**CHAPTER 4**

**ARCHITECTURE**

**4.1 Block Diagram**

Interactive object module

User interface

Mouse

Welcome Page

Select Object (Wired)

Select Object (Solid)

Demonstration

Key Control

GLUT Object Function Syntax

Transformation Functions Used

Example

References

Thank You

Exit

Keyboard

Navigation

Zoom(in/out)

Rotate

Reset

Full Screen

Exit

Cube

Cone

Torus

Dodecahedron

Octahedron

Tetrahedron

Icosahedron

Teapot

Rectangular Mesh

Sphere

Cube

Cone

Torus

Dodecahedron

Octahedron

Tetrahedron

Icosahedron

Teapot

3D Sierpenski Gasket

Rotation Demonstration

Translation Demonstration

Scaling Demonstration

**Figure 4.1** Block diagram of module

**4.2 Block diagram explanation**

This project is designed and implemented using OpenGL interactive application that basically deals with providing the graphical interfaces between user and system. The mini project “Interactive object module” is the implementation of the geometrical built-in functions.They are subjected to different transformations, to demonstrate Scaling, Translating and Rotation operations in all directions. We use mouse interface to perform select operation among the menu (list of choices). We will introduce keyboard interaction to rotate, zoom in, zoom out, reset, full screen, exit and move the objects in all direction from program output in various snapshots.

This project “**Interactive object module”** is developed using OpenGL. It shows Creation, Transformation and Shadowing of Objects. When you run this program,

* Left mouse button shows the Menu.
* Keyboard buttons control the Movement, size and rotation of objects.

The main features provided are:

* Creation of Objects
  + - * Wired form
      * Solid form
* Transformation of Objects
  + - * + Translation
        + Rotation
        + Scaling
* Shadows of Objects.

The program has modes for resetting the scaled objects and supporting full screen.

**4.3 Chapter summary**

A block diagram is a diagram of a system in which the principal parts or functions are represented by blocks connected by lines that show the relationships of the blocks. The schematic diagram of a Interactive object module does not show the width of each connection.

**CHAPTER 5**

**DESIGN**

Rotation Demonstration

Translation Demonstration

Scaling Demonstration

Mouse

Press Key

User interface

Keyboard

SOLID

Sphere

Cube

Cone

Torus

Dodecahedron

Octahedron

Tetrahedron

Icosahedron

Teapot

3D Sierpenski Gasket

WIRE

Cube

Cone

Torus

Dodecahedron

Octahedron

Tetrahedron

Icosahedron

Teapot

Rectangular Mesh

**5.1 Data flow diagram**

Starting of the program

Main()

Displaying the main screen

FF

stop

Figure 5.2 Dataflow diagram

**5.2 Class diagram**

Mouse

Name State

Keyboard

Name Key

Main Screen

Name String

Transformation

Rotation Demonstration

Translation Demonstration

Scaling Demonstration

Wire

Cube

Cone

Torus

Dodecahedron

Octahedron

Tetrahedron

Icosahedron

Teapot

Rectangular Mesh

Solid

Sphere

Cube

Cone

Torus

Dodecahedron

Octahedron

Tetrahedron

Icosahedron

Teapot

3D Sierpenski Gasket

Figure 5.2 Class diagram

**5.3 Explanations of design**

The design specifies the design of various aspects and different stages of the project. Design is the ultimate frame work of the project. In essence the design is a plan or a mind full blueprint of the project to be developed.The design of the system shows that when the program is made to run it first enters the output window. User interacts with the system using mouse.

* The mouse is used to obtain the pop down menu when the right mouse button is clicked.
* The various options are keyboard functions, mouse functions and quit.

A **data-flow diagram** (DFD) is a way of representing a flow of a data of a [process](https://en.wikipedia.org/wiki/Process) or a system usually an [information system](https://en.wikipedia.org/wiki/Information_system). The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops. Specific operations based on the data can be represented by a [flowchart](https://en.wikipedia.org/wiki/Flowchart).

A **class diagram** in the [Unified Modelling Language](https://en.wikipedia.org/wiki/Unified_Modeling_Language)(UML) is a type of static structure diagram that describes the structure of a system by showing the system's [classes](https://en.wikipedia.org/wiki/Class_(computer_science)), their attributes, operations (or methods), and the relationships among objects.

The class diagram is the main building block of [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) modelling. It is used for general [conceptual modelling](https://en.wikipedia.org/wiki/Conceptual_model) of the structure of the application, and for detailed modelling translating the models into [programming code](https://en.wikipedia.org/wiki/Programming_code). Class diagrams can also be used for [data modelling](https://en.wikipedia.org/wiki/Data_modeling). The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.

**5.4 Chapter summary**

DFD (Data Flow Diagram) is a graphical mapping of data structures and their interrelationships. In turn the class diagram shows the entities, objects and their relationships. The information system manages the flow of data introduced into the system. To graphically represent and document the flow of data through some information system is used a Data Flow Diagram

**CHAPTER 6**

**IMPLEMENTATION**

**Pseudo Code**

The execution of the program starts from the main function. It calls various in-built and user defined functions, the pseudo code is as follows:

**6.1 Functions Used**

* **glBegin()**

Marks the beginning of a vertex-data list that describes a geometric primitive.

* **glClearColor()**

Sets the current clearing color for use in clearing color buffers in RGBA mode.

* **glClear()**

Cleans the specified buffers to their current clearing values.

* **glFlush()**

Forces previously issued OpenGL commands to begin execution, thus guaranteeing that they complete.

* **glEnd()**

Marks the end of a vertex-data list.

* **glPushMatrix()**

Pushes all matrices in the current stack down one level. The current stack is determined by glMatrixMode().

* **glPopMatrix()**

Pops the top matrix off the stack, destroying the contents of the popped matrix.

* **glMatrixMode()**

Specifies whether the model view,projection, or texture matrix will be modified.

* **glutInitDisplayMode()**

Initialises GLUT and process any command line arguments.

* **glutInitWindowPosition()**

Specifies the initial x, y location for the upper left corner of the window in relation to the upper left corner of the monitor screen.

* **glutCreateWindow()**

Creates the window with an openGL context using the previously set characteristics.

* **glutMainLoop()**

Enters the GLUT processing loop, which is an infinite loop.

* **glutReshapeFunc()**

Specifies the function that is called whenever the window is resized or moved.

* **glutDisplayFunc()**

Specifies the function that is called whenever the contents of the window need to be redrawn.

* **glutKeyboardFunc()**

Specifies the function that is called when a key that generates an ASCII character is pressed. The key call back parameter is the generated ASCII value.

* **glutMouseFunc ()**

Specifies the function that is called when a mouse button is pressed or released. glutPostRedisplay (). Marks the current window as needing to be redrawn. At the next opportunity, the call back function registered in by glutDisplayFunc () will be called.

* **gluOrtho2D ()**

Creates the matrix for for projecting two-dimensional coordinates onto the screen and multiplies the current project matrix by it.

**glutMainLoop ()**

Enters the GLUT processing loop, which is an infinite loop.

* **glurtho ()**

Creates a matrix for an orthographic parallel viewing volume and multiplies the current matrix by it.

* **gltranslate ()**

Multiplies the current matrix by a matrix that moves(translates) an object

**6.2 Modules**

* glutSpecialFunc()
* glutBitmapCharacter()
* glutWireSphere()
* glutWireCube()
* glutWireCone()
* glutWireTorus()
* glutWireDodecahedron()
* glutWireOctahedron()
* glutWireIcosahedron()
* glutWireteapot()
* glutSolidSphere()
* glutSolidCone()
* glutsolidDodecahedron()
* glutSolidOctahedron()
* glutSolidTetrahedron()
* glutsolidIcosahedron()

**Solid and Wired Object Functions:**

**CONE:**

glutSolidCone, glutWireCone — render a solid or wireframe cone respectively.

C Specification

**void glutSolidCone** (GLdouble base , GLdouble height, GLint slices, GLint stacks);

**void glutWireCone** (GLdouble base, GLdouble height, GLint slices, GLint stacks);

**Parameters**

Base – The radius of the base of the cone.

Height- The height of the cone.

Slices- The number of subdivisions around the Zaxis.

Stacks- The number of subdivisions along the Zaxis.

**DODECAHEDRON:**

glutSolidDodecahedron, glutWireDodecahedron — render a solid or wireframe dodecahedron (12-sided regular solid) respectively.

C Specification

**void glutSolidDodecahedron( );**

**void glutWireDodecahedron( );**

**Description:**

glutSolidDodecahedron and glutWireDodecahedron render a solid or wireframe dodecahedron respectively centered at the modeling coordinates origin with a radius of 3.

**ICOSAHEDRON:**

glutSolidIcosahedron, glutWireIcosahedron — render a solid or wireframe icosahedron (20-sided regular solid) respectively.

C Specification

**void glutSolidIcosahedron();**

**void glutWireIcosahedron();**

**Description**

glutSolidIcosahedron and glutWireIcosahedron render a solid or wireframe icosahedron respectively. The icosahedron is centered at the modeling coordinates origin and has a radius of 1.0.

**OCTAHEDRON:**

glutSolidOctahedron, glutWireOctahedron — render a solid or wireframe octahedron (8- sided regular solid) respectively.

C Specification

**void glutSolidOctahedron();**

**void glutWireOctahedron();**

**Description**

glutSolidOctahedron and glutWireOctahedron render a solid or wireframe octahedron respectively centered at the modeling coordinates origin with a radius of 1.0.

**SPHERE:**

glutSolidSphere, glutWireSphere — render a solid or wireframe sphere respectively.

C Specification

**void glutSolidSphere** (GLdouble radius, GLint slices, GLint stacks);

**void glutWireSphere** (GLdouble radius, GLint slices, GLint stacks);

**Description**

Renders a sphere centered at the modeling coordinates origin of the specified radius. The sphere is subdivided around the Z axis into slices and along the Z axis into stacks.

**TEAPOT:**

glutSolidTeapot, glutWireTeapot — render a solid or wireframe teapot respectively.

C Specification

**void glutSolidTeapot(**GLdouble size);

**void glutWireTeapot(**GLdouble size);

**Description:** glutSolidTeapot and glutWireTeapot render a solid or wireframe teapot respectively. Both surface normals and texture coordinates for the teapot are generated. The teapot is generated with OpenGL evaluators.

**TETRAHEDRON:**

glutSolidTetrahedron, glutWireTetrahedron — render a solid or wireframe tetrahedron (4-sided regular solid) respectively.

C Specification

**void glutSolidTetrahedron();**

**void glutWireTetrahedron();**

**Description**

glutSolidTetrahedron and glutWireTetrahedron render a solid or wireframe tetrahedron respectively centered at the modeling coordinates origin with a radius of 3.

**TORUS:**

glutSolidTorus, glutWireTorus — render a solid or wireframe torus (doughnut) respectively.

C Specification

**void glutSolidTorus** (GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings) ;

**void glutWireTorus** (GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings) ;

**Description**

glutSolidTorus and glutWireTorus render a solid or wireframe torus (doughnut) respectively centered at the modeling coordinates origin whose axis is aligned with the Z axis.

**CUBE:**

glutSolidCube, glutWireCube — render a solid or wireframe cube respectively.

**void glutSolidCube**(GLdouble size);

**void glutWireCube**(GLdouble size);

**Description**

glutSolidCube and glutWireCube render a solid or wireframe cube respectively. The cube is centered at the mod

**void main ()** {

Sets the display mode

Sets the window position

Sets the window size

Calls the display () function for communication window

Calls the display () function for developer’s information window

Calls the keyboard () function

Calls the myreshape () function.

}

**TRANSFORMATION**

It is a function that takes a point and maps that point into another point.

**Types of Transformation**

## The Viewing Transformation

The Viewing Transformation specifies location of the viewer or the camera. This transformation should be performed first before any other transformations. All consequent transformations will be based with respect to the effects of this transformation.

## The Modeling Transformation

The Modeling Transformation is used to modify the position, rotation angle and size of your object (or parts of it) in a scene. Scaling is a type of modeling transformation if you apply it to your object. And if you do, that object will either appear smaller or bigger (depending on the scaling factor) in the view.

**The Projection Transformation**

As a final step the Projection Transformation is performed. This transformation finalizes what you will see on the screen by establishing the 2D view (based on 3D coordinates of desirably all visible objects' vertices and the camera position) as seen from the current camera view.

## The Viewport Transformation

Once we have the 2D view of our 3D scene. The Viewport Transformation stretches that view into the OpenGL window.

**(i) SCALING :** One of other types of 3D transformations is called scaling. Scaling is simply achieved by multiplying all vertices of an object by the amount you want to reduce or increase the size of that object. Scaling can be used in horizontal only or vertical only dimension to achieve stretching or shrinking of the object

**(ii) TRANSLATION :** An object can be moved (we will use the term *translating* from now on because this term is more common among 3D programmers) on all 3 axis (X Y and Z) in either negative or positive directionThe axis an object can be rotated about are different from the axis it is translated on.

**(iii) ROTATION :** Rotation is identical to translating. The function OpenGL provides you with to achieve rotation is glRotate. The variation of this function I will use is glRotatef, In theory this function multiplies the current matrix by a rotation matrix which in turn gives us rotation.

How exactly this is done will be covered in my matrix tutorial and is purely mathematical. glRotatef takes 4 parameters: the angle of rotation, and the direction of rotation specified by x, y and z coordinates, in the following form glRotatef(float angle, float x, float y, float z).

**Headers Defined**

The in-built are defined in the OpenGL library. Some of the headers that are used as follows:

* **#include<stdio.h>** : to take input from standard input and write to standard output.
* **#include<stdlib.h>** : to include standard library functions.
* **#include<GL/glut.h>** : to include glut library files.
* **#include<GL/gl.h**> : to include OpenGL-1.1 function and token declarations.
* **#include<GL/glu.h**> : to include a set of utility functions that make some OPENGL operations, easier to perform.
* #**include<string.h>**:to include all string related functions.
* **#include<math.h>**:to include all the mathematical defined functions.

**6.3 Chapter summary**

Most of the functions are two dimensional except which draws a 3d bar, you can also implement these functions using already existing algorithms. Every function is discussed with the arguments it needs. Pseudo code tells us how a function will work in English language, so that even a non- programmer can understand what the work is done by functions in the program.

**CHAPTER 7**

**TESTING**

**7.1 Introduction to testing**

Verification and validation is a generic name given to checking processes, which ensures that the software confirms to its specifications and meets the demands of the user.

* **Validation**

Are we building the right product?

Validation involves checking that the program has implanted meets the requirements of the users.

* **Verification**

Are we building the right product?

Validation involves checking that the program confirms to its specification.

**7.2 Stages in the implementation of testing**

* **Unit testing**

Each individual unit is tested for correctness. These individual components will be tested to ensure that they operate correctly.

* **Integration testing**

Each individual unit is integrated into one system and tested completely. This

type of testing is called integration testing.

* **System testing**

The sub-systems are integrated to form system. The errors that result from unanticipated interaction between sub-systems and system components are removed.

**7.2.1 Test case**

Careful planning is required to the most of the testing and controlled testing cost.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test**  **Case ID** | **Test Case Description** | **Input** | **Actual Output** | **Expected Output** | **Remarks** |
| 1. | Wired Torus  Movement  demonstration | Left Click on  Torus  from mouse | It is displaying a  Wired Torus as  expected.  Refer figure 8.3 | It should  display wired  torus in the  specified color | Pass |
| 2. | Wired Teapot  Movement  demonstration | Left Click on  Teapot  from mouse | It is displaying a  Wired Teapot as  expected.  Refer figure 8.4 | It should  display wired  teapot in  the specified  color | Pass |
| 3. | Solid Torus  movement  demonstration | Left Click on  Torus  from mouse | It is displaying a  Solid Torus as  expected.  Refer figure 8.5 | It should display  solid Torus in the specified color | Pass |
| 4. | Solid  Dodecahedron  movement  demonstration | Left Click on  Dodecahedron  from mouse | It is displaying a  Solid Dodecahedron as expected.  Refer figure 8.6 | It should display solid Dodecahedron  in the specified color | Pass |
| 5. | Solid Icosahedron  movement  demonstration | Left Click on  Icosahedron  from mouse | It is displaying a  Solid Icosahedron  as expected. | It should display  Solid Icosahedron  in the specified  color | Pass |
| 6. | 3D Sierpinski  Gasket  movement  demonstration | Left Click on 3D Sierpinski  Gasket from  mouse | It is displaying 3D Sierpinski  Gasket as expected. | It should display 3D Sierpinski  Gasket in the specified color | Pass |
| 7. | Rotation  Demonstration | Left Click on  Rotation  Demonstration  from mouse | It is displaying  Rotation Demonstration as expected.  Refer figure 8.7 | It should display  Rotation demonstration  object in the specified color | Pass |
| 8. | Translation  Demonstration | Left Click on  Translation Demonstration  from mouse | It is displaying  Translation Demonstration as expected.  Refer figure 8.8 | It should display  Translation demonstration  object in the  specified color | Pass |
| 9. | Scaling  Demonstration | Left Click on  Scaling Demonstration  from mouse | It is displaying  Scaling  Demonstration as expected.  Refer figure 8.9 | It should display  Scaling  Demonstration  object in the  specified color | Pass |
| 10. | Example | Right Click and  Left Click on Example  from mouse | It is displaying Demonstration of Transformation  Functions as expected.  Refer figure 8.10 | It should display  Demonstaration of Transformation function specified | Pass |

**Chapter summary**

Inaccurate project planning can have a serious implication on the test execution of the system under test and on the overall quality of the product as a whole. the testshave to identify and hence determine what items have to be tested. These items are heavily based on how the end user will consume the system and hence has to be measurable, detailed and meaningful.

**CHAPTER 8**

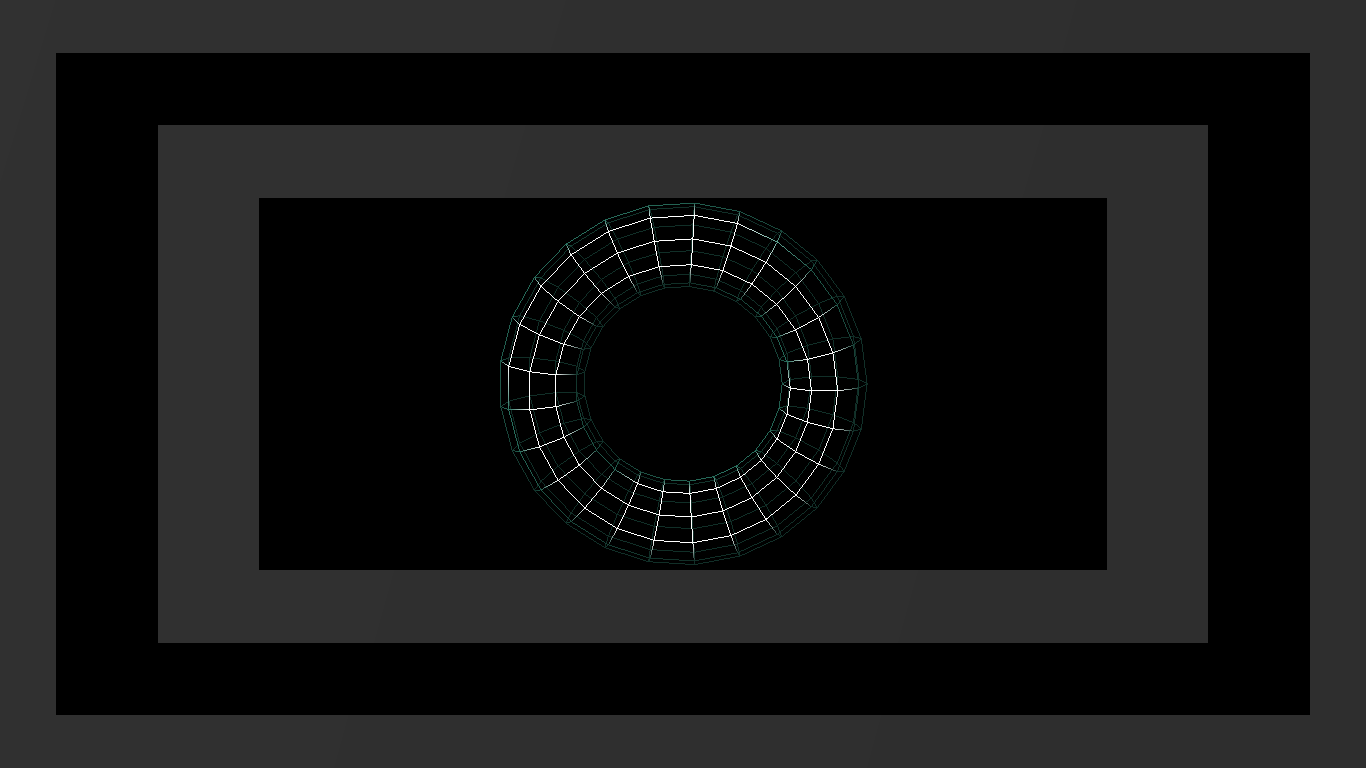
**SNAPSHOTS**

****

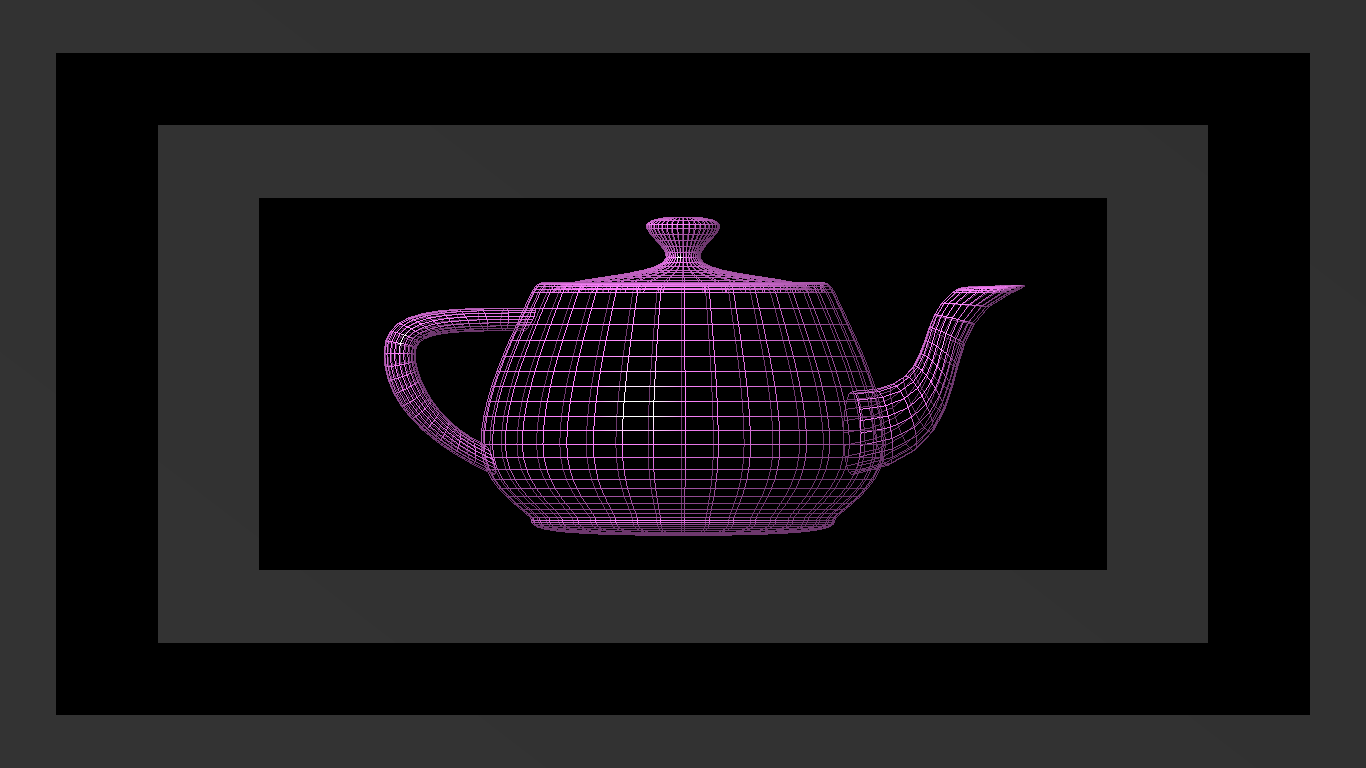
**Figure 8.1** Welcome screen

****

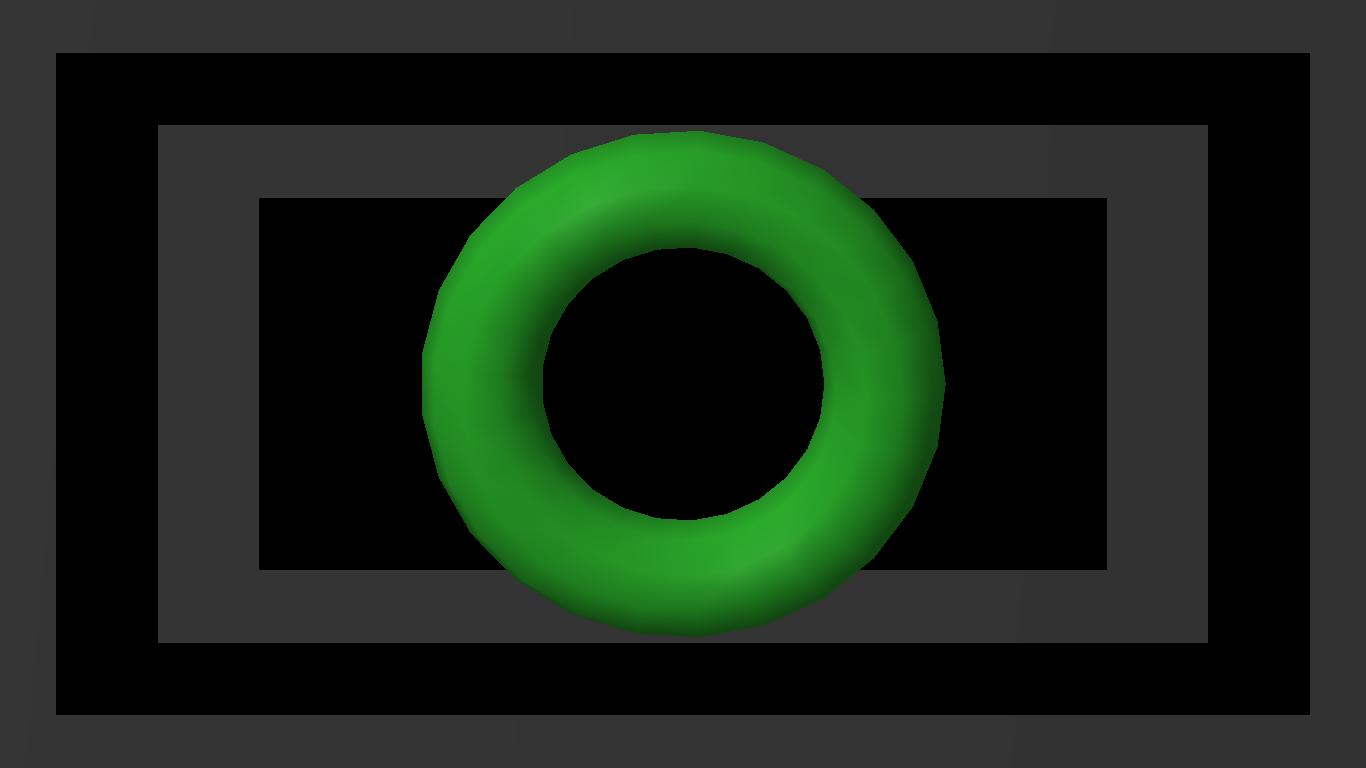
**Figure 8.2** Welcome screen with mouse interface

****

**Figure 8.3** Wired Torus.

****

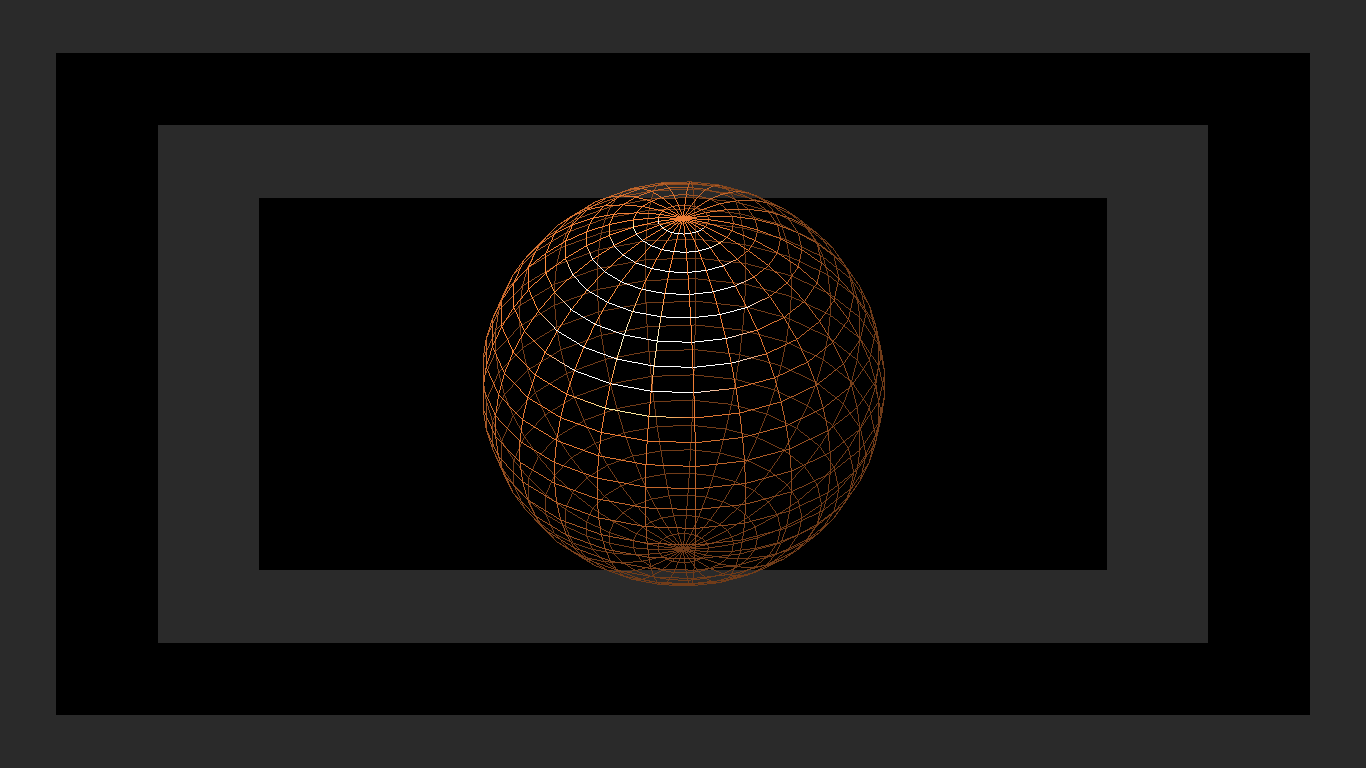
**Figure 8.4** Wired Teapot.



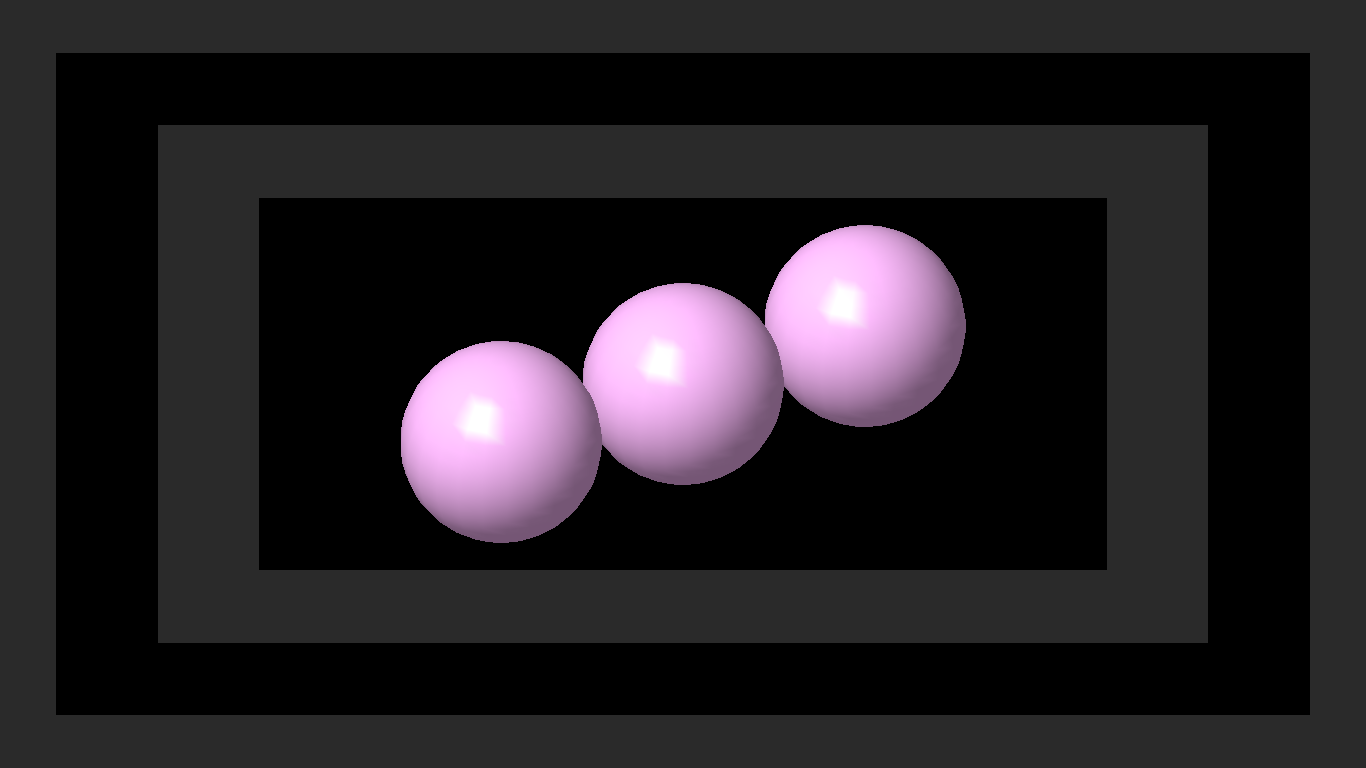
**Figure 8.5** Solid Torus.



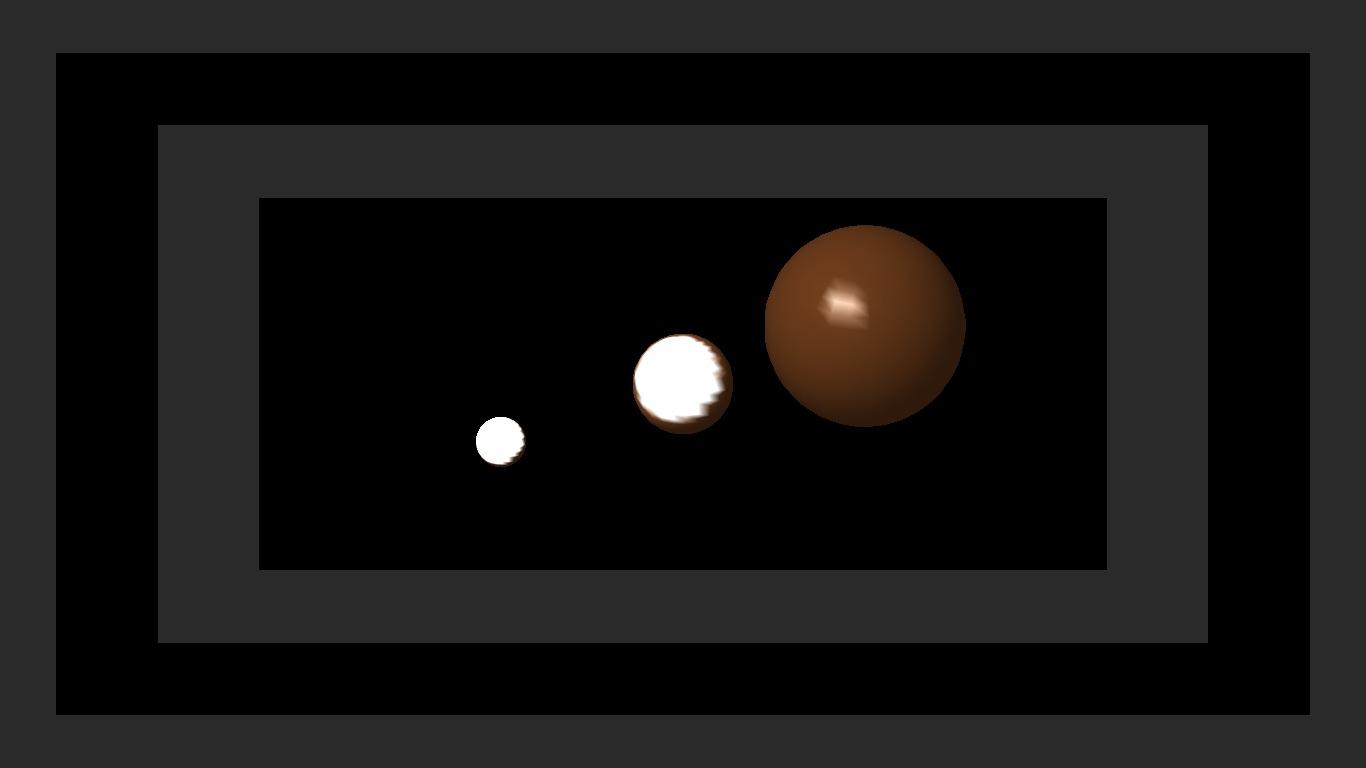
**Figure 8.6** Solid Dodecahedron.



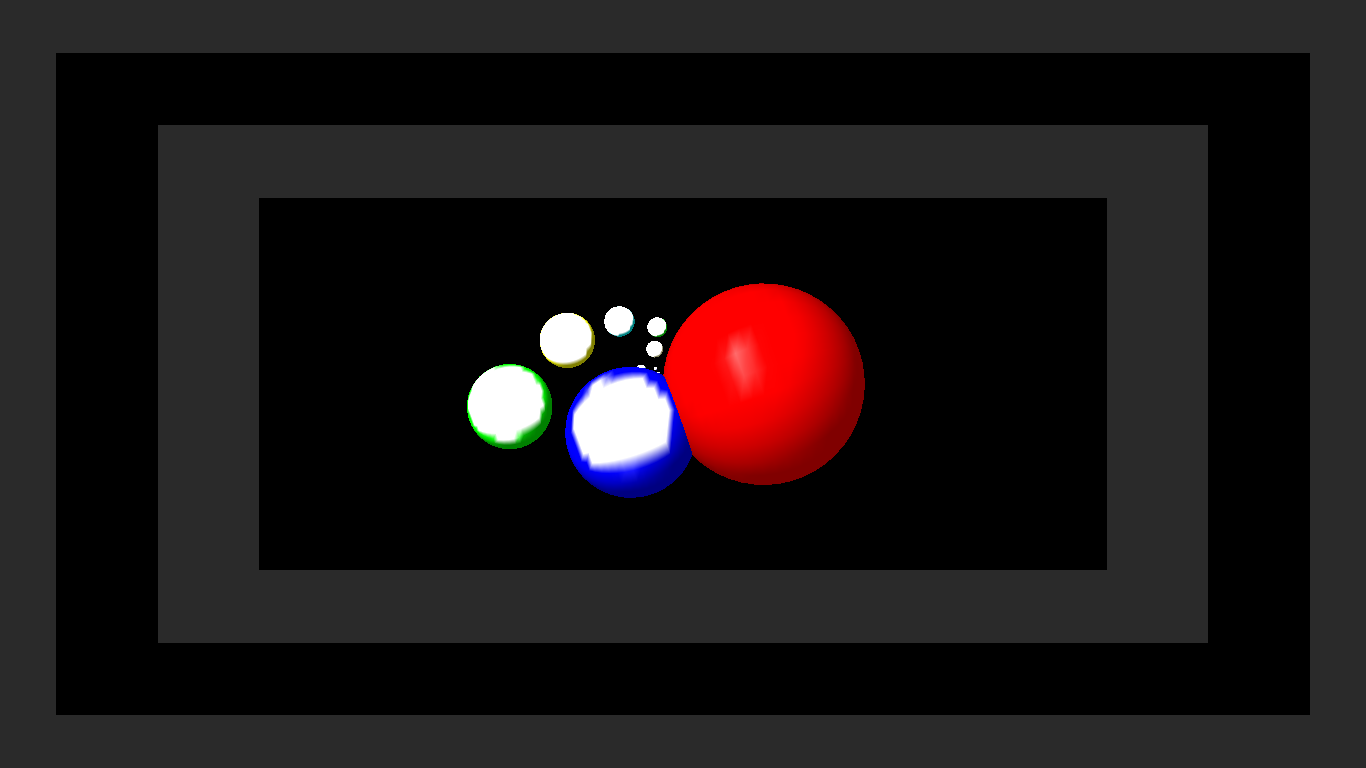
**Figure 8.7** Rotation Demonstration.



**Figure 8.8** Translation Demonstration.



**Figure 8.9** Scaling Demonstration.



**Figure 8.10** Example for demonstrating all three Transformations.

**CONCLUSION**

This project was implemented using OpenGL. This project provides us the information about OpenGL Objects. This project also helps us to analyze how we can use the different standard functions for creating the animated objects and can be used for making a gaming package. This package is very useful for the users since it provides the basic information about various OpenGL functions used for creating and transforming objects.

This product has been demonstrated to fulfill the requirements. The functionality of all the modules and the module level integration is found to be satisfactory.

* This project is very user friendly.
* The interface is mouse driven and the user can select a function by clicking on an icon representing that function.
* One of the important feature of this project is it has extracted the advantage of openGL supporting graphical concepts such as transformation and shadowing.

We have tried our level best to make this project very realistic, so that the user does not face any trouble in observing the different graphical effects when switching from one mode to other.

**FUTURE ENHANCEMENTS**

The Graphics simulator is done as a part of laboratory curriculum due to critical time constraints all features could not be implemented. Features that can be added to enhance .user friendliness and performance of graphics simulator are as follows,

* Mouse Interface.
* Menu based Interface
* Play mode to built like a game.
* Giving a try to user to create more number of objects.

**BIBILIOGRAPHY**

**Books referred**

* Interactive Computer graphics A Top Down Approach with OpenGL-Edward Angel,5th Editon, Addison Wesley,2008.
* Computer Graphics-James D Foley, Adries Van Dam, Steven K Feiner , Hughes.
* Computer Graphics using OpenGL-F.S.Hill, jr 2nd Edition.
* Computer Graphics-OpenGL Version-Donald Hearn and Pauline Baker

**Websites referred**

<http://www.>google.co.in

<http://www.sourceforge.net>

<http://www.jaapsch.net/puzzles/hanoi.htm>

[www.dreamincode.net](http://www.dreamincode.net)

http://www.opengl.org

<http://unreal.srk.fer.hr/theredb>

<http://www.sun.com/software/graphics/opengl/index.xml>