**A**

**Training Report**

**On**

**3D Object Modeling & Visualization Using OpenGL**

**[Work Done at DRDO ,Jodhpur]**

*Submitted in partial fulfillment for the*

*B.Tech Degree of Rajasthan Technical University*



**Submitted To:** **Submitted By:**

**Mr. Manish Choudhary Ms. Amita Mehra**

**Scientist ‘B’ III Year (VI Sem)**

**INFORMATION TECHNOLOGY**

**JIET SCHOOL OF ENGINEERING & TECHNOLOGY FOR GIRLS**

**JIET UNIVERSE.N.H.-65, PALI ROAD, JODHPUR-342 002(RAJ)**

**CERTIFICATE**

This is to Certify That **Ms. AMITA MEHRA** student of **B-TECH III Year information Technology** ,**JIET School of Engieering & Technology For Girls** has successfully completed the project titled **“3D Object Modeling & Visualization Using OpenGL”**

under my supervision at Defence Research And Development Organization, Defence Laboratory, Jodhpur.

She has fulfilled the tenure successfully from 23*rd* May– 30th June 2012 for completing this project.

She has worked very sincerely throughout the tenure and has enthusiasm and ability to learn and implement.

**DATE Mr. Manish Choudhary**

**30-6-2012 Scientist ‘B’**

**Defence Laboratory**

**Jodhpur**

**Acknowledgement**

I take immense pleasure in thanking **Dr. S. R. Vadera** (Director, Defence Lab Jodhpur) for having permitted me to carry out this project work. I would also like to forward my sincere thanks to thank **Sh. N. K. Agarwal (**Oi/c Visual Camouflage, Defense Lab Jodhpur) for his cooperation and helping me in earning this learning opportunity.

I wish to express my deep sense of gratitude to **Sh. Manish Choudhary (**Sc ‘B’) for his able guidance and useful suggestions, which helped me in completing the project work, in time.

I extend my genuine thanks to **Sh. R.K. Khatri (**Sc ‘D’), **Dr. M. C. Vyas (**Sc ‘C’) and **Sh. Madhu Sudan** (STA ‘C’) for their kind help, cooperation to develop this project and providing essential resources to carry out project work.

I would also like to thank **Dr. Raj Bala Chaudhary** (Oi/c Tech. Coord of Training) for her cooperation and support for registration and enrolment.

I pay my special thanks to my **Teachers**  who encouraged me and provided me full support during entire project development.

Finally, yet importantly, I would like to express my heartfelt thanks to my beloved **Parents** for their blessings and wishes for the successful completion of this project.

Yours FaithFully

Amita Mehra

**Contents**

|  |  |
| --- | --- |
|  | Profile of Organization |
|  | Detail Report of Work Done |
|  | Project Brief |
|  | System Analysis |
|  | Technical Details |
|  | Design |
|  | Testing |
|  | Implementation & Output |
|  | Future Enhancements |
|  | Bibliography |

1. **About The Organization**

**Defence Laboratory, Jodhpur** established in 1959, is a laboratory of the Defence Research and Development Organisation, Ministry of Defence, Govt. of India, located in Ratenada Palace, Jodhpur. This laboratory is working in the field of problems related to Defence Services in desert environment. Its motto is to enhance operational capability of defence services in the harsh and hot desert climate. This lab is working in the following three major research areas:

1. Camouflage and Low observable technologies
2. Desert Environment Science and technology
3. Nuclear Radiation Management and Application
4. **Detail Report of Work Done**

Computers have become powerful tool for rapid and economical production of pictures. Computer Graphics and visualization is used routinely in diverse areas of science, engineering, medicine, education etc. It improves communication between human and machine. Videogames, animation, multimedia are the well-known examples of graphics being used in present scenario.

Initially the dependence of applications on hardware was up to large extent but with the advancement in the computer technology, producing an interactive graphical application has become an easy task and that too in windows environment. Most of the utilities and drivers available for Unix environment are now also available on windows platform.

Good 3-D capabilities are becoming a increasingly important part of many fields including CAD/CAM, architectural design, product design, financial analysis, computer animation and simulation, multimedia applications, World Wide Web applications, statistical and scientific visualization, medical imaging, publishing and game design and their implementations.

Keeping above in view this project has been conceived and undertaken with regular geometric 3D Object’s creation, rendering, manipulation and surface development.

The term three-dimensional, or 3D, means that an object being described or displayed has three dimensions of measurement: width, height, and depth.Three-dimensional graphics have many uses in modern computer applications. Applications for real-time 3D graphics range from interactive games and simulations to data visualization for scientific, medical, or business uses. Higher-end 3D graphics find their way into movies and technical and educational publications as well.3D computer graphics are actually two-dimensional images on a flat computer screen that provide an illusion of depth, or a third dimension.

OpenGL is a programming interface for creating real-time 3D graphics.It is strictly defined as “a software interface to graphics hardware.” In essence, it is a 3D graphics and modeling library that is highly portable and very fast. Using OpenGL,

we can create elegant and beautiful 3D graphics with exceptional visual quality. Thegreatest advantage to using OpenGL is that it is orders of magnitude faster than a raytracer or software rendering engine.

This project has been conceived and undertaken with regular geometric 3D Object’s creation, rendering, manipulation and surface development.. The software has been developed using VC++ and OpenGL library This project is a step towards development of complex software for graphics application.

.

1. **Project Brief**

The project aims toward building a 3D scene Generator that allows development and manipulation of regular 3D objects as well as irregular 3D surfaces. The software has been developed using VC++ and OpenGL library. No other already available graphics application is used. This project is a step towards development of complex software for graphics application.

Following points give an overview on the objective of the project :

* Creation of regular 3D objects like cube, cylinder, sphere etc as well as irregular 3D surfaces.
* To perform different transformations on the object like translation, scaling and rotation.
* To provide lighting effects and material properties so as to allow realistic display of the object.
* To get the visible surface according to the Viewpoint.
* To open the final visible object with respect to any of its surface.
* To allow texture mapping.
* Finally allow all the above tasks through user-friendly interface.

1. **System Analysis**

An efficient and error free system can be developed with the help of System Analysis and design. It is part of System Development Life Cycle, which cannot be overseen .It helps to design a better product taking in consideration the need of customer.

System Analysis is the process of gathering and interpreting facts, diagnosing problems and using this information to make improvement in the system. It defines the role of each element in a computer-based system, ultimately allocating the role that the software will play.

To accomplish this task, a study was conducted to do the following:

* Learned details of previously developed system
* Documentation of features of the new system being developed, for understanding its components (and their interrelationship), doing so in a manner that will allow the development of new system to be managed.
* Involved the user to draw on their expertise and knowledge as well as learn their ideas, opinions about the new system.
  1. **Problem Identification :**

Considering the great impact of the System Analysis on the overall project development, a systematic and detailed analysis was conducted for developing the software “*3D Object Modeling & Visualization Using OpenGL*”. Study was conducted to determine the initial scope of the system. Preliminary investigations comprising of request clarification, feasibility study etc were carried out.

Requirement of the organization was the development of basic software, which would help into further development of the software for 3D Objects and their manipulations.

**Requirements :**

Requirement of development of this software is producing 3D scenes with all following capabilities :

* Development of 3D objects.
* Change color of object.
* Change background color.
* Producing transformation effects like translation,scalling,rotation.
* Producing lighting effects.
* Change shade of object.
* Providing Texture Mapping.

For generation of 3D scenes OpenGL wws the best option. So the software is developed using OpenGL and VC++.

* 1. **Feasibility Study :**

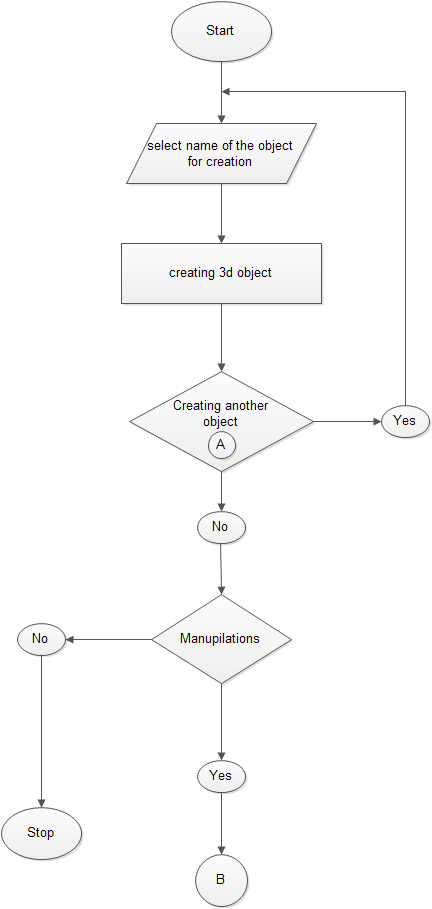
After the study of the existing system comes the feasibility study. It includes the investigation of the information needs of the objectives, constraints, basic resource requirement and cost of proposed system. This can be evaluated in terms of following categories.

* Economic feasibility
* Operational feasibility
* Organizational feasibility

All the above-mentioned tests were conducted and the project was seen to have satisfactorily passed all the tests. Thus the project is feasible to the Organization.

* 1. **Requirement Gathering :**

It is aimed at requiring, detailed understanding of all the important facts of the system under consideration. To understand the nature of operations to be performed it is essential to understand the information domain for software as well as required functions, performance and interfacing. The detailed requirement analysis activity was performed to determine precisely

* How the objects need to be created so that further manipulations become possible
* What are the future enhancements that would be required
* The storage for objects
* A suitable interface
* Inputs required by the system
* Data entry formats
* The source of information
* Authencity of collected information
  1. **Flowcharts:** 

* 1. **Data Flow Diagrams :**

1. **Technical Details**
   1. **Hardware Requirements :**

Processor : Pentium Processor

RAM : 32MB

Hard disk : 50MB

Display : SVGA

* 1. **Operating System and Development :**

Operating System : Windows XP

Tools and Packages : Microsoft Visual Studio 2008,

Visual C++ with MFC classes,

OpenGL 1.0

OpenGL and MFC classes used to write the software has been described briefly in the following paragraphs.

* 1. **OpenGL :**

OpenGL is strictly defined as “a software interface to graphics hardware.” In essence, it is a

3D graphics and modeling library that is highly portable and very fast. It is a software interface to graphics hardware. This interface consists of more than 700 distinct commands that we use to specify the objects and operations needed to produce interactive three-dimensional applications.

Understanding 3-D graphics programming techniques require a strong background in mathematics. Many of the today’s 3-D graphics programming tools hide most of the mathematics details from the programmer leaving him to worry about creating images rather than understanding the trigonometry and differential calculus involved in it. One such tool is **‘Open GL’**, a library of graphics routines that makes sophisticated 3-D graphics programming accessible to common programmers.

OpenGL is intended for use with computer hardware that is designed and optimized for the display and manipulation of 3D graphics. Software-only implementations of OpenGL are also possible, and the older Microsoft implementations, and Mesa3D (www.mesa3d.org) fall into this category. Apple also makes a software implementation available on OS X as a fallback mechanism. With these software-only implementations, rendering may not be performed as quickly, and some advanced special effects may not be available at all. However, using a software implementation means that your program can potentially run on a wider variety of computer systems that may not have a full-featured 3D accelerated graphics processor available.

OpenGL is used for various purposes, from CAD engineering and architectural applications to modeling programs used to create computer-generated monsters and machines in blockbuster movies. The introduction of an industry-standard 3D API to mass-market operating systems such as Microsoft Windows and the Macintosh OS X has some exciting repercussions. With hardware acceleration and fast PC microprocessors now commonplace, 3D graphics have become standard components of consumer and business applications, not only of games and scientific applications.

* + 1. ***Advantages of OpenGL :***

Using OpenGL, we can create elegant and beautiful 3D graphics with exceptional visual quality. The greatest advantage to using OpenGL is that it is orders of magnitude faster than a ray tracer or software rendering engine. Initially, it used algorithms carefully developed and optimized by Silicon Graphics, Inc. (SGI), an acknowledged world leader in computer graphics and animation. Over time, OpenGL has evolved as other vendors have contributed their expertise and intellectual property to develop high-performance implementations of their own.

The OpenGL API itself is not a programming language like C or C++. It is more like the C runtime library, which provides some prepackaged functionality. On the other hand, the OpenGL specification includes GLSL, the OpenGL Shading Language, which actually is a very C-like programming language. GLSL, however, does not control your application’s flow and logic, but rather it is intended for rendering operations. At a high level, application programs are not written in OpenGL, as much as they use OpenGL. There really is no such thing as an “OpenGL program” (with the aforementioned exception of shader programs) but rather a program the developer wrote that “happens” to use OpenGL as one of its Application Programming Interfaces (APIs). You might use the C runtime library to access a file or the Internet, and you might use OpenGL to create real-time 3D graphics.

# *OpenGL Architecture :*

OpenGL is designed as a streamlined, hardware independent interface to be implemented on many different hardware platforms. It is a procedural rather than a descriptive graphics API. Instead of describing the scene and how it should appear, the programmer actually prescribes the steps necessary to achieve a certain appearance or effect. These “steps” involve calls to the many OpenGL commands. These commands are used to draw graphics primitives such as points, lines, and triangles in three dimensions. In addition, OpenGL supports texture mapping, blending, transparency, animation, and many other special effects and capabilities.

The following list briefly describes the major graphics operations that OpenGL performs to render an image on screen.

* Construct shapes from geometric primitives, thereby creating mathematical descriptions of objects.
* Arrange the objects in 3-D space and select the desired vantage point for viewing the composed scene.
* Calculate the colors of all the objects. The colors might be assigned by the application determined from specified lighting conditions obtained by pasting a texture onto the objects or some combination of these 3 actions.
* Convert the mathematical description of objects and their associated color information to pixels on the screen. This process is called ‘Rasterization’.

During these stages, OpenGL might perform other operations such as eliminating parts of objects that are hidden by other objects.

* + 1. ***Features of OpenGL :***

Some of the main features of OpenGL include the following:

* **Transformation matrices**:The ability to change the location, size and perspective of an object in 3-D co-ordinate space. This gives you the freedom to change an object’s size and location just by modifying the elements of matrix.
* **Lighting effects**: The ability to calculate the effects on the lightness of a surface’s color when different lighting models are applied to the surface from one or more light sources.
* **Smooth shading**:The ability to calculate the shading effects that occur when light hits a surface at an angle and results in subtle color differences across the surface. This effect is important for making a model look realistic.
* **Material properties**: The ability to specify the material properties of a surface. These properties modify the lighting effects on the surface by specifying such things as the dullness or shininess of the surface.
* **Alpha blending**: The ability to specify an alpha, or ‘opacity’, value in addition to the regular RGB values. The alpha component is used to specify the opacity, allowing the full range, from completely transparent to totally opaque. When used in combination with Z-buffer, alpha blending gives the effect of being able to see ‘through’ objects.
* **Texture mapping**: The ability to apply an image to a graphics surface. This technique is used to rapidly generate realistic images without having to specify an excessive amount of detail e.g. you could create a wooden floor by painting the floor’s rectangular surface with a wood grain texture.
* **Z-Buffering**:The ability to calculate the distance from the viewer’s location. This makes it easy for the program to automatically remove surfaces or parts of surfaces that are hidden from view.
* **Double-Buffering**: Support for smooth animation using double buffering. A smooth animation sequence is achieved by drawing into the back buffer while displaying the front buffer and then swapping the buffers when you’re ready to display the next animation sequence.
  + 1. ***Few terms used in OpenGL code :***

RENDERING is the process by which a computer creates images from models. These MODELS or objects are constructed from geometric primitives-points, lines and polygons that are specified by their VERTICES. The final rendered image consists of pixels drawn on the screen; a PIXEL is the smallest visible element the display hardware can put on screen. Information about the pixels is organized in memory into BITPLANES. A bitplane is an area of memory that holds one bit of information for every pixel on the screen; the bit might indicate how red a particular pixel is supposed to be. The bitplanes are themselves organized into a FRAME BUFFER, which holds all the information that the graphics display needs to control the color and intensity of all the pixels on the screen.

* + 1. ***OpenGL command syntax:***
* OpenGL commands use the prefix ‘gl’ and initial capital letters for each word making up the command name (glClearColour()).
* OpenGL defined constants begin with ‘GL\_’, use all capital letters and use underscores to separate words (GL\_COLOUR\_BUFFER\_BIT).
* Some OpenGL commands have appended letters like ‘3f’, ‘4i’ etc. The ‘3’ part of the suffix indicates that 3 arguments are given. The ‘f’ part of the suffix indicates that the arguments are floating point numbers.
  + 1. ***The OpenGL Library :***

As mentioned earlier, OpenGL is a library of graphics routines that makes sophisticated 3-D graphics programming accessible to the programmers. A programmer can do anything from displaying simple shapes to composing animated 3-D scenes. There are more than 100 functions available for this purpose. In addition to the core functions, Windows 95 supports four other categories of functions. These are:

* Open GL utility library.
* Open GL auxiliary library.
* Wiggle functions unique to windows.
* New Win32 API functions.

The core set of 115 Open GL functions represent the basic set of functions that should be implemented on any Open GL platform. These functions allow the programmer to create various

* Types of shapes,
* Produce lightening effects,
* Incorporate texture mapping,
* Perform matrix transformations and much more.

In fact these core functions appear in several forms for e.g. these functions like vertex3f(),vertex3d(),vertex3dv(),vertex3fv(),vertex3I() etc; all attempting to define a vertex and if you consider these as different functions then these are well over 300 functions in the core set itself.

* + - 1. ***Open GL Utility Library :***

The functions under this category begin with a prefix **‘glu’**. These functions actually calls on the core set to do their work. Their job is to simplify the use of texture images, perform high-level co-ordinate transformations, and render such polygons based objects as spheres, cylinders and disks.

* + - 1. ***Open GL Auxiliary Library :***

These functions have a word ‘aux’ attached at the beginning of their name. These are platform dependent functions. They carry out tasks like managing windows, handling I/O, and drawing certain 3-D objects. You got to exercise caution while using these functions since their use is likely to reduce the portability of your program.

* + - 1. ***The WGL Functions :***

These are six functions under this category and their names begin with ‘wgl’ and hence the names wiggle functions. These functions link OpenGL to windows, enabling the programmer to create and select rendering context (the OpenGL version of a device context) and create bitmapped fonts that are used to place text in OpenGL windows and these functions are unique to windows implementation of OpenGL.

* + - 1. ***New Win32 Functions :***

Unlike others, these functions do not begin with a prefix. They deal with pixel formats and double buffering and these functions being extensions to the win32 system aren’t implemented on other OpenGL functions.

* + 1. ***Open GL Data Types:***

In windows programming we use many special data types like:  **RECT, POINT, HDE, HWND**  etc, for commonly needed values. Similarly, OpenGL too define many data types that we can use in OpenGL programs. Some of these data types are shown in the table:

|  |  |
| --- | --- |
| **Data types** | **Equivalent** |
| GLbyte | Signed char |
| Glshort | Unsigned char |
| Glshort | Short |
| Glushort | Unsigned short |
| Glint | Long |
| GLuint | Unsigned |
| Glfloat | Float |
| Gldouble | Double |
| Glboolean | Unsigned |
| Glvoid | Void |
| HGLRC | HGDLOBJ |

* + 1. ***Device Context :***

GDI is capable of drawing to the screen, to memory, to printers or to any other device that provides a GDI interface layer and that can process GDI calls. GDI accomplishes this by a Rendering ‘handle’ to the currently selected device. This handle is called the ‘Device Context’or DC. All GDI calls pass through a DC and DC does the correct operation.

* + 1. ***Rendering Context :***

A Rendering Context is OpenGL’s version of a Device Context (DC). As we know, DC information that specifies pen and brush colors, drawing mode, mapping modes, palette contents etc. These parameters decide how graphical information would be displayed in a window. To draw anything in a window it is necessary to first create a device context and this is usually done by following set of statements:

CCLientDC d (this);

d.Rectangle(10,10,100,100);

Unless the device context is created we cannot call the CDC::Rectangle() function windows based OpenGL programs use DC’s, just like any other windows programs.

* + 1. ***Pixel Format :***

Before a program creates a rendering context, it must set the device’s ‘pixel format’, which contains the attributes for the device’s drawing surface. These attributes include: whether the drawing surface uses RGBA or indexed colour mode, whether the pixel buffer uses single or double buffering, the number of colour bits, the number of bits used in the depth and stencil buffer and other OpenGL graphical information.

The win32 functions that manage pixel format are:

|  |  |
| --- | --- |
| Function name | Use |
| **ChoosePixelFormat()** | **Returns the pixel format that most closely matches the requested pixel format** |
| **DescribePixelFormat()** | **Obtains information about a given pixel format** |
| **SetPixelFormat()** | **Sets the pixel format of the given device context** |
| **GetPixelFormat()** | **Gets the pixel format of the given device context** |

Depending on the capabilities of the displayed device it would support a specific number of pixel formats. The attributes of a particular pixel format are represented by a 26-field structure called ‘PIXELFORMATDESCRIPTOR’. To set up a pixel format we have to fill this structure and then pass it’s base address to the SetPixelFormat() function.

* + 1. ***OpenGL for windows :***

OpenGL programs make heavy use of Microsoft’s Visual C++ programming platforms, the C++ programming language and the Microsoft Foundation Classes (MFC). These are used as wrappers to the generic Windows API that starts off with a generic ‘C’ application that can be compiled from any 32-bit ‘C’ compiler that provides access to the OpenGL Headers.

Six basic steps required to use OpenGL calls in a Windows Program are:

* Getting a Device Context for the rendering location.
* Selecting and setting a Pixel Format for the Device Context.
* Creating a Rendering Context associated with the Device Context.
* Drawing using OpenGL commands.
* Releasing the Rendering Context.
* Releasing the Device Context.

# *How do 3-D objects appear realistic?*

By default OpenGL displays a 3-D objects using orthographic (parallel) projection, e.g. though the back of the cube is farther away, it is shown to be of the same size as the front of the object. And so for making the objects more realistic they have to display it using “perspective projection” which is done in following three steps:

**Step 1:** **Applying the projection transformation:**

* For using perspective projection, projection transformation is to be performed. The following code does this:

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glFrustum(-1.0,1.0,-1.0,2.0,7.0);

* The first statement selects the OpenGL’s projection matrix to apply the perspective projection to the 3-D object to be displayed.
* The second statement loads the currently related matrix with the identity matrix, which ensures the matrix start off fresh.
* The third statement limits the viewing volume, which is the 3-D area in which our object exists. The glFrustum() function creates a perspective
* Matrix and multiplies it with the currently selected matrix, the six arguments passed to this function define the left, right, bottom, top near and far co-ordinates of the viewing volume.

**Step 2:** **Applying the ModelView transformations.**

After defining a frustum for viewing the object, still the object remains outside the specified area. Moving the object along the z-axis, which is done by the following code, can solve this problem:

glMatrixMode(GL\_MODELVIEW)

glLoadIdentity();

glTranslate(0.0f,0.0f,-3.5f);

* The first function selects Model View matrix.
* The second function initialises the model view matrix to the identity matrix and,
* The third command translates the cube to the specified place along the z-axis. The glTranslatef() function creates a transaction matrix and multiplies it with the currently selected matrix, while the three arguments specify the amount of translation on the x, y and z axes respectively.

**Step 3: Rotating the object.**

Even after putting the object inside the viewing volume, only the front face of the object is visible. So to make other faces visible, it is required to rotate the object and this is done by the following code:

glRotatef(degree, x-axis, y-axis, z-axis);

* + 1. ***Modeling Transformations :***

In many graphic applications, there is a need for altering or manipulating displays. Arranging the orientations and sizes of the component parts of the scene creates design applications and facility layouts. Changes in orientation, size and shape are accomplished with geometric transformations that alter the co-ordinate descriptions of objects. The basic geometric transformations are:

* Translation
* Rotation
* Scaling

Modeling transformations are used to manipulate your model and the particular objectswithin it. These transformations move objects into place, rotate them, and scale them.Following figure illustrates three of the most common modeling transformations that we will apply to our objects. Figure a shows translation, in which an object is moved along a given axis. Figure b shows a rotation, in which an object is rotated about one of theaxes. Finally, Figure shows the effects of scaling, where the dimensions of the object are increased or decreased by a specified amount. Scaling can occur nonuniformly (the various dimensions can be scaled by different amounts), so you can use scaling to stretch and shrink objects.

* + 1. ***Lighting 3-D Objects :***

3-D objects can be made to appear more realistic by applying appropriate lighting to them. Adding lighting makes a big difference to how the object looks on the screen. The overall view of the object is governed by 2 factors:

* The Light Source
* The light reflecting properties of the object’s surface.

OpenGL supports 8 different light sources and 4 types of light reflecting properties. Each light source is defined by the position in terms of x, y, and z co-ordinates.

In the OpenGL model, the light sources have effects only when there are surfaces that absorb and reflect light. Each surface is assumed to be composed of a material with various properties. A material might emit its own light, it might scatter some incoming light in all directions, and it might reflect some portion of the incoming light in a preferential direction. The OpenGL lighting model considers the lighting to be divided into four independent components:

* Ambient
* Diffuse
* Specular
* Emissive

All four components are computed independently and then added together.

* + - 1. ***Ambient Light :***

Ambient illumination is light that’s been scattered so much by the environment that its direction is impossible to determine-it seems to come from all directions. It models the combinations of light reflections from various surfaces to produce a uniform illumination, also called ‘Background Light’.

* + - 1. ***Diffuse Light :***

The Diffuse component is the light that comes from one direction, so it’s brighter if it comes squarely down on a surface then if it barely glances off the surface. Once it hits a surface, it’s scattered equally in all the directions, so it appears equally bright.

* + - 1. ***Specular Light :***

Specular light comes from a particular direction, and it tends to bounce off the surface in a preferred direction. On looking at an illuminated shiny surface a highlighted, or bright spot is seen at certain viewing directions. This phenomenon, called Specular Reflection, is the result of total reflection of the incident light in a concentrated region.

* + - 1. ***Emissive Light :***

In the OpenGL lighting model, the Emissive colour of a surface adds intensity to the object, but is unaffected by any light sources. Also, the Emissive colour does not introduce any additional light into the overall scene.

* 1. **Microsoft Foundation Classes :**

At first software developers had to use ‘C’ to build applications to run under Windows. Such programs connected directly to the Windows API a bare bone set of functions provided by the OS. In those days, besides a Windows compatible compiler the only available development tools were packaged by Microsoft into the SDK (Software Development Kit). Learning to use SDK to develop Windows program has always been a difficult task.

Programmers can also use C++ for building Windows applications because of its built in compatibility with ‘C’ it can be used for SDK style programming. However such use doesn’t take advantage of Object Oriented features of ‘C++’. Realising the benefits of those features requires a class library that’s written to simplify the tedious housekeeping chores and frequent quirks that make SDK programming so difficult. To overcome the difficulties Microsoft Foundation Classes (MFC) libraries were developed.

The MFC library was preceded by AFX library, first attempt to make Windows programming Object Oriented. But it was given up, as it was bloated and slow.

The MFC library is an attempt by Microsoft to provide ‘C++’ programmers with an Object Oriented programming interface to Windows. Thus, ‘Visual C++’ is visual development tool that makes use of the MFC library to make the development under Windows environment fast and reliable.

MFC library is a collection of ‘C++’ classes. It is provided as a DLL so your application has access to the classes of MFC. A DLL consist of executable functions that are loaded into memory and are independent from any application. The MFC classes have been built using the operating system’s functions.

* + 1. ***MFC and Windows OS Interaction :***

Windows OS has three major components:

* User
* Graphics Device Interface(GDI)
* Kernel

User is a module of code that services input devices such as keyboard. GDI is a module that services output to graphics devices-screens, printers etc. Kernel services file management and internal memory management.

Collectively these three components are called API. These components interact with the MFC application. An MFC application calls functions in the API.

The base classes in the MFC library incorporate API functions, so the MFC classes also call functions in the API. Each of the three API components is provided as a DLL. An application can call functions in the DLL as though they were part of the application. The API DLLs are normally found in Windows Operating system directory, files required by the system are usually stored. This directory is typically C:/WINDOWS/SYSTEM. The filenames for the three API DLLs are USER.EXE, GDI.EXE and KRNL386.EXE. In WIN32 API the DLLs are stored as USER32.DLL, GDI32.DLL and KERNEL32.DLL.

* + 1. ***MFC Application Architecture Classes :***

MFC provides several architectural classes used to define the architecture of a framework application. These classes supply functionality common to most windows applications. The typical structure of an application is based on one of the following four architecture generalisations:

* **Console Application :**

Console applications are written as command line programs with DOS like interface. Console programs are typically written without making use of the MFC library. Those rare console applications that do use MFC differ from all other types of MFC applications in that they don’t use a CWinApp-derived application object.

* **Frame Window Based Applications :**

Some applications need only minimal user interface and a simple main window. In this scenario, you simply derive an application object [CWinApp] and a frame window object [CframeWnd] from the MFC base classes. Then class member methods and variables and override inherited class methods to define your application’s functionality.

* **Dialog Based Applications :**

The Dialog based application window is derived from CDialog. The CDialog object is attached to the m\_pMainWnd member of the applications CWinApp object.

* **Document/View Applications :**

The document/View application architecture uses the more complex MFC concepts. Documents and Views are the classes where most of the code will go when a doc/view application is written. MFC provides the CDocument class and CView class that allows to design document and view classes respectively.

* + 1. ***The MFC Class Hierarchy :***

The foundation class library’s classes are broken into two main categories: those derived from CObject, and those that aren’t. The CObject derived classes make up the following subsystems that can be further broken into several different categories:

* The root class, CObject
* Array, list and map classes
* Debugging and exception classes
* Drawing and printing classes
* File and database classes
* Internet and networking classes
* MFC application architecture classes
* Window, Dialog and control classes

Classes not derived from CObject are broken into the following application framework subsystems:

* Simple data type classes
* Support classes
* Typed template classes
* OLE classes
* Structure classes
* Synchronisation classes
  + 1. ***Drawing graphics using MFC :***

As discussed earlier, the GDI provides all the basic drawing functionality for Windows; the device context represents the device providing a layer of abstraction that insulates your application from the trouble of drawing directly to the hardware. GDI provides this insulation by calling the appropriate device driver in response to the Windows graphics function calls. When writing MFC programs there is an added benefit when using device contexts and the GDI functions: The GDI functions are built right into MFC device context classes as device context member functions. This makes using the GDI functions very convenient.

* + 1. ***MFC Device Context classes :***

In MFC a Device Context object and it’s member functions are used for drawing. MFC wraps a window’s Device Context and GDI functions that require a Device Context handle, in a class called CDC. For drawing in different situations MFC provides classes called CPaintDC, CClientDC, CWindowDC and CMetaFileDC, all which have been derived from CDC. Of these CPaintDC class represents a paint Device Context and should be used only in OnPaint() handlers. To do drawing in the client area in a handler function other than the OnPaint ( ) handle the CClientDC object must be used. For drawing in entire window including the non-client area CWindowDC object should be used and to store sequence of drawings that can be played back to produce physical output CMetaFileDC object must be used.

* + 1. ***Messages and Message maps :***

A central concept of the Windows operating system is the ‘Message’, or event. An application responds to messages and all the actions result from a window or an application, receiving a message. An MFC application consists of message handler functions, functions that respond to specific messages. When a message arrives, its handler function is entered and the code is executed. If no message arrives, no code is executed.

MFC provides a robust and convenient mechanism for handling messaging within a process: the message map. A message map is an object oriented alternative to the ‘switch: case’ message handling statement found in the Windows procedures of the traditional SDK C programs. Each object is capable of receiving messages or a command owns its own message map. The message map associates a message or command with a specific message handler method.

e.g. of a message map:

BEGIN\_MESSAGE\_MAP(myframe, CframeWnd)

ON\_WM\_CREATE( )

ON\_WM\_PAINT( )

ON\_WM\_RBUTTONDOWN( )

END\_MESSAGE\_MAP( )

These message map macros are predefined by MFC and take no parameters. These macros also expect the predefined message handler methods OnCreate(), OnPaint(), OnRButtonDown() to be present in the class declaration and implementation. The ‘CFrameWnd’ is the base class and ‘myframe’ is the derived class.

The creation of a message map is done using several macros that are defined in the <afxwin.h> header file. The declaration of any class using message maps must contain the macro DECLARE\_MESSAGE\_MAP(). By convention, it is placed at the end of the class declaration. The message map’s table is defined with a set of macros that expand to message map entries. The table begins with BEGIN\_MESSAGE\_MAP() macro call, which defines the class that is handled by this message map and the parent class to which unhandled messages are passed. The table ends with the END\_MESSAGE\_MAP() macro call. Between these two macros, there can be any number of references to messages that will be processed by the program.

1. **Design**

Designing is the process of selecting and documenting the most effective and efficient sys-tem elements that together will implement the soft-ware system requirements. The design represents a specific, logical approach to meet the software requirements.

*Architectural design* also called top-level design or preliminary design—typically defines and structures computer program components, interfaces ..etc. It includes information such as the overall processing architecture, function allocations (but not detailed descriptions), system utilities, operating system interfaces, and storage throughput.

*Detailed design* is equivalent to component engineering. The components in this case are independent software modules and artifacts.

In this software this part of the life cycle of software engineering has been accomplished by

* Allocating Creation of different objects to different functions e.g.: DrawCube(), DrawSphere() etc.
* For manipulations different methods have been used e.g.: Lighting(),Shape\_color() etc.
* Interface Designing

1. **Testing**

Software Testing is critical element of the software quality assurance and represents the ultimate review of specification design and coding. This point was kept in mind throughout the software development activity and a conscious effort was made to test the system as thoroughly as possible. The objective behind was to systematically uncover different classes of errors with minimum amount of time and effort.

Testing Strategy Used

Two complementary test activities were carried out. They are as under:

**White Box Testing**

This method exploits the control structure of the software to test for any as yet undiscovered errors. In this technique:

* All independent paths within a module are exercised at least once.
* All logical decisions are exercised on their true or false side
* All loops are executed at their boundaries and within their operational bounds
* The internal data structures are exercised to ensure their validity.

**Black Box Testing**

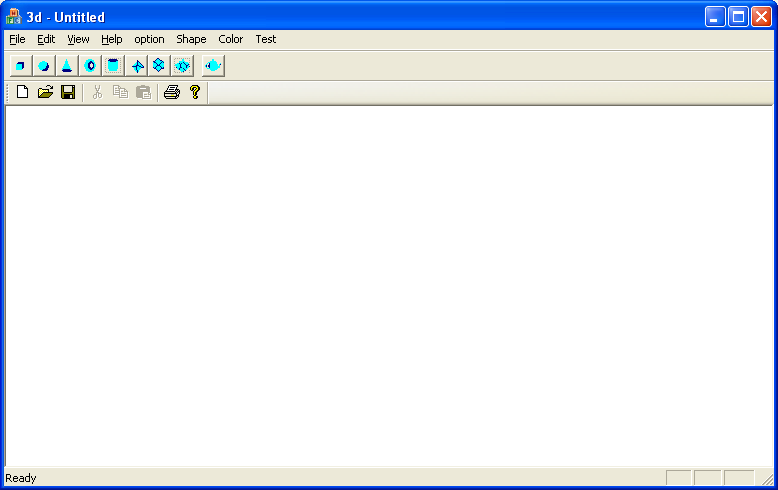
This method concentrates on testing the functional requirement fulfillment by the software and ignores its control structure. It attempts to find errors in the following categories:

* Incorrect or missing functions
* Interfacing errors
* Errors in data structures
* Initialization and termination error

Testing is the process of evaluating software at the end of the development process to ensure compliance with software requirements. It helps us in answering the question whether we are building the right product or not.

1. **Implementation & Output**

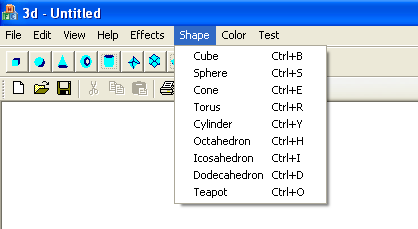
When run the software, this displays following window.



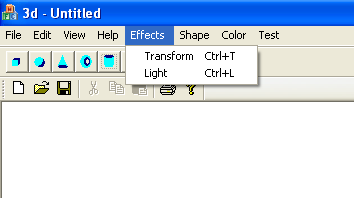
This has following menubar and toolbars.



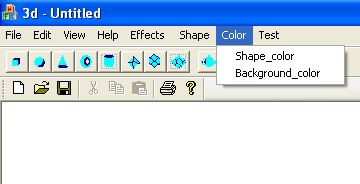
* **Shape Menu :**



* **Effects Menu :**



* **Color Menu :**



* **Transform Dialog Box :**
* Translation :

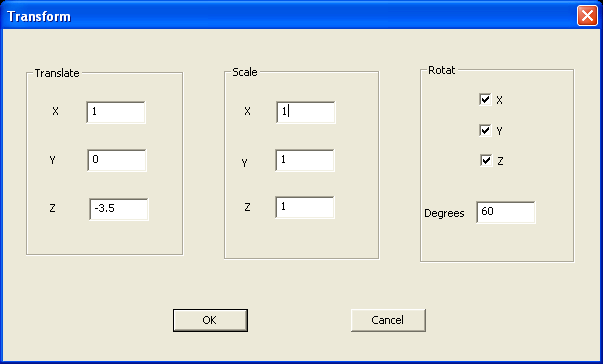
void glTranslate{fd}(TYPE x, TYPE y, TYPE z);

* Rotation :

void glRotate{fd}(TYPE angle, TYPE x, TYPE y, TYPE z);

* Scaling :

void glScale{fd}(TYPE x, TYPE y, TYPE z);



* **Color Dialog Box :**

CColorDialog cd1;

if(cd1.DoModal()==IDOK)

{

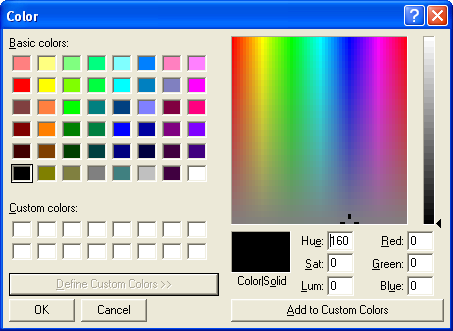
cl=cd1.GetColor();

rr=((GetRValue(cl))/255.0f);

gg=((GetGValue(cl))/255.0f);

bb=((GetBValue(cl))/255.0f);

}



* **Light Dialog Box :**
* Creating Light Sources :

glLightfv(GL\_LIGHT0,GL\_AMBIENT,ambient\_light);

glLightfv(GL\_LIGHT0,GL\_DIFFUSE,diffuse\_light);

glLightfv(GL\_LIGHT0,GL\_SPECULAR,specular\_light);

glLightfv(GL\_LIGHT0,GL\_POSITION,position\_light);

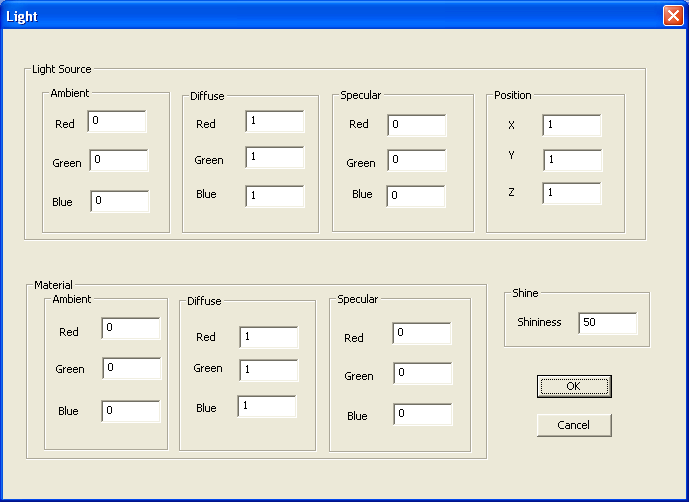
* Specifying current material property for use in lighting calculations :

glMaterialfv(GL\_FRONT,GL\_AMBIENT,ambient\_material);

glMaterialfv(GL\_FRONT,GL\_DIFFUSE,diffuse\_material);

glMaterialfv(GL\_FRONT,GL\_SPECULAR,specular\_material);

glMaterialf(GL\_FRONT,GL\_SHININESS,shininess\_material);



* **Create a Cube :**

Click on Shape > Cube

* Creating front of cube :

glBegin(GL\_POLYGON);

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

glNormal3f(-0.5f,0.5f,0.5f);

glVertex3f(-0.5f,0.5f,0.5f);

glNormal3f(-0.5f,-0.5f,0.5f);

glVertex3f(-0.5f,-0.5f,0.5f);

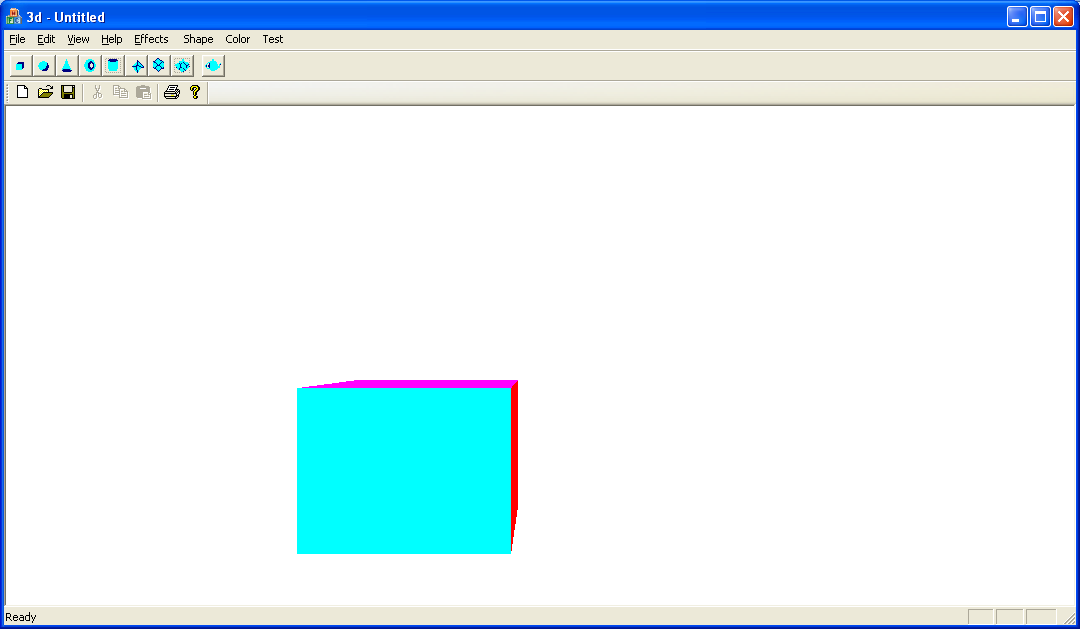
glNormal3f(0.5f,-0.5f,0.5f);

glVertex3f(0.5f,-0.5f,0.5f);

glNormal3f(0.5f,0.5f,0.5f);

glVertex3f(0.5f,0.5f,0.5f);

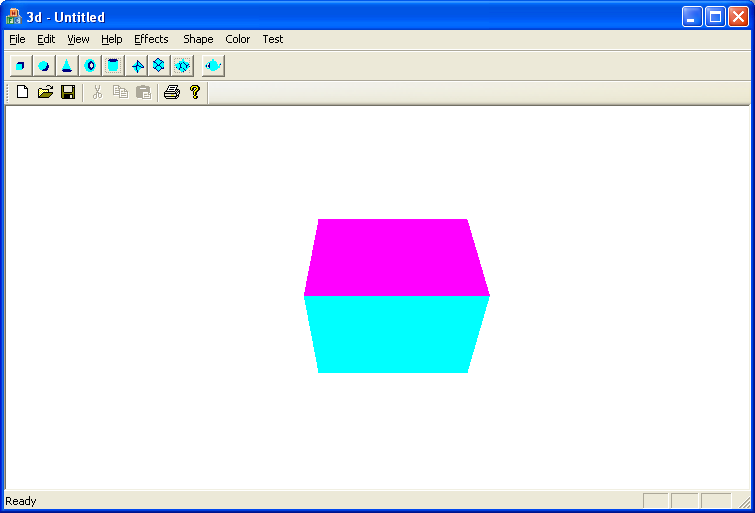
glEnd();

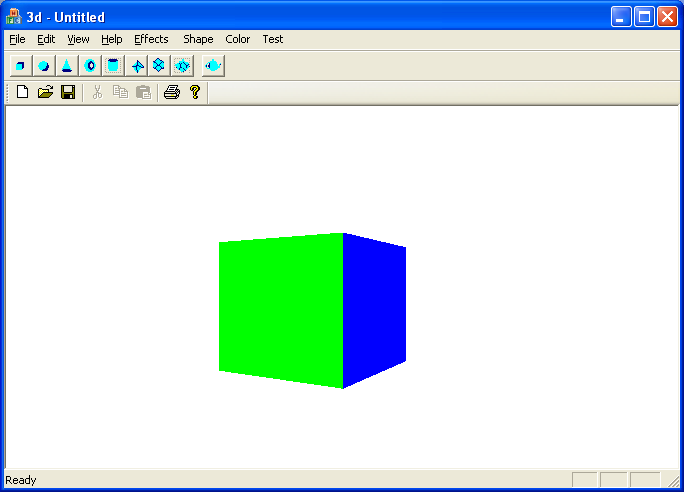


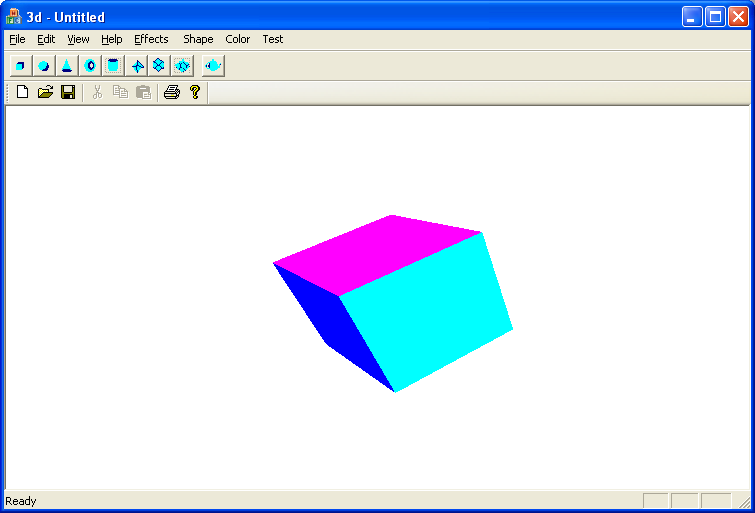
* **Rotate the Cube :**

Click on Effects > Transform

* + Rotate along X-axis
  + Rotate along Y-axis
  + Rotate along Z-axis



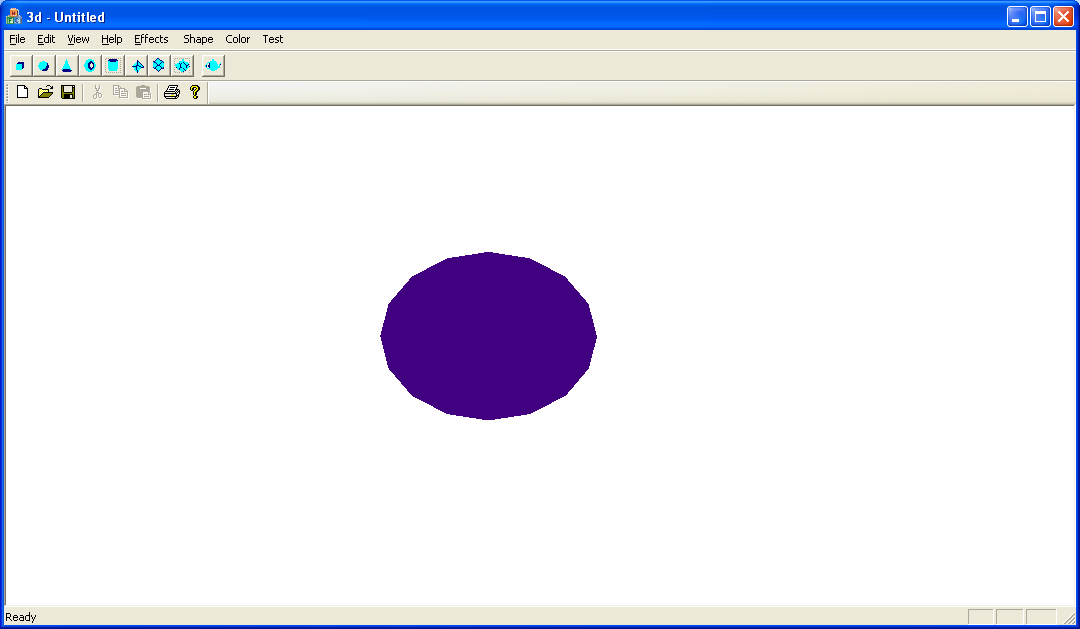




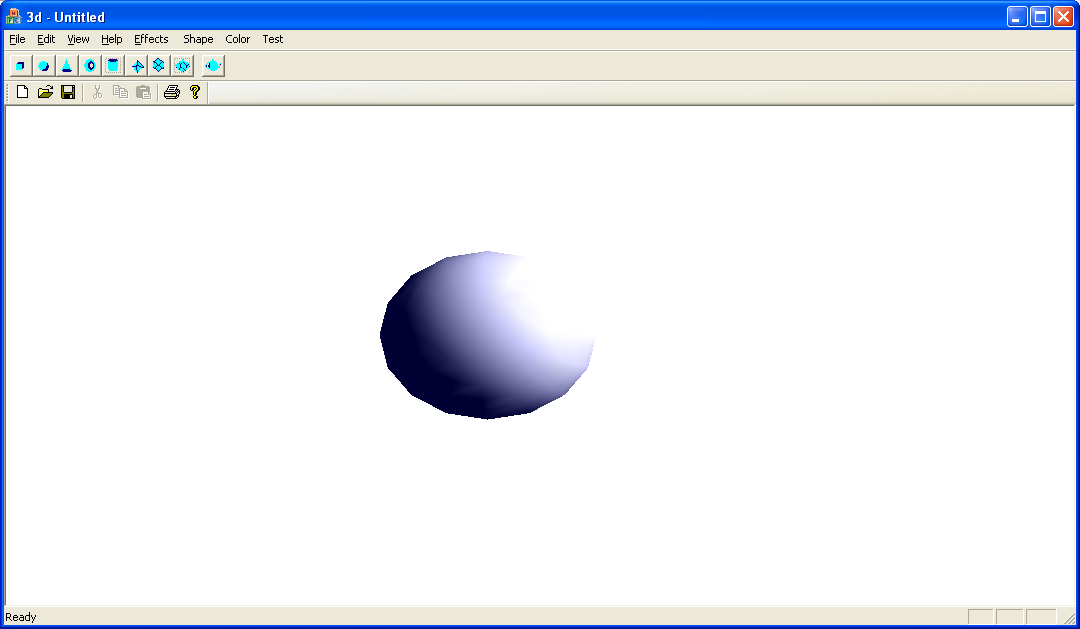
* **Create a Sphere :**

Click on Shape > Spher

auxSolidSphere(GLdouble);

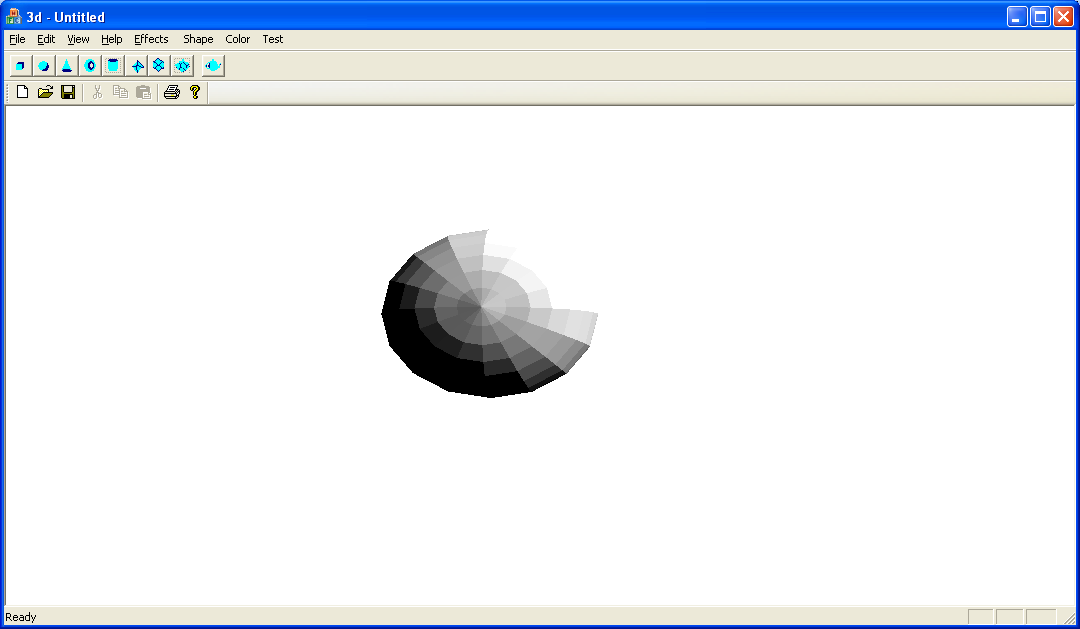


* **Lighting the Sphere**



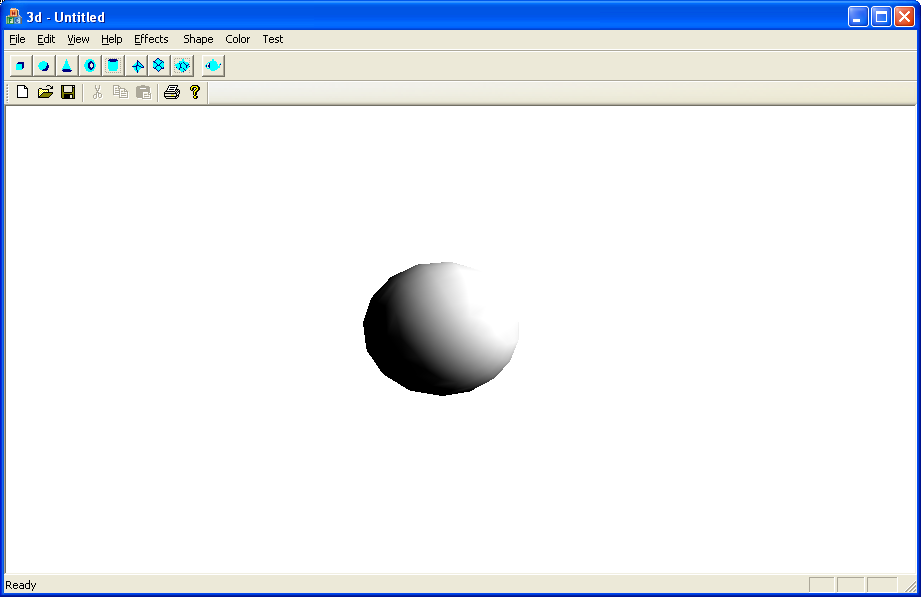
* **Change Shade of Sphere :**
* Flat shade :

glShadeModel(GL\_FLAT);



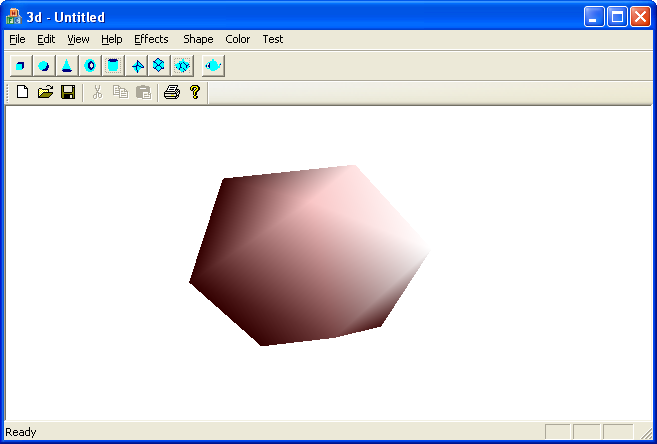
* Flat shade :

glShadeModel(GL\_SMOOTH);

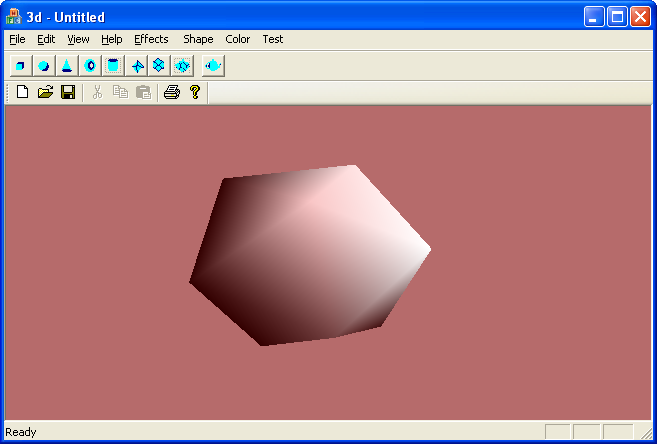


* **Create a Icosahedron :**

Click on Shape > Icosahedron

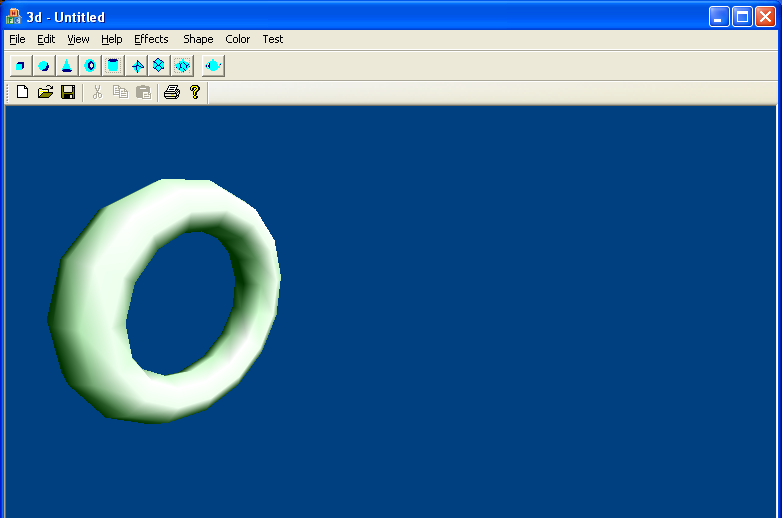


* **Change the Background Color:**

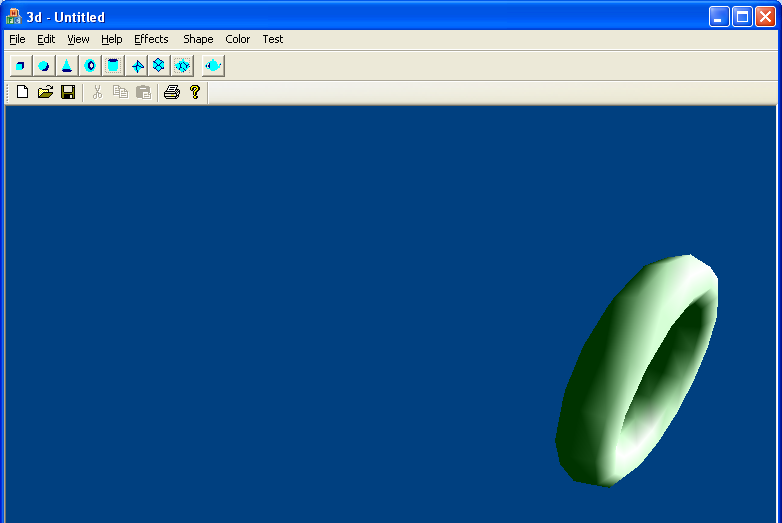


* **Create a Torus :**

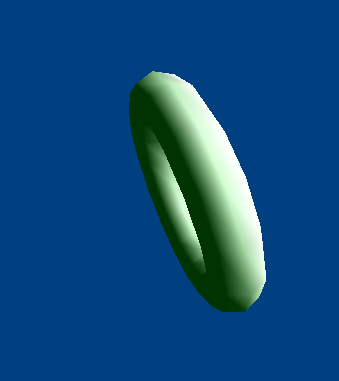
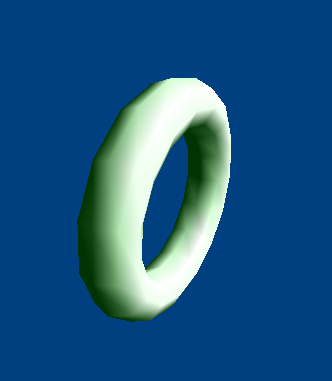
Click on Shape > Torus

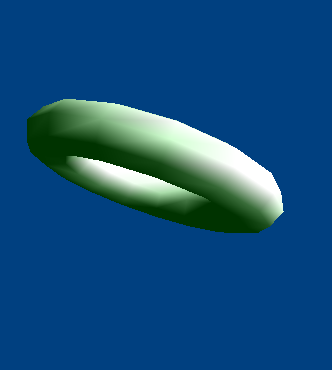


* **Move the Torus :**



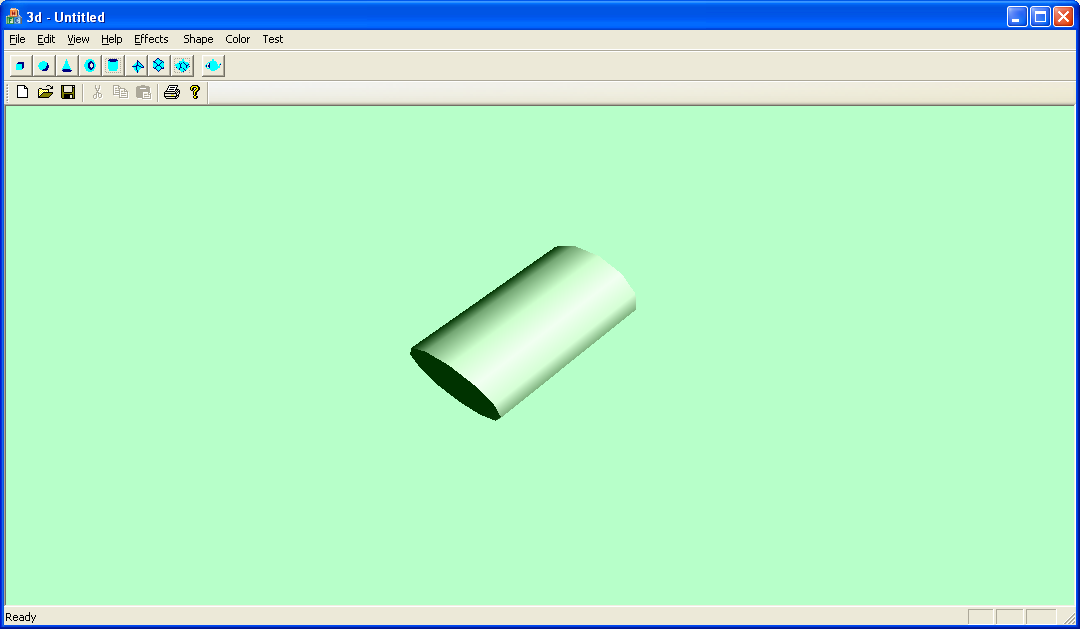
* **Rotate the Torus :**



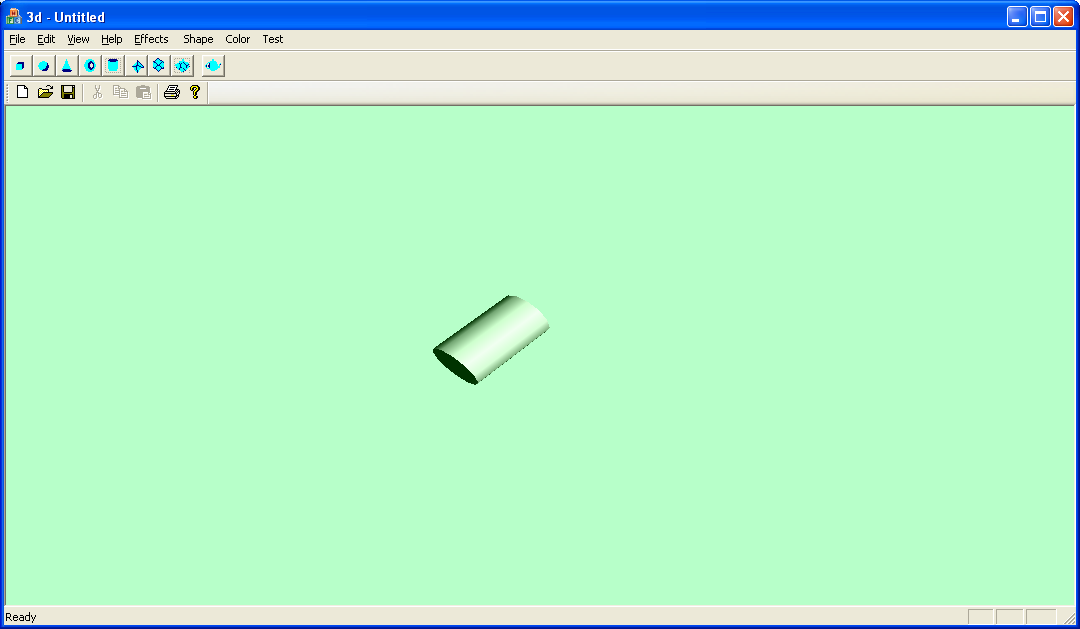


* **Create a Cyllinder :**

Click on Shape > Cyllinder



* **Change the Scale:**



* **Apply Texture Mapping on Cube :**

GLuint texName;

glPixelStorei(GL\_UNPACK\_ALIGNMENT, 1);

glGenTextures(1, &texName);

glBindTexture(GL\_TEXTURE\_2D, texName);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, 64, 64, 0, GL\_RGBA,

GL\_UNSIGNED\_BYTE, checkImage);

glEnable(GL\_TEXTURE\_2D);

glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_REPLACE);

glBindTexture(GL\_TEXTURE\_2D, texName);

glBegin(GL\_POLYGON);

glTexCoord2f(0.0f, 0.0f);

glVertex3f(-0.5f,0.5f,0.5f);

glTexCoord2f(0.0f, 1.0f);

glVertex3f(-0.5f,-0.5f,0.5f);

glTexCoord2f(1.0f, 1.0f);

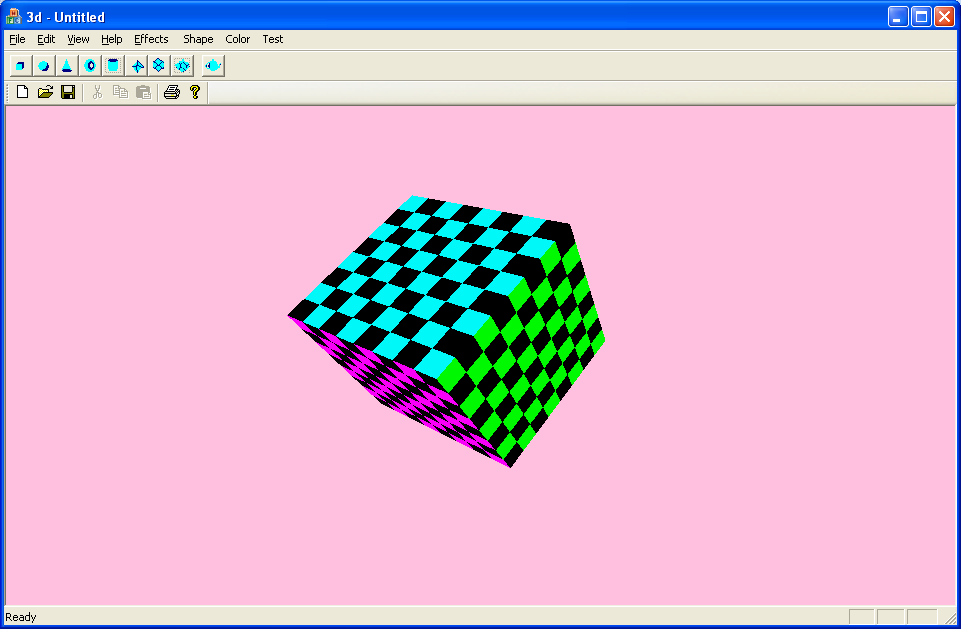
glVertex3f(0.5f,-0.5f,0.5f);

glTexCoord2f(1.0f, 0.0f);

glVertex3f(0.5f,0.5f,0.5f);

glEnd();

glDisable(GL\_TEXTURE\_2D);



1. **Future Enhancements**

The current project has been targeted for creation and manipulation of 3-D objects. The scope of the project was limited to this work. The work can be continued and many other features can be added such as:

* 1.Creation of complex 3D Objects
* 2.To view the object from different projections and different viewpoints simultaneously
* Shadow effects can be added to get a more realistic graphics display.
* Animation effects can also be added. Computer Animation generally refers to
* any time sequence of visual changes in a scene. In addition to changing object position with translations and rotations, a computer-generated animation could display time variations in object size, colour, transparency, or surface texture.
* 5.Irregular surface can be created using Bezier equations. Other equations are also available which create curves in some other way.
* 6.Interface can be made more user friendly by providing some more mouse interactions.

1. **Bibliography**
2. Visual C++ Programming: Yashvant Kanetkar.
3. VC++, COM and Beyond: Yashvant Kanetkar; Sudesh Saoji.
4. OpenGL Programming Guide: Dave Shreiner.
5. OpenGL SuperBible: Richard S. Wright; Nicholas Haemel; Graham Sellers; Benjamin Lipchak.
6. Programming Windows: Charles Petzold.
7. OpenGL and VC++ tutorials from Internet.