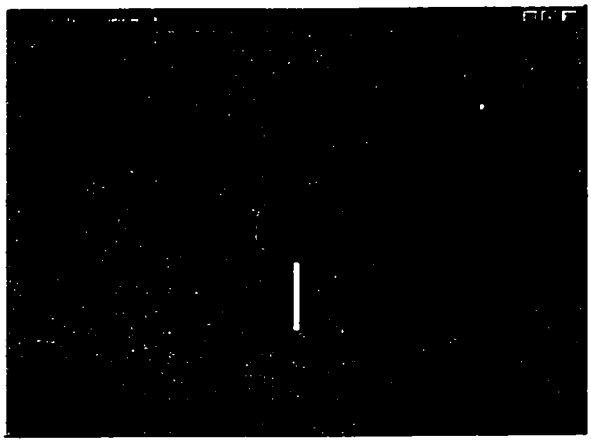
5.11 Create Some Other Assembly Objects

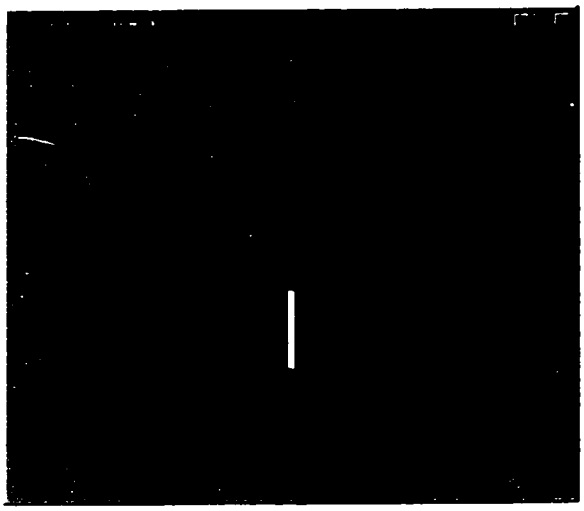
In this subsection, we create some smaller assembly objects, toys, cup and the bottle flower.

- For creating a cup, we need load a cylinder, sphere and torus.



- For creating this project in window, we load a dodecahedron, an icosahedron, octosahedron and tetrahedron, cone, cube and cylinder.





5.12 Assembly the Created Objects

For building the final assembly graphic object, we create a bachelor bedroom and load all the created assembly objects.

- Select window menu file item and file names item under the file item;
- Select window menu file item and import files item under the file item;
- Load the bed assembly file;
- Load the chair assembly file in four times;
- Load the bookshelf assembly file;
- Load the living room light and table light assembly files;
- Load the wardrobe assembly file;
- Load the clock assembly file;
- Load the telephone assembly file;
- Zoom the view.

Figure 5.12 Bedroom Assembly Object

For loading the assembly files, we reshape the objects by the transformations with special keys. We cannot change the assembly color as we take the assembly data as the final design result. We also can repeat to load the assembly objects, for example, here we load chair in four times. The background color can be changed with the user needs.

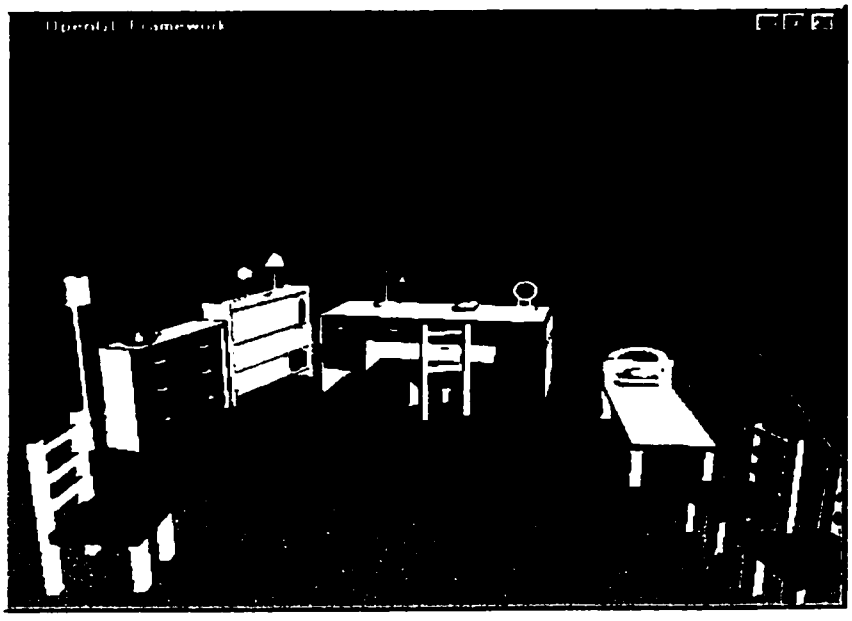


Figure 5.13 Change Background Color to Black

5.13 Create Multiple Windows

For creating and terminating multi-windows, the functions, glutCreateWindow(), glutCreateSubWindow(), glutDestroyWindow(), glutGetWindow() are called to process the tasks. In this system, we use window menu to control the window creating and closing actions.

- Create a window dynamically, select menu Window item and Create Subwindow item at any time.

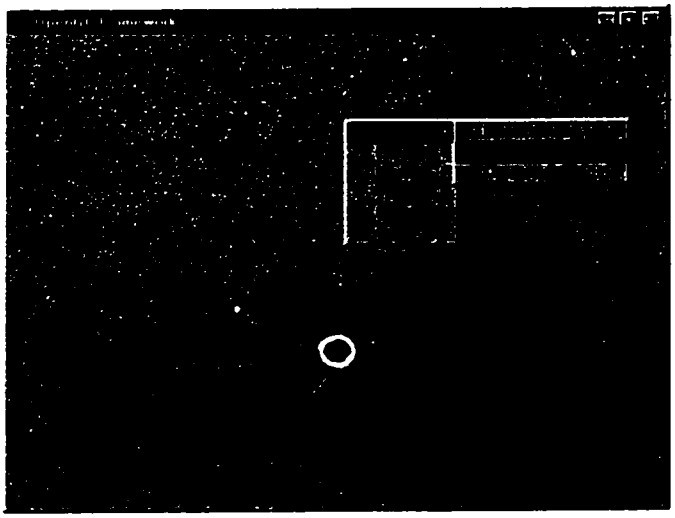


Figure 5.14 Select Menu Create Subwindow

- Move mouse cursor to define location for creating a subwindow. Press mouse left button, then window is created.

Figure 5.15 Create Sub window

- Repeat the sub window creation process, you can create the subwindows you want.

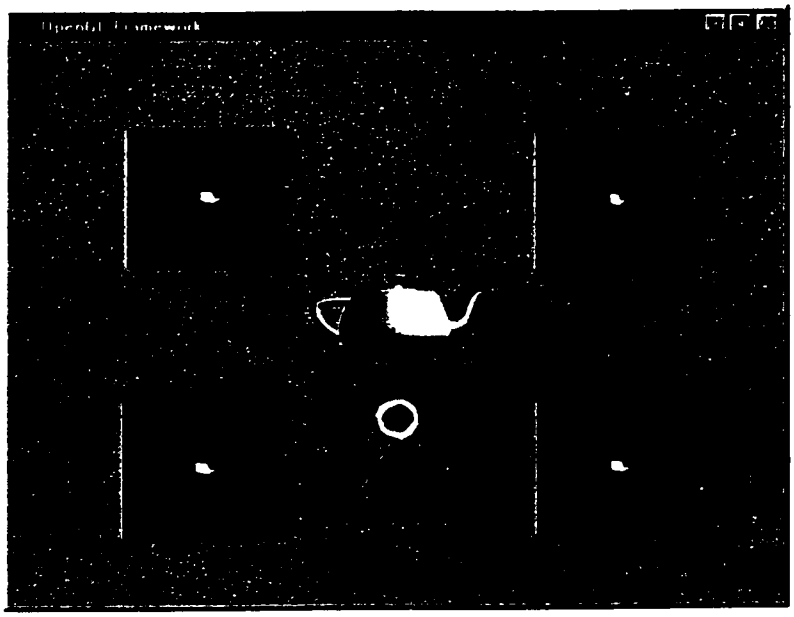


Figure 5.16 Multi – Subwindows

- For closing subwindow, press right mouse button on the subwindow and select Close Window item, then the window is closed.

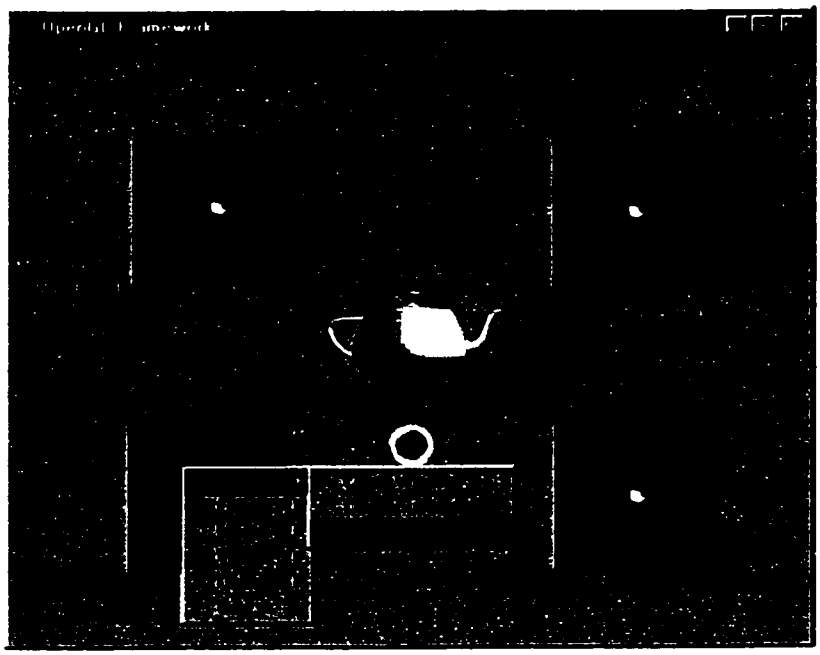


Figure 5.17 Select Close Window Item

Figure 5.18 The Window Is Closed

The subwindows can be created and closed at any time and any location of the parent window.

- For creating new windows, the principle is same as to create subwindow, but the new window is not contained in the other window and is independent from other window.

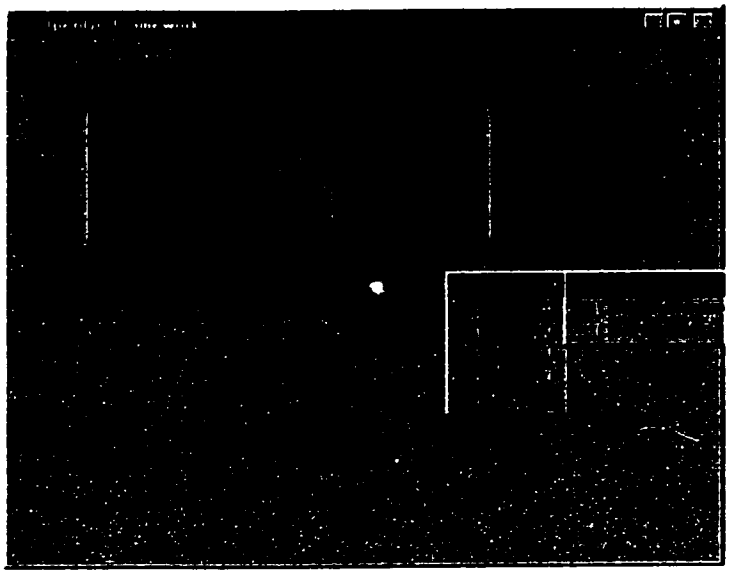


Figure 5.19 Select Menu Create Window Item

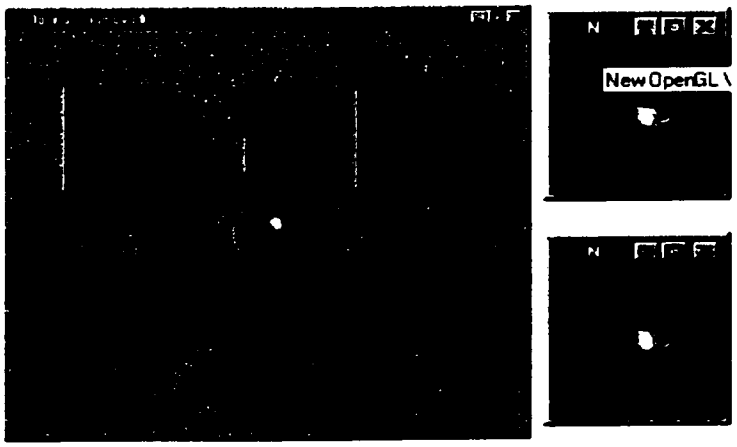


Figure 5.20 Multi-Windows Layout

5.14 Global and Object features

In this demo example, some global environment and individual object features are presented. For the global features, the system enables background color, move light, look around, zoom, switch shade model between flat and smooth, and switch projections between orthographic and perspective projections; for individual object features, change object color and switch object show states between solid and wire.

- Load objects to window.

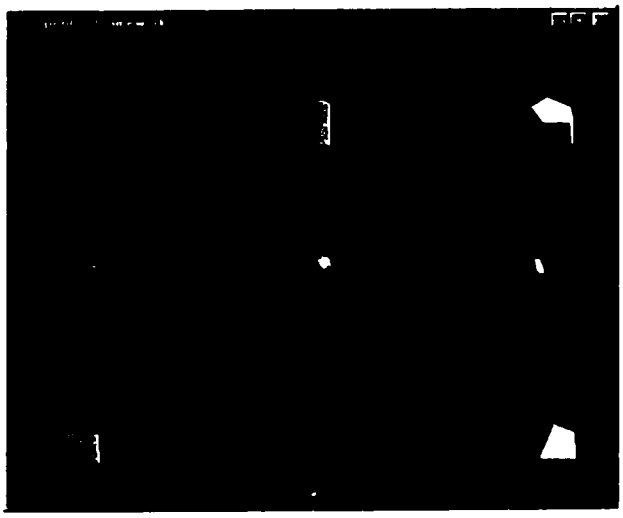


Figure 5.21 Objects In Multi-Color

- Change light position, projection to orthographic and some objects to wire state.

Figure 5.22 Global and Objects Features

Chapter 6: Conclusion

In designing a Graphic Editor for 3D objects, we used the GLUT framework and object-oriented technology to develop a system with many capabilities, including management of multiple windows and files of graphical object data.

6.1 Experiences on Object-Oriented Programming

From the system analysis, design and implementation, we use object-oriented technology to develop the project. We understand well about internal knowledge of inheritances, dependency, associations, encapsulations and polymorphisms. It provides us an easy way to model the project well in object-oriented concept.

C++, as an object-oriented language, is concerned with the creation, management, and manipulation of objects. An object encapsulates data and methods used to manipulate the data.

For OpenGL framework, we learn how to build a bridge to pass callback functions to create multiple windows and subwindows, and other OpenGL API functionalities. We find the data structure to store assembly data, and build a hierarchical tree to call assembly and primitive objects recursively.

6.2 Further Work

When documenting the graphic objects by using 3D graphics editor, we find the graphic user interface is convenient to user to operate its items.

For the future work, we need,

- Get more functionalities to build the GUI part;
- Have multiple views and multiple controls;
- Add surface rendering from GLU library;
- Add more light functionalities.

Bibliography

[PG00] Peter Grogono, *Requirement of Glut Framework Application,*

Faculty Website, Concordia University, 2000.

[PG98] Peter Grogono. *Getting Started with OpenGL*, Concordia University, 1998.

[RW96] Richard S. Wright JR. *OpenGL Super Bible*, 1996.

[BRJ99] Grady Booch, James Rumbaugh, Ivar Jacobson, *The Unified Modeling Language User Guide*, Press. Addison-Wesley, 1999.

[MJT99] Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide*, Third Edition, Addison-Wesley, 1999.

[HS98] Herbert Schildt. *C++: The Complete Reference*, Third Edition, 1998.

[SLY] Shuli Yang, *Implementation of 3D Graphics Editor*, Major Report, Department of CS, Concordia, May, 2002.