AMC Engineering College

NAAC Accredited with 'B+' Grade, Kalkere , Bengaluru-560083
(Affiliated to Visvesvaraya Technological University, Belagavi)

Department of Computer Science and Engineering

15CSL68
Computer Graphics & Visualization Laboratory

Table of Contents

Vision of the Institution

"To be a leader in imparting value based Technical Education and Research for the benefit of society."

Mission of the Institution

M1 To provide state of the art Infrastructure facilities.

M2 To implement modern pedagogical methods in delivering the academic programs with experienced and committed faculty.

M3 To create a vibrant ambience that promotes Learning, Research, Invention and Innovation.

M4 To undertake manpower and skill development programmes for Academic Institutions and Industries.

M5 To enhance Institute Industry Interface through Collaborative Research and Consultancy.

M6 To generate and disseminate knowledge through training programmes / workshops/ seminars/ conferences/ publications.

M7 To be a more comprehensive college in terms of the number of programs offered.

M8 To relentlessly pursue professional excellence with ethical and moral values.

Vision of the Department

“Be a premier department in the field of Computer Science & Engineering to meet the technological challenges of the society”

Mission of the Department

MD 1	To provide state of the art infrastructure facilities
MD 2	To provide exposure to the latest tools in the area of computer hardware and software
MD 3	To strive for academic excellence through research in Computer Science and Engineering with creative teaching-learning pedagogy
MD 4	To establish Industry Institute Interaction and make students ready for the Industrial environment
MD 5	To transform students into entrepreneurial, technically competent, socially responsible and ethical computer science professional

Program Educational Objectives (PEOs)

After the course completion, CSE graduates will be able to:

PEO1: Graduates possess advanced knowledge of Computer Science & Engineering and excel in leadership roles to serve the society.

PEO2: Graduates of the program will apply Computer Engineering tools in core technologies for improving knowledge in the Interdisciplinary Research and Entrepreneurs.

PEO3: Graduates adapt Value-Based Proficiency in solving Real Time problems.

Program Specific Outcomes (PSO)

PSO1: Professional Skills: Ability of applying the Computing Concepts, Data Structure, Computer Hardware, Computer Networks and Suitable Algorithm..

PSO2: Software Skills: Ability to build Software Engineering System with Development Life Cycle by using analytical knowledge in Computer Science & Engineering and applying modern methodologie

Program Outcomes (POs)

Engineering Graduates will be able to:

1.**Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2.**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3.**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4.**Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5.**Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6.**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7.**Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8.**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9.**Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

COMPUTER GRAPHICS LABORATORY WITH MINI PROJECT

[As per Choice Based Credit System (CBCS) scheme]

Course objectives:

This course will enable students to

• Demonstrate simple algorithms using OpenGL Graphics Primitives and attributes.

• Implementation of line drawing and clipping algorithms using OpenGL functions

• Design and implementation of algorithms Geometric transformations on both 2D and 3D objects.

Syllabus

Subject Code	**: 15CSL68**	**IA Marks**	**: 20**
No. of Practical Hrs. / Week	**: 01I + 02P**	**Exam Hours**	**: 03**
Total No. of Practical Hrs.	**: 4**	**Exam Marks**	**: 80**

Lab Experiments:

PART A

Design, develop, and implement the following programs using OpenGL API

1. Implement Bresenham's line drawing algorithm for all types of slope.
2. Create and rotate a triangle about the origin and a fixed point.
3. Draw a color cube and spin it using OpenGL transformation matrices.
4. Draw a color cube and allow the user to move the camera suitably to experiment with Perspective viewing.
5. Clip a lines using Cohen-Sutherland Algorithm.
6. To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the surfaces of the solid object used in the scene.
7. Design, develop and implement recursively subdivide a tetrahedron to form 3D Sierpinski Gasket. The number of recursive steps is to be specified by the user.
8. Develop a menu driven program to animate a flag using Bezier Curve algorithm
9. Develop a menu driven program to fill the polygon using scan line algorithm

Project: PART –B (MINI-PROJECT) :

Student should develop mini project on the topics mentioned below or similar applications using Open GL API. Consider all types of attributes like color, thickness, styles, font, background, speed etc., while doing mini project. (During the practical exam:
the students should demonstrate and answer Viva-Voce) Sample Topics: Simulation of concepts of OS, Data structures, algorithms etc.

Course outcomes:

The students should be able to:

• Apply the concepts of computer graphics

• Implement computer graphics applications using OpenGL

• Animate real world problems using OpenGL

Conduction of Practical Examination:

1. All laboratory experiments from part A are to be included for practical examination.
2. Mini project has to be evaluated for 30 Marks as per 6(b).
3. Report should be prepared in a standard format prescribed for project work.
4. Students are allowed to pick one experiment from the lot.
5. Strictly follow the instructions as printed on the cover page of answer script.
6. Marks distribution:
   a) Part A: Procedure + Conduction + Viva:10 + 35 +5 =50 Marks
   b) Part B: Demonstration + Report + Viva voce = 15+10+05 = 30 Marks
7. Change of experiment is allowed only once and marks allotted to the procedure part to be made zero.

Reference books:

1. Donald Hearn & Pauline Baker: Computer Graphics-OpenGL Version,3rd Edition, Pearson Education,2011
2. Edward Angel: Interactive computer graphics- A Top Down approach with OpenGL, 5 th edition. Pearson Education, 2011
3. M M Raikar, Computer Graphics using OpenGL, Fillip Learning / Elsevier, Bangalore / New Delhi (2013)

1. INTRODUCTION

OpenGL -- The Industry's Foundation for High Performance Graphics

OpenGL (**O**pen **G**raphics **L**ibrary) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D Computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives.

OpenGL provides a powerful but primitive set of rendering commands, and all higher-level drawing must be done in terms of these commands. Also, OpenGL programs have to use the underlying mechanisms of the windowing system. A number of libraries exist to allow you to simplify your programming tasks, including the following:

- The OpenGL Utility Library (GLU) contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections, performing polygon tessellation, and rendering surfaces. GLU routines use the prefix **glu**.
- For every window system, there is a library that extends the functionality of that window system to support OpenGL rendering. For machines that use the X Window System, the OpenGL Extension to the X Window System (GLX) is provided as an adjunct to OpenGL. GLX routines use the prefix **glX**. For Microsoft Windows, the WGL routines provide the Windows to OpenGL interface. All WGL routines use the prefix **wgl**. For IBM OS/2, the PGL is the Presentation Manager to OpenGL interface, and its routines use the prefix **pgl**.
- The OpenGL Utility Toolkit (GLUT) is a window system-independent toolkit, written by Mark Kilgard, to hide the complexities of differing window system APIs. Provides functionality common to all window systems
  - Open a window
  - Get input from mouse and keyboard
  - Menus
  - Event-driven

  Code is portable but GLUT lacks the functionality of a good toolkit for a specific platform
  - No slide bars
- Open Inventor is an object-oriented toolkit based on OpenGL which provides objects and methods for creating interactive three-dimensional graphics applications. Open Inventor, which is written in C++, provides prebuilt objects and a built-in event model for user interaction, high-level application components for creating and editing three-dimensional scenes, and the ability to print objects and exchange data in other graphics formats. Open Inventor is separate from OpenGL.

2. GETTING STARTED

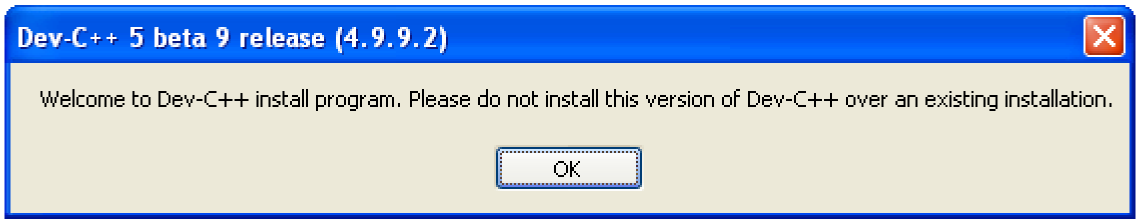
2.1 On Windows:

2.1.1 Working with Visual C++ →

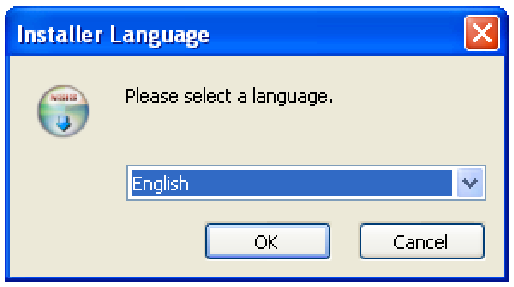
- Get glut.h, glut32.lib and glut32.dll from web.
- Copy and paste the glut.h file in the C:\ProgramFiles\MicrosoftVisual studio\VC98\Include\GL folder.
- Copy and paste the glut32.lib file in the C:\ProgramFiles\MicrosoftVisual studio\VC98\Lib folder.
- Copy and paste the glut32.dll file in the C:\WINDOWS\system32
- After following the above steps we can compile and edit a program on VC++ in the usual way by creating a win32 console application project adding a c/ c++ source file to it, editing our program and building it and later executing it.

2.1.2. Working with Dev C++ →

Find Dev C++ 's Beta version at this site>>
http://bloodshed-dev-c.en.softonic.com/
Click on the dev C++ set up icon the download will begin.
Here is the step by step guide to installation of Dev.

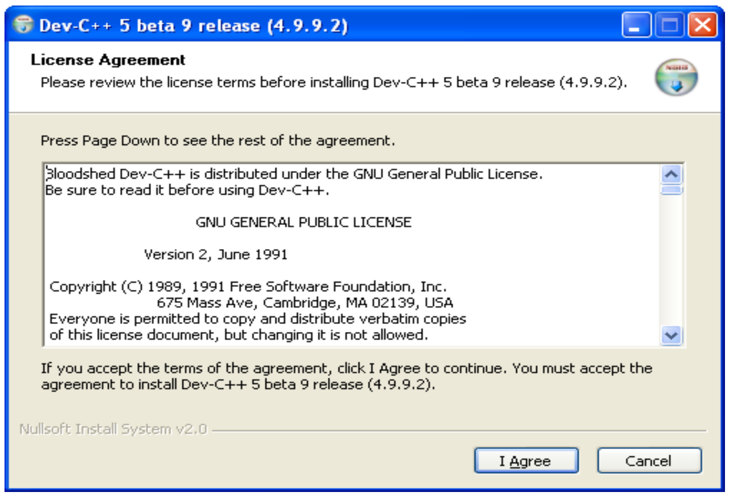

Dev-C++ 5 beta 9 release (4.9.9.2)

Welcome to Dev-C++ install program. Please do not install this version of Dev-C++ over an existing installation.


Installer Language

Please select a language.

English


Dev-C++ 5 beta 9 release (4.9.9.2)

License Agreement
Please review the license terms before installing Dev-C++ 5 beta 9 release (4.9.9.2).

Press Page Down to see the rest of the agreement.

Bloodshed Dev-C++ is distributed under the GNU General Public License.
Be sure to read it before using Dev-C++.

GNU GENERAL PUBLIC LICENSE

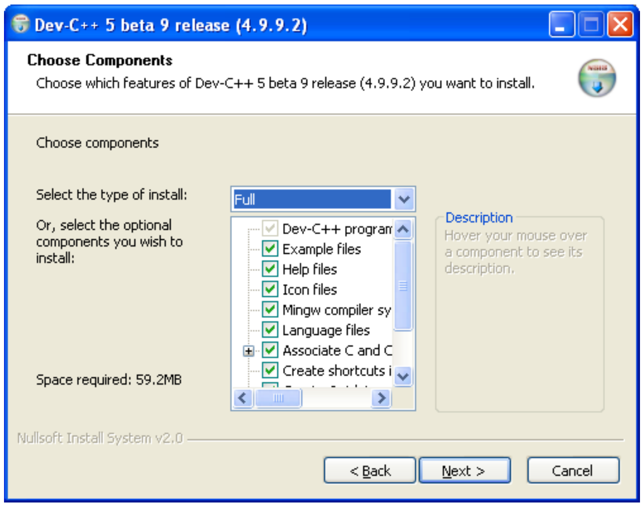
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
675 Mass Ave, Cambridge, MA 02139, USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Dev-C++ 5 beta 9 release (4.9.9.2).

Nullsoft Install System v2.0

Once clicked agree, you will prompted to choose between full or limited installs, the drive in which you would like the installation, whether it should be available for all users of your pc. And then you need to click on finish and ok.



Dev-C++ 5 beta 9 release (4.9.9.2)

Choose Components

Choose which features of Dev-C++ 5 beta 9 release (4.9.9.2) you want to install.

Choose components

Select the type of install: Full

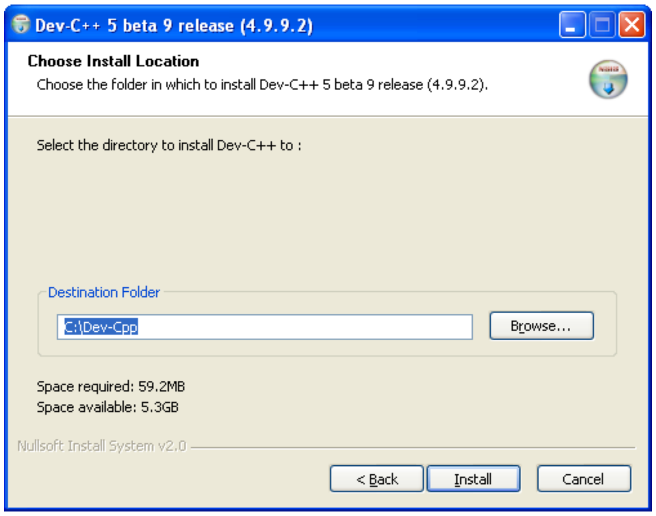
Or, select the optional components you wish to install:

- ✓ Dev-C++ program
- ✓ Example files
- ✓ Help files
- ✓ Icon files
- ✓ Mingw compiler sy
- ✓ Language files
- ✓ Associate C and C
- ✓ Create shortcuts i

Description
Hover your mouse over a component to see its description.

Space required: 59.2MB

Nullsoft Install System v2.0



Dev-C++ 5 beta 9 release (4.9.9.2)

Choose Install Location

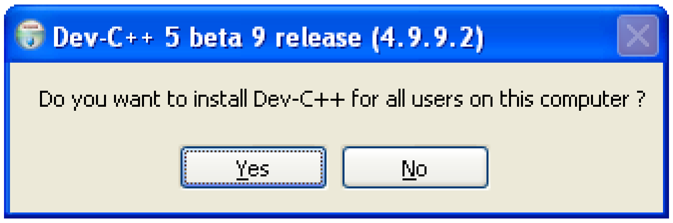
Choose the folder in which to install Dev-C++ 5 beta 9 release (4.9.9.2).

Select the directory to install Dev-C++ to :

Destination Folder: C:\Dev-Cpp

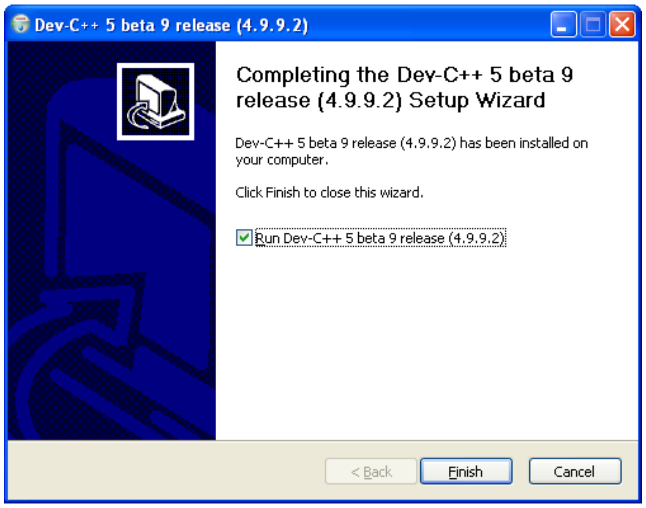
Space required: 59.2MB
Space available: 5.3GB

Nullsoft Install System v2.0



Dev-C++ 5 beta 9 release (4.9.9.2)

Do you want to install Dev-C++ for all users on this computer ?



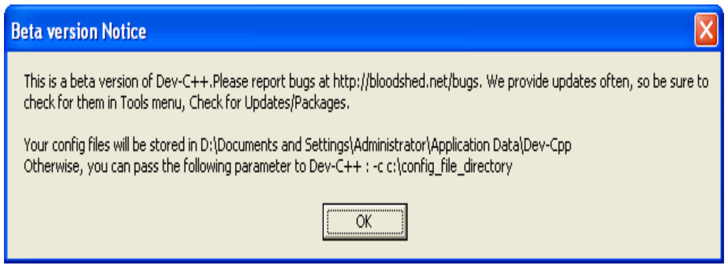
Dev-C++ 5 beta 9 release (4.9.9.2)

Completing the Dev-C++ 5 beta 9 release (4.9.9.2) Setup Wizard

Dev-C++ 5 beta 9 release (4.9.9.2) has been installed on your computer.

Click Finish to close this wizard.

✓ Run Dev-C++ 5 beta 9 release (4.9.9.2)

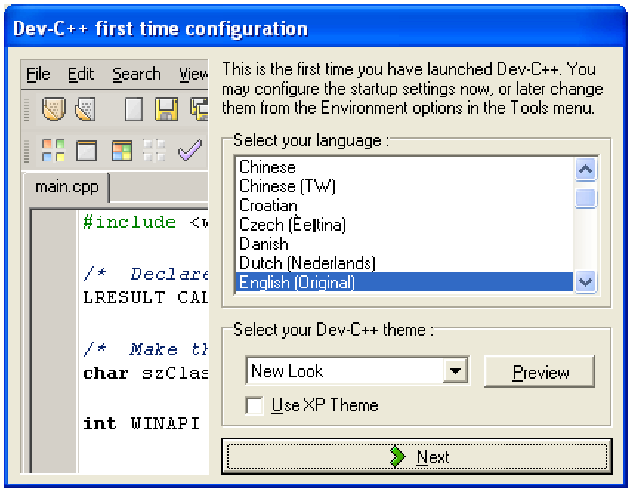


Beta version Notice

This is a beta version of Dev-C++.Please report bugs at http://bloodshed.net/bugs. We provide updates often, so be sure to check for them in Tools menu, Check for Updates/Packages.

Your config files will be stored in D:\Documents and Settings\Administrator\Application Data\Dev-Cpp
Otherwise, you can pass the following parameter to Dev-C++ : -c c:\config_file_directory

Then the following window will pop for the configuration, use English as the language and click on next. Then you will asked for enabling certain features and generation of cache, click yes and proceed by clicking next, Once configured click ok



Dev-C++ first time configuration

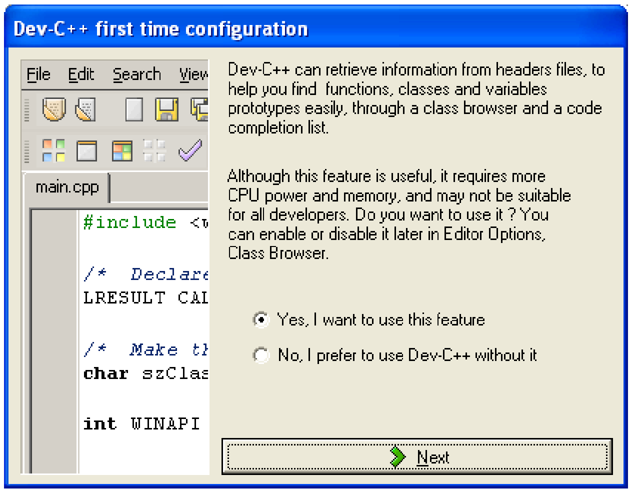
This is the first time you have launched Dev-C++. You may configure the startup settings now, or later change them from the Environment options in the Tools menu.

Select your language :

- Chinese
- Chinese (TW)
- Croatian
- Czech (Èeltina)
- Danish
- Dutch (Nederlands)
- English (Original) [✓]

Select your Dev-C++ theme : New Look

☐ Use XP Theme

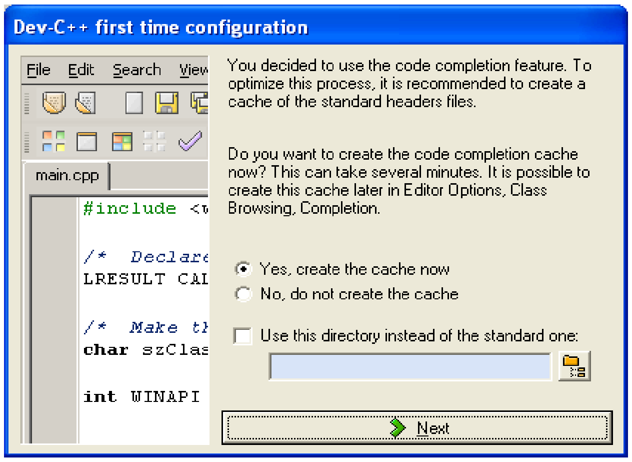


Dev-C++ first time configuration

Dev-C++ can retrieve information from headers files, to help you find functions, classes and variables prototypes easily, through a class browser and a code completion list.

Although this feature is useful, it requires more CPU power and memory, and may not be suitable for all developers. Do you want to use it ? You can enable or disable it later in Editor Options, Class Browser.

[✓ Yes, I want to use this feature] No, I prefer to use Dev-C++ without it


Dev-C++ first time configuration

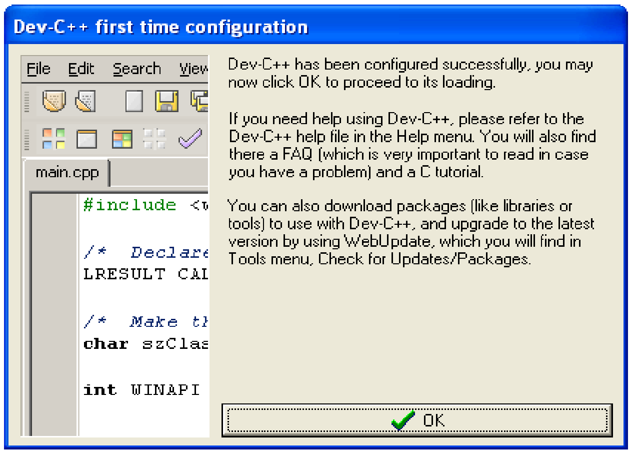
You decided to use the code completion feature. To optimize this process, it is recommended to create a cache of the standard headers files.

Do you want to create the code completion cache now? This can take several minutes. It is possible to create this cache later in Editor Options, Class Browsing, Completion.

(●) Yes, create the cache now
( ) No, do not create the cache

[ ] Use this directory instead of the standard one:

```
#include <w
/*  Declare
LRESULT CAL
/*  Make th
char szClas
int WINAPI
```


Dev-C++ first time configuration

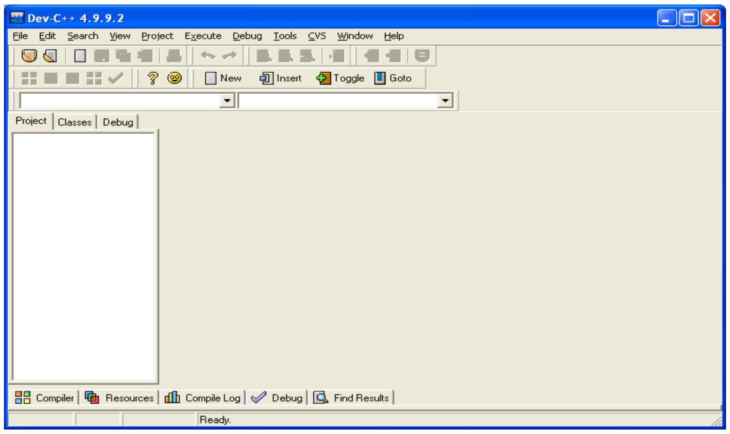
Dev-C++ has been configured successfully, you may now click OK to proceed to its loading.

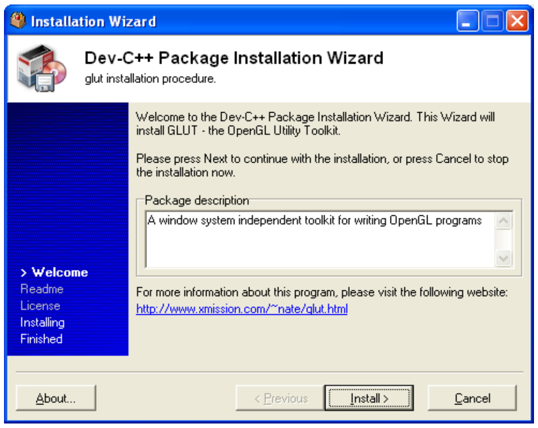
If you need help using Dev-C++, please refer to the Dev-C++ help file in the Help menu. You will also find there a FAQ (which is very important to read in case you have a problem) and a C tutorial.

You can also download packages (like libraries or tools) to use with Dev-C++, and upgrade to the latest version by using WebUpdate, which you will find in Tools menu, Check for Updates/Packages.

```
#include <w
/*  Declare
LRESULT CAL
/*  Make th
char szClas
int WINAPI
```

Now Dev C++ is ready to use, but can we still compile a glut program here??
The answer is NO!
Now we need to install the glut pack!




Installation Wizard

Dev-C++ Package Installation Wizard
glut installation procedure.

Welcome to the Dev-C++ Package Installation Wizard. This Wizard will install GLUT - the OpenGL Utility Toolkit.

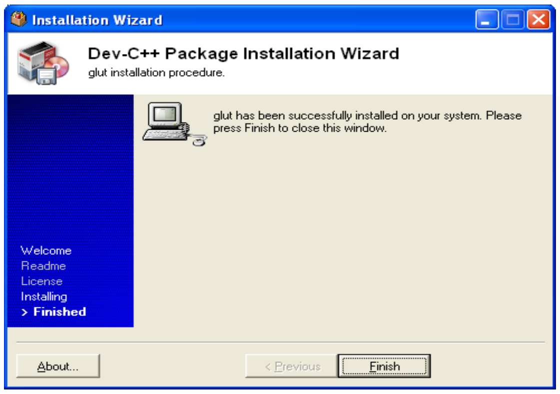
Please press Next to continue with the installation, or press Cancel to stop the installation now.

Package description
A window system independent toolkit for writing OpenGL programs

For more information about this program, please visit the following website:
http://www.xmission.com/~nate/glut.html

Find the DevPak downloads for Dev C++ here>>
You can download any version of glut Devpaks from here:
http://www.nigels.com/glt/devpak/

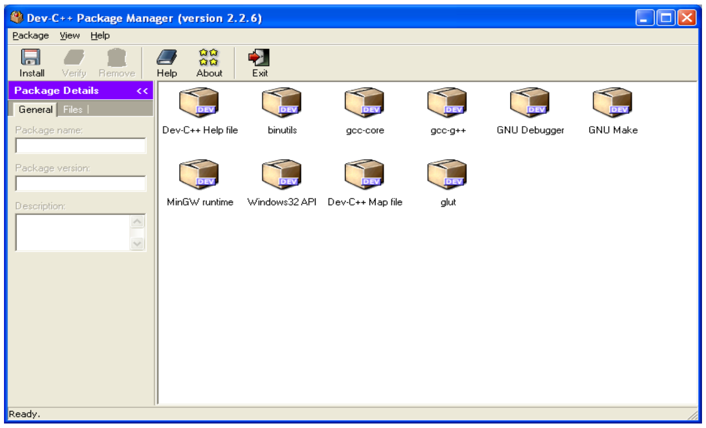
Save the package on your drive, to install double-click on the package a window like the one on right>> will pop up click install and glut will be installed for use .


Installation Wizard

Dev-C++ Package Installation Wizard
glut installation procedure.

glut has been successfully installed on your system. Please press Finish to close this window.

Once the download completes click finish.
The window shown below will pop, showing the installation of the new package.



or

Open Dev C++ and go to "Tools > Check for Updates/Packages". This should bring you to the update manager.

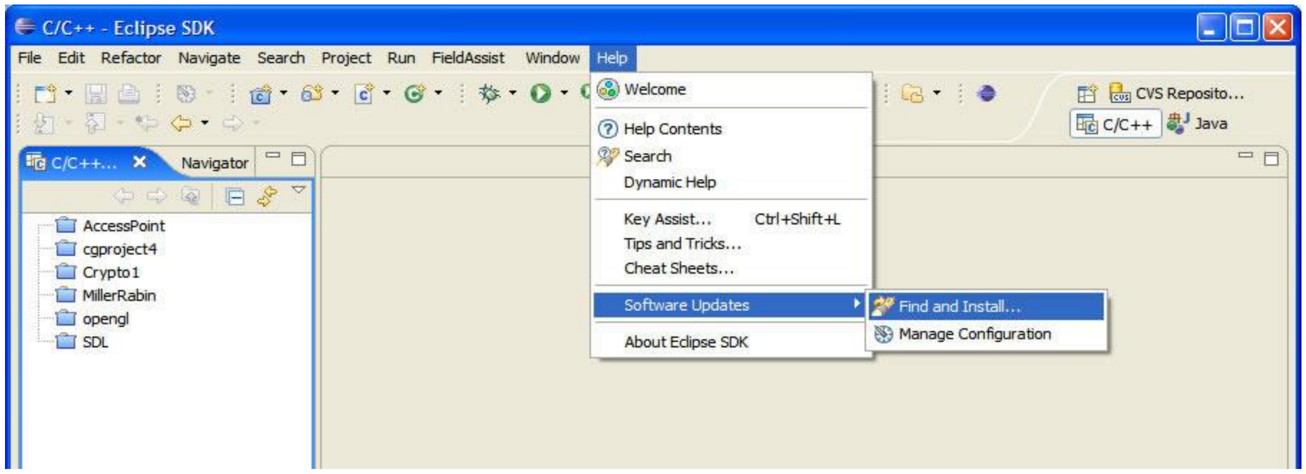
Click below the "**Select devpack server**" and select "devpacks.org community **devpacks**".

Now click "**check for updates**" at the bottom of the page. Once its done loading, scroll through the lists of packages until you find a glut package. Once you are ready, check the small box to the left of the name. Download it, and then the computer will install it automatically. Now you have opengl (opengl comes w/dev c++) and glut.

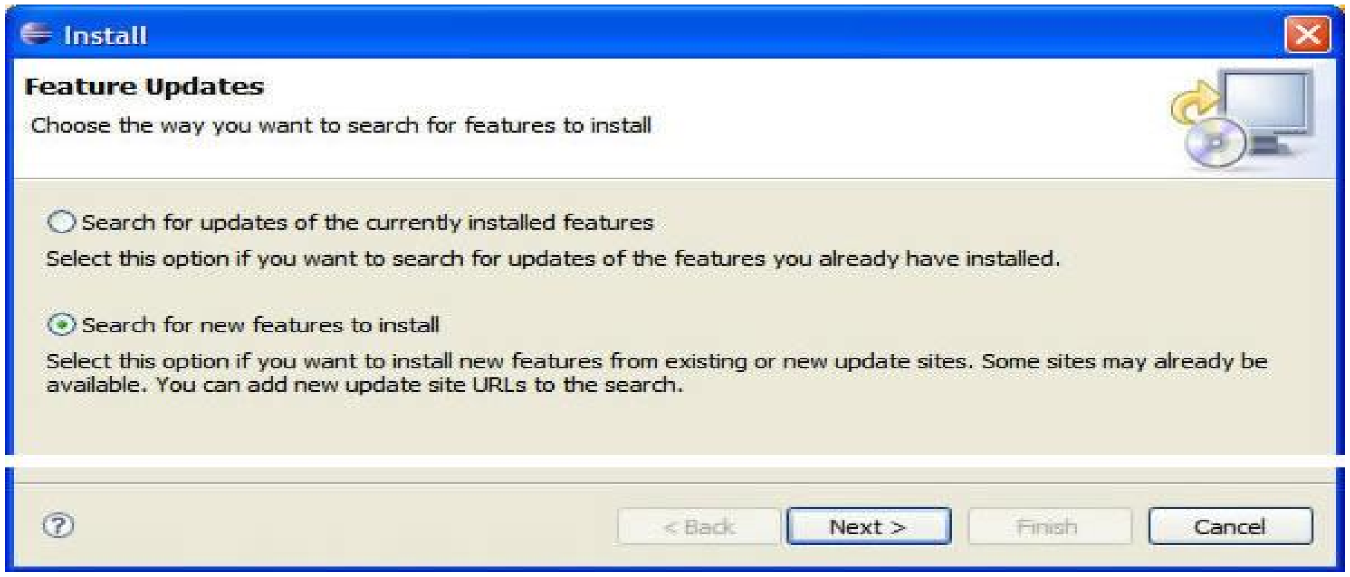
Once you download glut, dev c++ it even does you a favor by creating a template for glut programs (isnt that cool!). To use the template, click the "**multimedia**" tab in the "create a project" window. The **glut template** should be there for you to use. Clear the editor to edit your own program and compile and run/execute it.
Note: Since you select the multimedia template, your project may not be provided with a console when you run it. so before compiling or running it, Hold alt+p keys together you will get the project options window select here console so next time you run your program you will also receive the console, this is quiet helpful when your program takes input through console.

2.1.3. Working with Eclipse CDT→
1. Download Eclipse SDK 3.2.2 or newer at Eclipse.org/downloads
2. Install MinGW and the gcc/g++ compiler
Eclipse CDT does not come with a compiler so you will have to install one. The easiest way is to use MinGW (Minimalist GNU for Windows) and the gcc/g++ compiler. The compiler comes with the necessary header and library files for programming with OpenGL, but you will need to add several GLUT files in step 4.
3. Install the C/C++ Development Tooling (CDT)
CDT is an environment which allows you to develop in C/C++ using Eclipse. It's relatively easy to download the plug-in from inside Eclipse.

a. Go to Help -> Software Updates -> Find and Insta



b. Choose "Search for New Features to Install" and click on both "Callisto Discovery Site" and "The Eclipse Project Updates"



Install

Feature Updates

Choose the way you want to search for features to install

( ) Search for updates of the currently installed features

Select this option if you want to search for updates of the features you already have installed.

(•) Search for new features to install

Select this option if you want to install new features from existing or new update sites. Some sites may already be available. You can add new update site URLs to the search.

c. Choose an "Update Site Mirror" and install any updates for Eclipse and then install "C and C++ Development" (CDT) from the Callisto Discovery site.

4. Download and setup GLUT

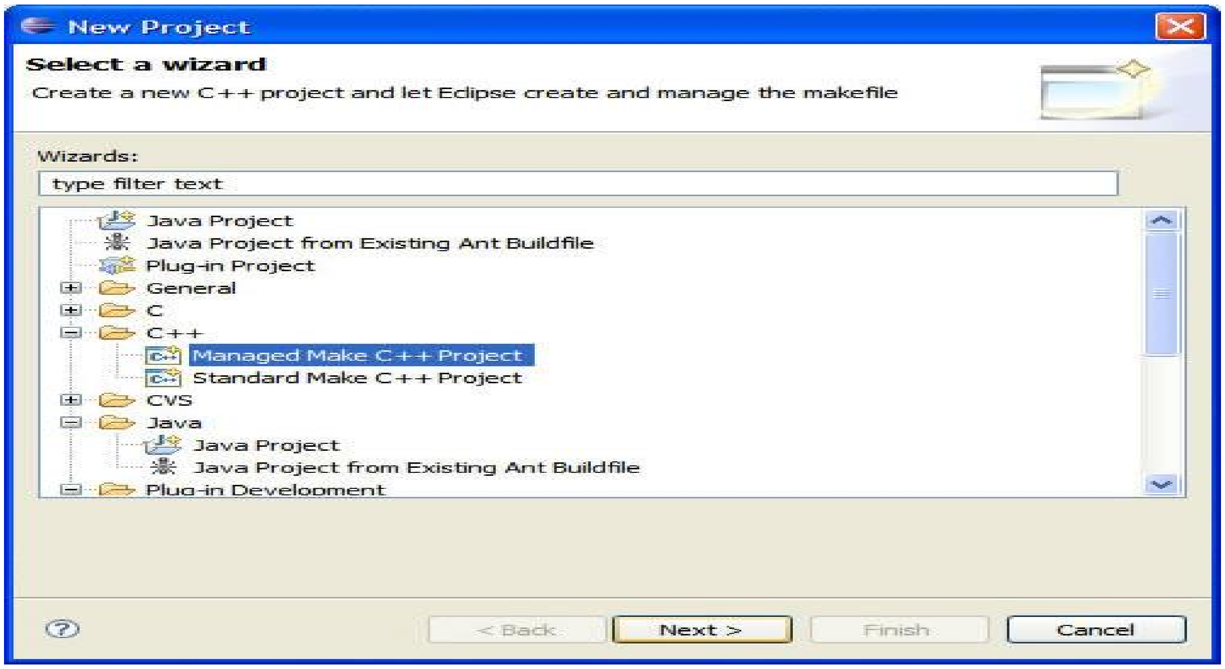
a. Download GLUT MinGW

b. Place glut32.dll in your C:\Windows\System32 folder

c. Put glut.h in the folder C:\MinGW\include\GL folder and libglut32.a in C:\MinGW\lib (These folders are relative to where MinGW was installed).

5. Start a new C/C++ Project in Eclipse

a. Go to File -> New -> Project, and choose "Managed Make C++ Project" from the wizard.


New Project

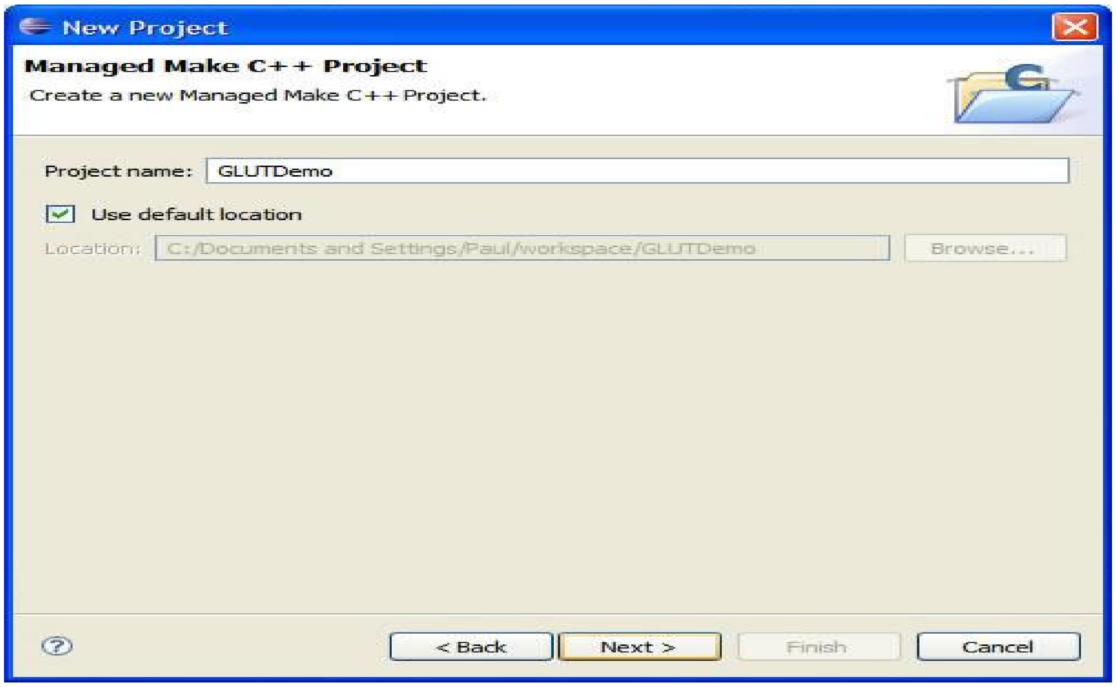
Select a wizard

Create a new C++ project and let Eclipse create and manage the makefile

Wizards:

type filter text

b. Name the project and press Next -> Finish.


New Project

Managed Make C++ Project

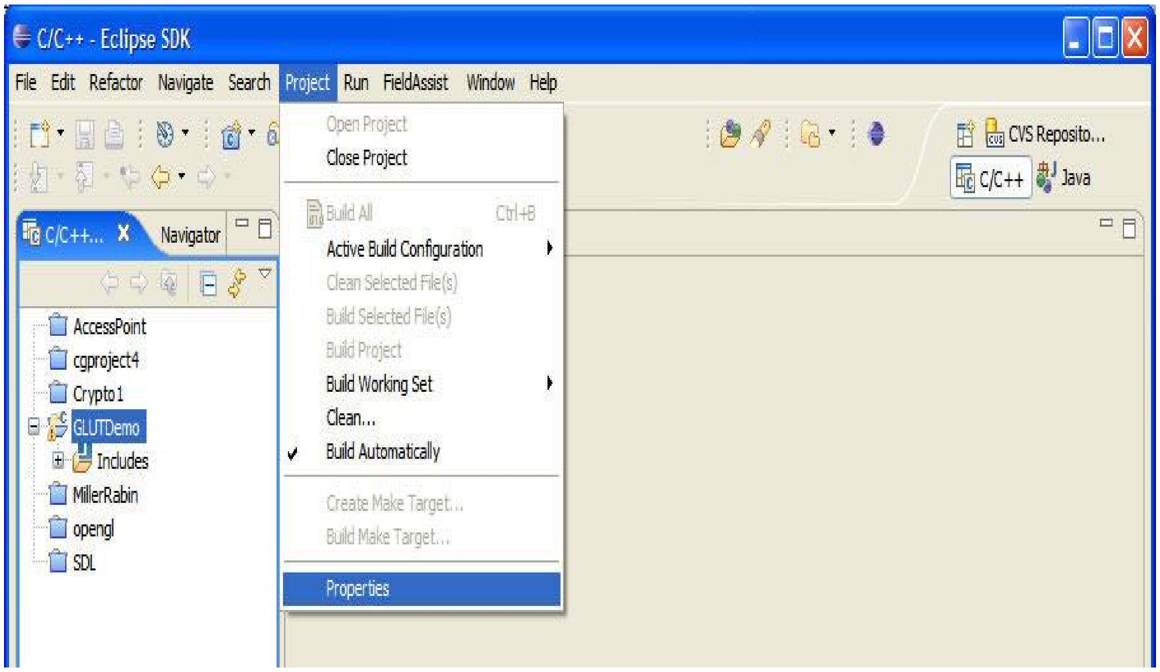
Create a new Managed Make C++ Project.

Project name: GLUTDemo

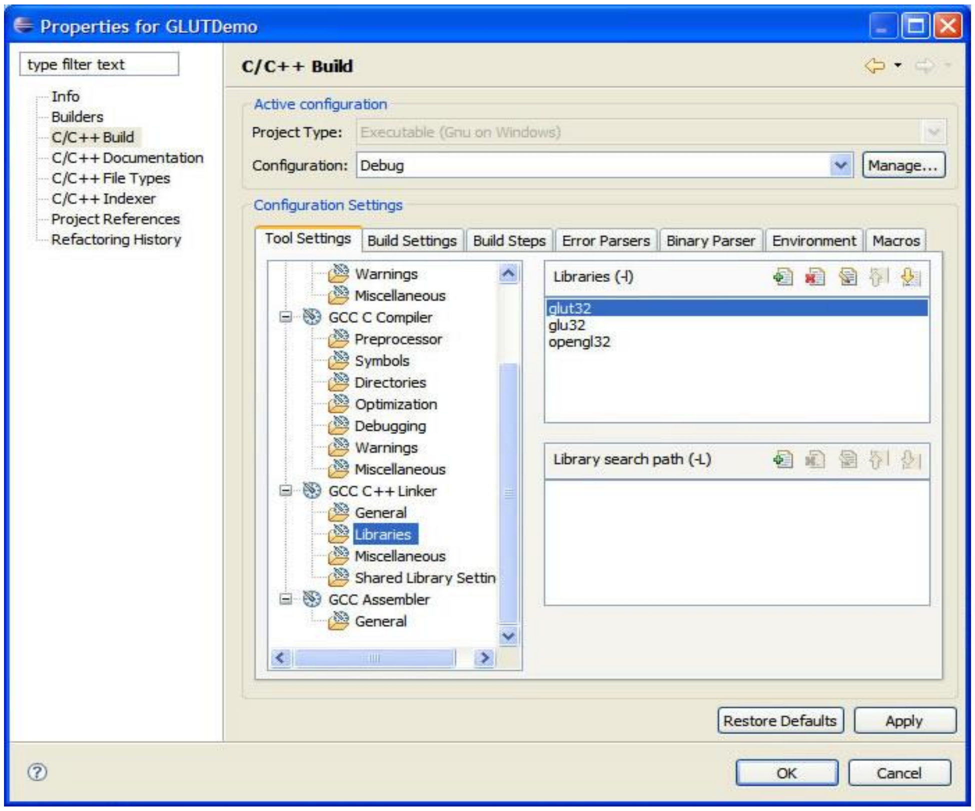
Use default location [✓]

Location: C:/Documents and Settings/Paul/workspace/GLUTDemo

c. After you finish creating the project you need to change the project settings to include the OpenGL and GLUT libraries. Highlight the new project and go to Project -> Properties.



d. Choose "C/C++ Build" and select "Libraries" from under the "GCC C++ Linker" branch. You need to add glut32, glu32, and opengl32 to the list of libraries.



Project Type: Executable (Gnu on Windows)
Configuration: Debug

Libraries (-l)
glut32
glu32
opengl32

6. Create a simple GLUT C++ program

a. Right click on the project name -> New -> Source File.

b. Edit a glut program

NOTE: When you create programs using GLUT, you need to include windows.h and any standard library header before the glut.h header file.

c. Build the program by clicking at project->build. Run the program by right clicking on the Project Name -> Run As -> Run As Local C/C++ Application

d. Now you have a basic GLUT OpenGL window which can be used to create 2D/3D applications.

2.2 On Red hat and Core Linux

Glut is included in full installs of Redhat Linux 8 and 9 and Core 1 and 2. It is located in the standard search paths. If your Redhat or Core Linux installation did not end up with a working glut or your glut does not get on with ODE or the like, obtain the zipped file for linux, This folder contains an rpm file, glut.h file and an example program cube.cpp and its makefile. Install the rpm file:

```
rpm -i glut-3.7-8.i386.rpm
```

Note:

1.If you already have a later glut, use rpm -i --force glut-3.7-8.i386.rpm.Also in case it shows errors in installing due to dependencies use --nodeps like, rpm -i glut-3.7-8.i386.rpm –nodeps When rpm is searching for dependencies it only looks in the rpm database, It does not go looking through the file tree to see if the files are actually there. A file is registered in the rpm data base when the rpm package containing that file is installed. So it is possible for a file to exist but not be registered in the rpm data base if it was installed by any method other than rpm. What you do in this situation is to check to see if the missing dependencies really exist. If they do then you use the rpm --nodeps parameter to tell rpm to install the package anyway in spite of whatever dependencies that rpm thinks are missing.

2 .Also during the install one should be logged in as the root user.

3 .Also if there is error like: can't create transaction lock along with cannot open /var/lib/rpm/__db00 or any other __db**. You can resolve this by changing directory to /var/lib/rpm and delete the db files mentioned in the error. this should now run the command normally.

Move the file "glut.h" to "/usr/include/GL":

```
mv glut.h /usr/include/GL
```

Change directories to "/usr/lib"

```
cd /usr/lib
```

And, finally, if libglut.so is not the same file as libglut.so.3.7, copy libglut "libglut.so.3.7" to "libglut.so"

```
cp libglut.so.3.7 libglut.so
```

At this point,running the Makefile that was included should create the executable "cubes", which can be run and should produce a spinning 3d cube.(note that to run make file, type "make" at command prompt, you being in root directory and run it by typing ./cubes on the prompt)
After the installation is being done as instructed above, lets move on to compiling and running a glut program:

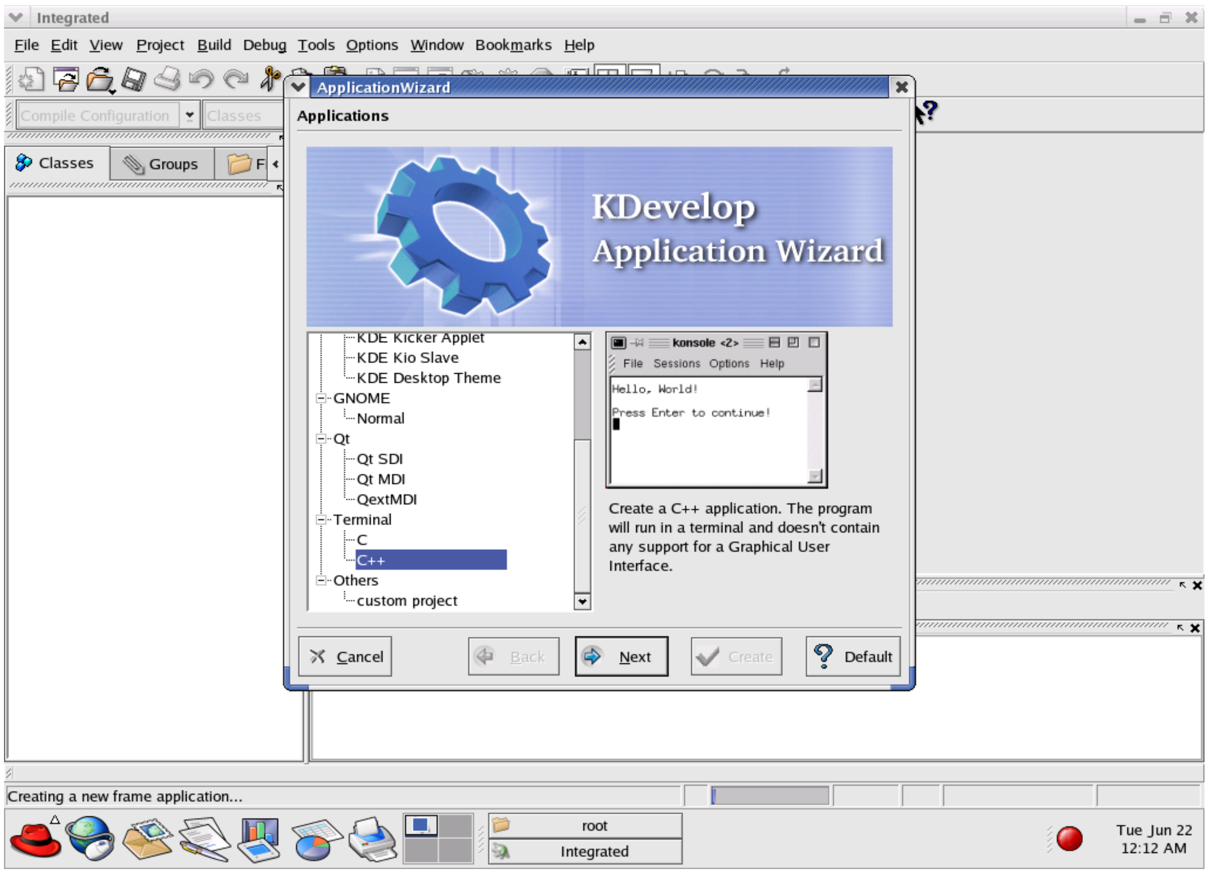
2.2.1: Running a program on the command prompt->

1. Edit the program in any editor
2. At the prompt compile using gcc compiler
   eg: # gcc myprog.cpp –lGL -lGLU –lglut
3. If program contains no errors, prompt is returned immediately after the last command, If it contains errors those will be displayed. Edit and remove the errors.
4. After compilation is successful, i.e., no errors, Then run the ./a.out i.e, type it at the prompt you will now have the output of your program on a separate window.

2.2.2: Running through an IDE->

To run through an IDE like Kdevelop on any flavor of the Linux operating system, again involves creating a project and changing its configuration settings to include the glut, gl, glu library files. and editing and running as we did with other IDEs on windows, we successfully run it here too!

1. For the Kdevelop IDE, On the menu bar, Click on project→ terminal→ c++

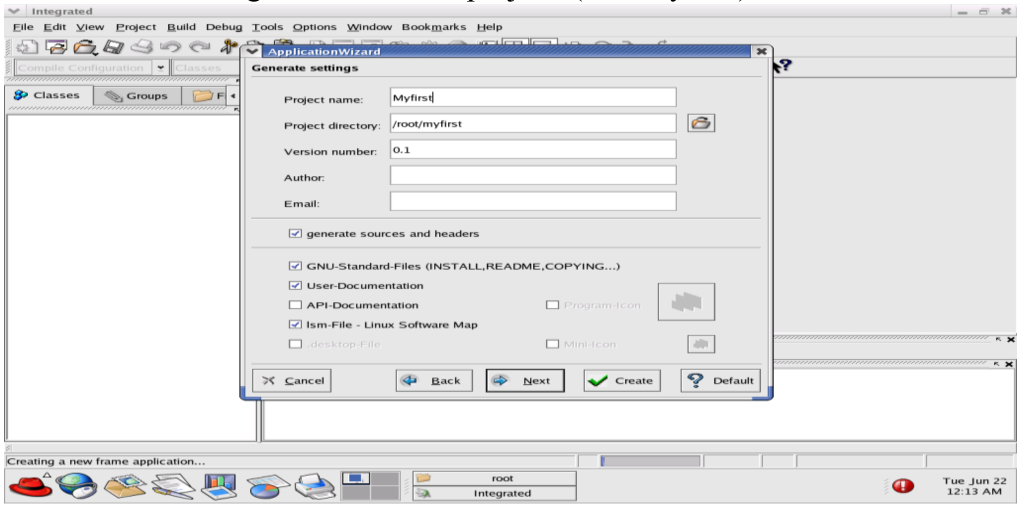


```
Hello, World!
Press Enter to continue!
```

Create a C++ application. The program will run in a terminal and doesn't contain any support for a Graphical User Interface.

Creating a new frame application...

2. Click next and give name to the project. (here myfirst).



Generate settings

Project name: Myfirst
Project directory: /root/myfirst
Version number: 0.1
Author:
Email:

- [✓] generate sources and headers
- [✓] GNU-Standard-Files (INSTALL,README,COPYING...)
- [✓] User-Documentation
- [ ] API-Documentation
- [ ] Program-Icon
- [✓] lsm-File - Linux Software Map
- [ ] .desktop-File
- [ ] Mini-Icon

Creating a new frame application...

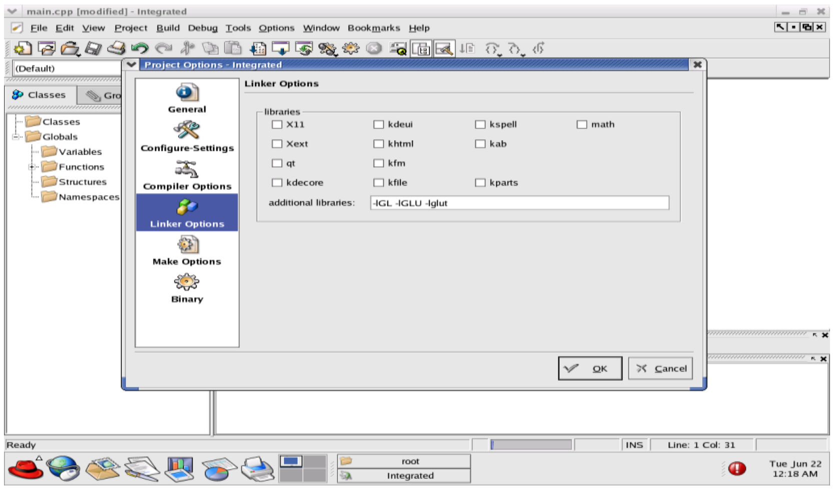
3. Go on clicking next (thrice) and finally click create. click ok on the pop up which says about user documentation, wait till you get “READY” in the processes box , then close this window. you will get the below window.
4. Clear the editor window to edit your own program

Go to project→

5. Options

6 .In the options window click on linker options

7 .In linker options, in additional libraries field type –lGL –lGLU –lglut, click ok



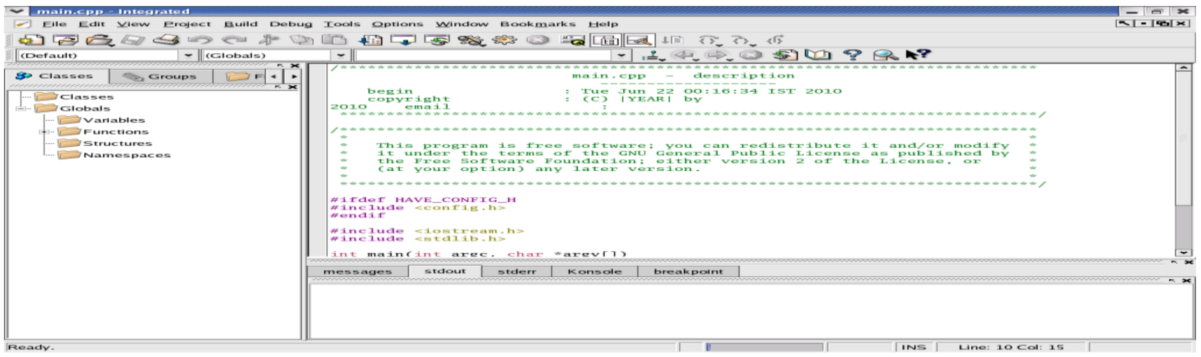
Linker Options

libraries

[ ] X11	[ ] kdeui	[ ] kspell	[ ] math
[ ] Xext	[ ] khtml	[ ] kab	
[ ] qt	[ ] kfm		
[ ] kdecore	[ ] kfile	[ ] kparts	

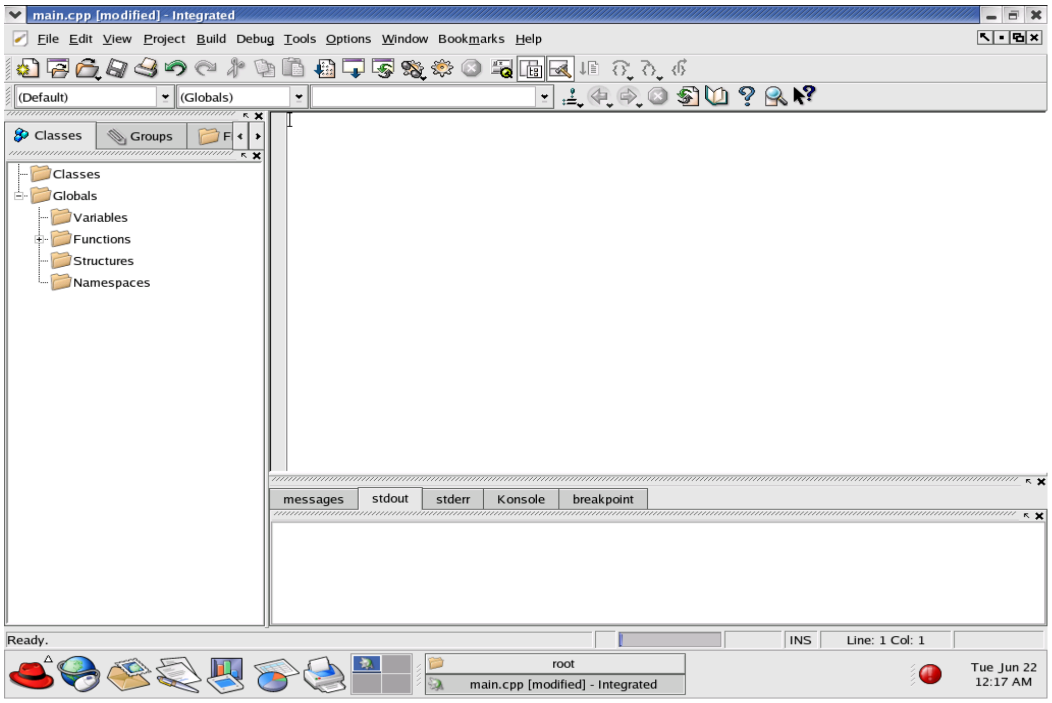
additional libraries: -lGL -lGLU -lglut

Ready | INS | Line: 1 Col: 31

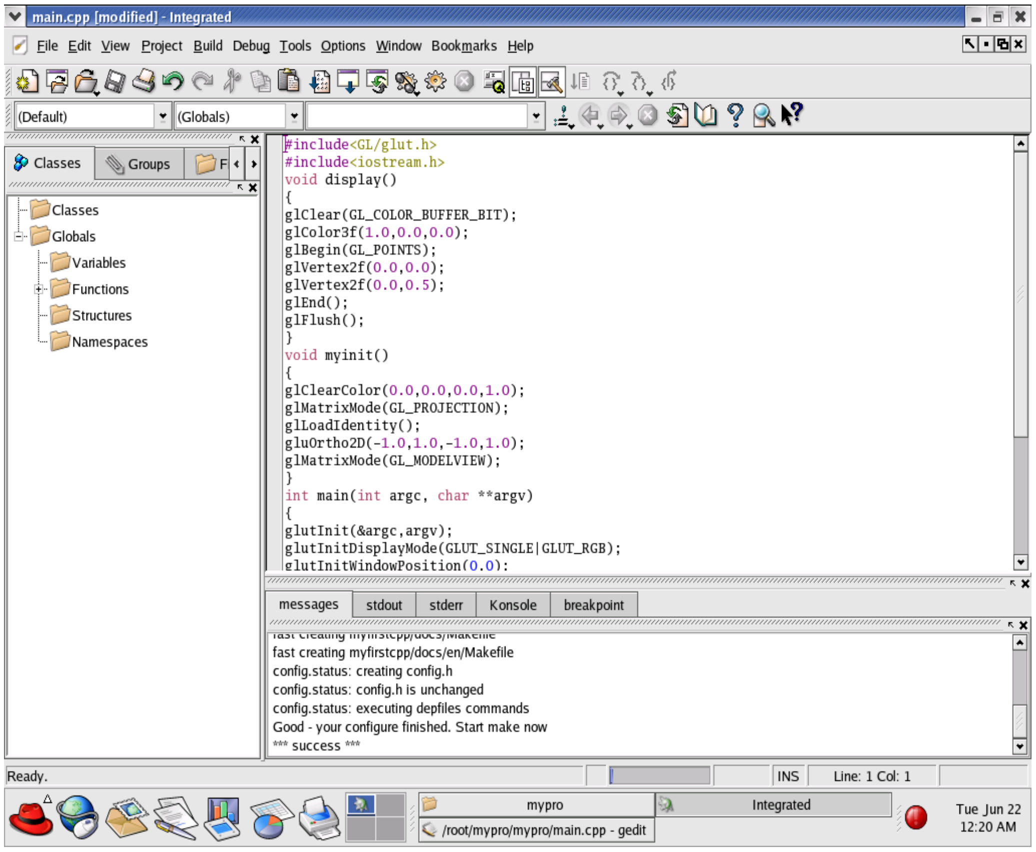


```
/***************************************************************************
                          main.cpp  -  description
                             -------------------
    begin                : Tue Jun 22 00:16:34 IST 2010
    copyright            : (C) |YEAR| by
2010    email                :
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <iostream.h>
#include <stdlib.h>

int main(int argc, char *argv[])
```

Ready. | INS | Line: 10 Col: 15

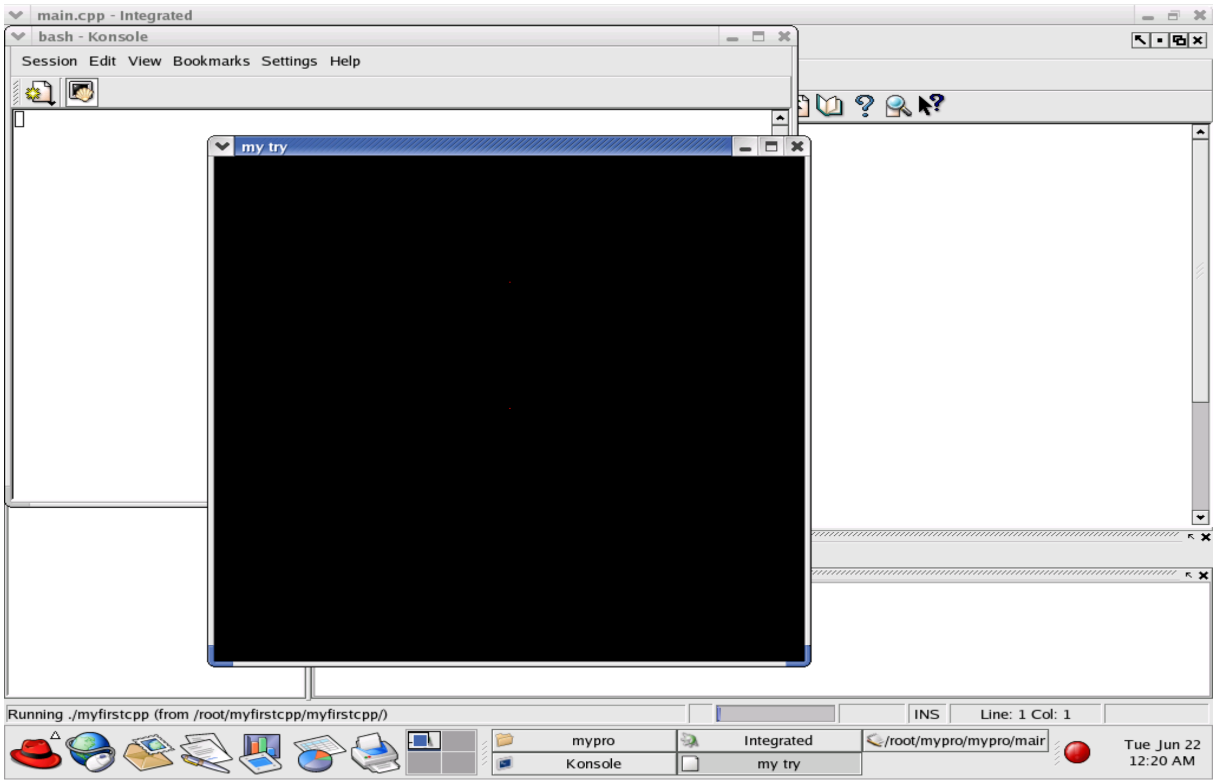


8 .Build the program, click on build→ compile file, if no errors exist and the build is successful, click
build→execute.



```
#include<GL/glut.h>
#include<iostream.h>
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0,0.0,0.0);
glBegin(GL_POINTS);
glVertex2f(0.0,0.0);
glVertex2f(0.0,0.5);
glEnd();
glFlush();
}
void myinit()
{
glClearColor(0.0,0.0,0.0,1.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(-1.0,1.0,-1.0,1.0);
glMatrixMode(GL_MODELVIEW);
}
int main(int argc, char **argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowPosition(0,0);
```

```
fast creating myfirstcpp/docs/en/Makefile
config.status: creating config.h
config.status: config.h is unchanged
config.status: executing depfiles commands
Good - your configure finished. Start make now
*** success ***
```

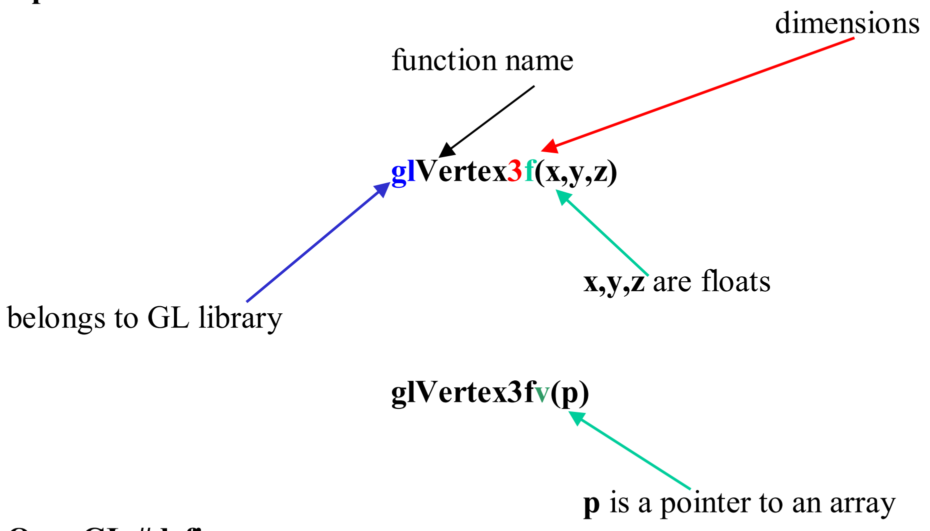


Running ./myfirstcpp (from /root/myfirstcpp/myfirstcpp/)

SO NOW LETS BEGIN LEARNING AND EXPLORING
THE MAGICAL WORLD
OF
COMPUTER GRAPHICS & VISUALIZATION!

3.INTRODUCTORY CONCEPTS

Basics1: Drawing points
OpenGL function format:



OpenGL #defines:

- Most constants are defined in the include files **gl.h**, **glu.h** and **glut.h**
  Note **#include <GL/glut.h>** should automatically include the others
  Examples
  glBegin(GL_POLYGON)
  glClear(GL_COLOR_BUFFER_BIT)
- include files also define OpenGL data types: **GLfloat**, **GLdouble**,….

Program Structure

- Most OpenGL programs have a similar structure that consists of the following functions
  main():
  - defines the callback functions
  - opens one or more windows with the required properties
  - enters event loop (last executable statement)

  init(): sets the state variables
  - Viewing
  - Attributes

  callbacks:
  - Display function
  - Input and window functions

```
#include<GL/glut.h>
#include<stdio.h>
void display()
{
```

includes **gl.h**

```
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0,0.0,0.0);        /*←fill/ draw in red*/
glPointSize(2.0);              /*← set size of the point*/
glBegin(GL_POINTS);            /*← type of object*/
glVertex2f(0.0,0.0);           /*←location of vertex*/
glVertex2f(0.0,0.5);
glEnd();                       /*←end of object definition*/
glFlush();
}
void myinit()                  opaque window
{
glClearColor(0.0,0.0,0.0,1.0);←——————————————  black clear color
gluOrtho2D(-1.0,1.0,-1.0,1.0); ←———————————————viewing volume
}
void  main(int argc, char **argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
glutInitWindowSize(500,500); ←———————————————define window properties
glutInitWindowPosition(0,0);
glutCreateWindow("Simple demo");
myinit(); ←———————————————————————  set OpenGL state
glutDisplayFunc(display);←——————————————  display callback

glutMainLoop();←——————————————————  enter event loop

}
```

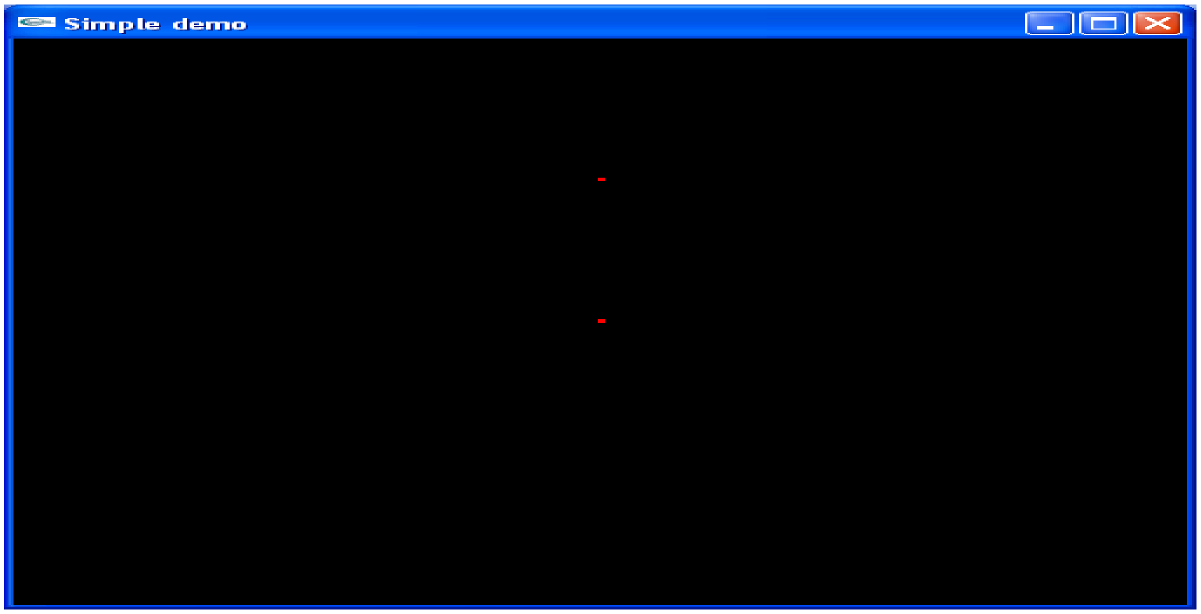
* Note that the program defines a *display callback* function named **display**
Every glut program must have a display callback
The display callback is executed whenever OpenGL decides the display must be refreshed, for example when the window is opened
The **main** function ends with the program entering an event loop

GLUT functions

- **glutInit** allows application to get command line arguments and initializes system
- **gluInitDisplayMode** requests properties for the window (the *rendering context*)
  - RGB color
  - Single buffering
  - Properties logically ORed together

- **glutWindowSize** in pixels
- **glutWindowPosition** from top-left corner of display
- **glutCreateWindow** create window with title “simple demo”
- **glutDisplayFunc** display callback
- **glutMainLoop** enter infinite event loop

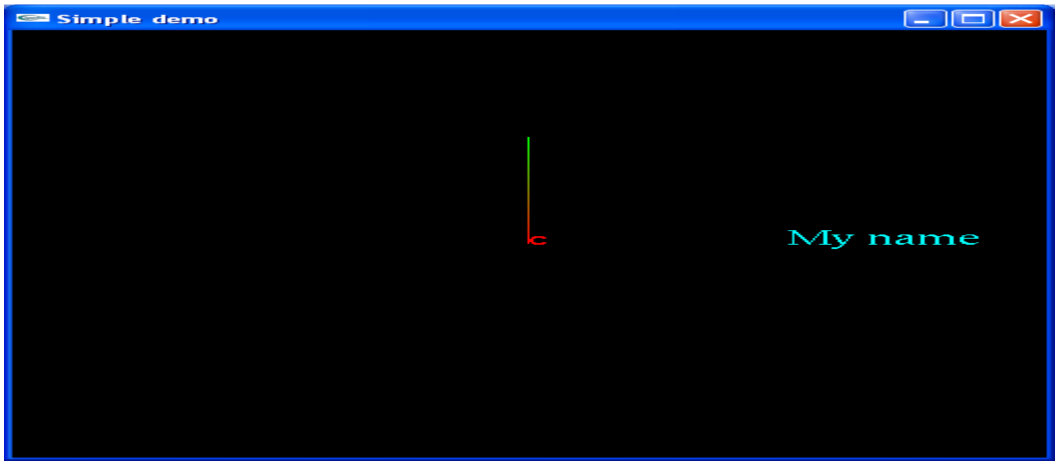
Output:



Basics 2: **Drawing lines, displaying text part of picture.**

```
#include<GL/glut.h>
#include<stdio.h>
#include<string.h>
char *str= "My name";
void display()
{
int i;
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0,0.0,0.0);
glBegin(GL_LINES);
glVertex2f(0.0,0.0);
glColor3f(0.0,1.0,0.0);
glVertex2f(0.0,0.5);
glEnd();
glColor3f(1.0,0.0,0.0);                    position of the character

glRasterPos2f(0.0,0.0);          two character types bitmap/ stroke


glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,'c');
glColor3f(0.0,1.0,1.0);
glRasterPos2f(0.5,0.0);                    font type         character to be displayed
for(i=0;i<strlen(str);i++)
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,str[i]);
glFlush();
}
void myinit()
{
glClearColor(0.0,0.0,0.0,1.0);
gluOrtho2D(-1.0,1.0,-1.0,1.0);
}
int  main(int argc, char **argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("Simple demo");
myinit();
glutDisplayFunc(display);
glutMainLoop();
}
```

Output:



PART A

Design, develop, and implement the following programs in C/C++ using OpenGL API.

LAB PROGRAM-1

Implement Bresenham's line drawing algorithm for all types of slope.

PREAMBLE:

Bresenham's line algorithm is an algorithm that determines the points of an n-dimensional raster that should be selected in order to form a close approximation to a straight line between two points. It is commonly used to draw line primitives in a bitmap image (e.g. on a computer screen), as it uses only integer addition, sub-traction and bit shifting, all of which are very cheap operations in standard computer architectures. It is an incremental error algorithm. It is one of the earliest algorithms developed in the eld of computer graphics. An extension to the original algorithm may be used for drawing circles .

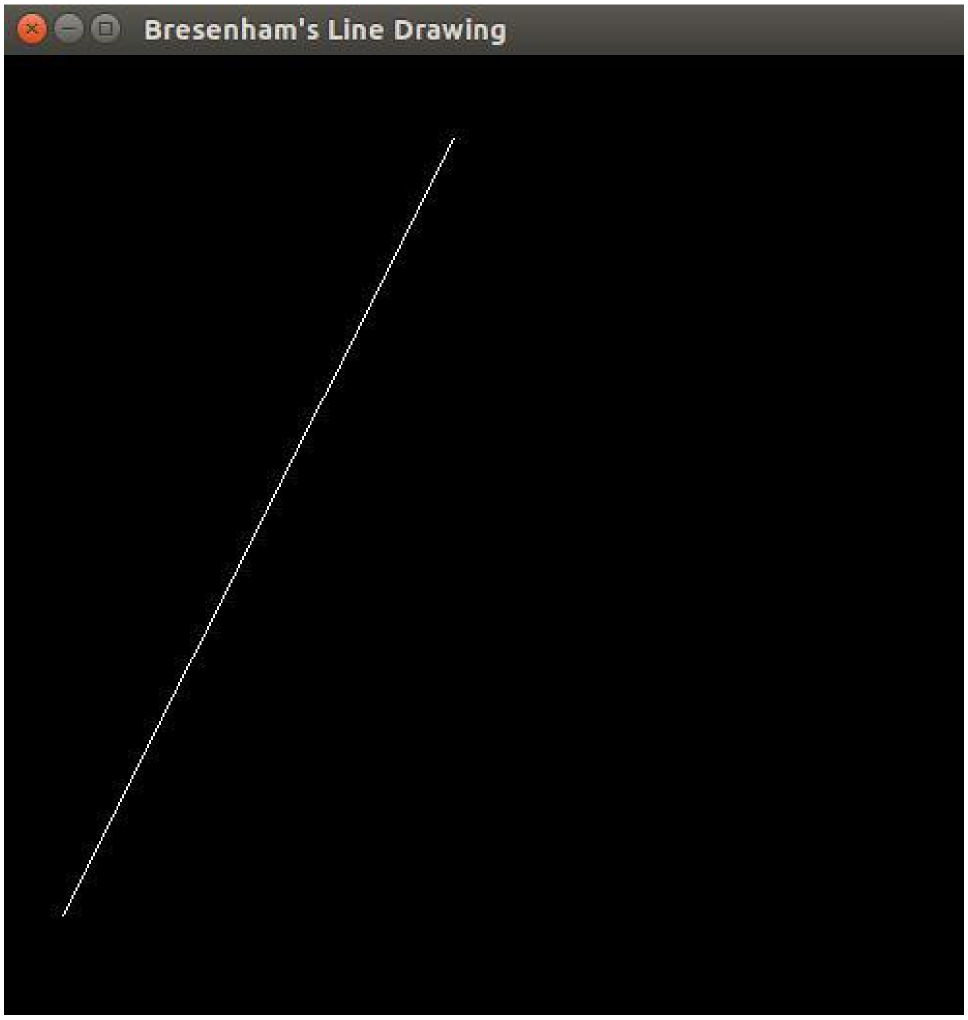
While algorithms such as Wu's algorithm are also frequently used in modern com-puter graphics because they can support antialiasing, the speed and simplicity of Bresenham's line algorithm means that it is still important. The algorithm is used in hardware such as plotters and in the graphics chips of modern graphics cards. It can also be found in many software graphics libraries. Because the algorithm is very simple, it is often implemented in either the rmware or the graphics hardware of modern graphics cards.

CODE:

```
# Bresenham's Line Drawing
#include <GL/glut.h>
#include <stdio.h>
    int x1, y1, x2, y2;
    void myInit() {
        glClear(GL_COLOR_BUFFER_BIT);
        glClearColor(0.0, 0.0, 0.0, 1.0);
        glMatrixMode(GL_PROJECTION);
        gluOrtho2D(0, 500, 0, 500);
    }
    void draw_pixel(int x, int y) {
        glBegin(GL_POINTS);
        glVertex2i(x, y);
        glEnd();
    }
    void draw_line(int x1, int x2, int y1, int y2) { int dx,
        dy, i, e;
        int incx, incy, inc1, inc2;
        int x,y;
        dx = x2-x1;
        dy = y2-y1;
        if (dx < 0) dx = -dx;
        if (dy < 0) dy = -dy;
        incx = 1;
        if (x2 < x1) incx = -1;
        incy = 1;
        if (y2 < y1) incy = -1;
        x = x1; y = y1;
        if (dx > dy) {
                draw_pixel(x, y);
                e = 2 * dy-dx;
                inc1 = 2*(dy-dx);
                inc2 = 2*dy;
                for (i=0; i<dx; i++) {
                        if (e >= 0) {
                                y += incy;
                                e += inc1;
                        }
                        else
                                e += inc2;
                        x += incx;
                        draw_pixel(x, y);
                }
        } else {
                draw_pixel(x, y);
                e = 2*dx-dy;
                inc1 = 2*(dx-dy);
                inc2 = 2*dx;
                for (i=0; i<dy; i++) {
                        if (e >= 0) {
                                x += incx;
                                e += inc1;
                        }
                        else
                                e += inc2;
                        y += incy;
                        draw_pixel(x, y);
                }
        }
}
void myDisplay() {
        draw_line(x1, x2, y1, y2);
```

```
    glFlush();
}
int main(int argc, char **argv) {
    printf( "Enter (x1, y1, x2, y2)\n");
    scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Bresenham's Line Drawing");
    myInit();
    glutDisplayFunc(myDisplay);
    glutMainLoop();
    return 0;
}
```

SAMPLE OUTPUT:



LAB PROGRAM-2

Create and rotate a triangle about the origin and a fixed point.

PREAMBLE:

In linear algebra, a rotation matrix is a matrix that is used to perform a rotation in Euclidean space. For example the matrix rotates points in the xy-Cartesian plane counterclockwise through an angle about the origin of the Cartesian coordinate

system. To perform the rotation using a rotation matrix R, the position of each point must be represented by a column vector v, containing the coordinates of the point. A rotated vector is obtained by using the matrix multiplication Rv. Since matrix multiplication has no eect on the zero vector (i.e., on the coordinates of the origin), rotation matrices can only be used to describe rotations about the origin of the coordinate system. Rotation matrices provide a simple algebraic description of such rotations, and are used extensively for computations in geometry, physics, and computer graphics. In 2-dimensional space, a rotation can be simply described by an angle of rotation, but it can be also represented by the 4 entries of a rotation

matrix with 2 rows and 2 columns. In 3-dimensional space, every rotation can be interpreted as a rotation by a given angle about a single xed axis of rotation (see Euler's rotation theorem), and hence it can be simply described by an angle and a vector with 3 entries. However, it can also be represented by the 9 entries of a rotation matrix with 3 rows and 3 columns. The notion of rotation is not commonly used in dimensions higher than 3; there is a notion of a rotational displacement, which can be represented by a matrix, but no associated single axis or angle.

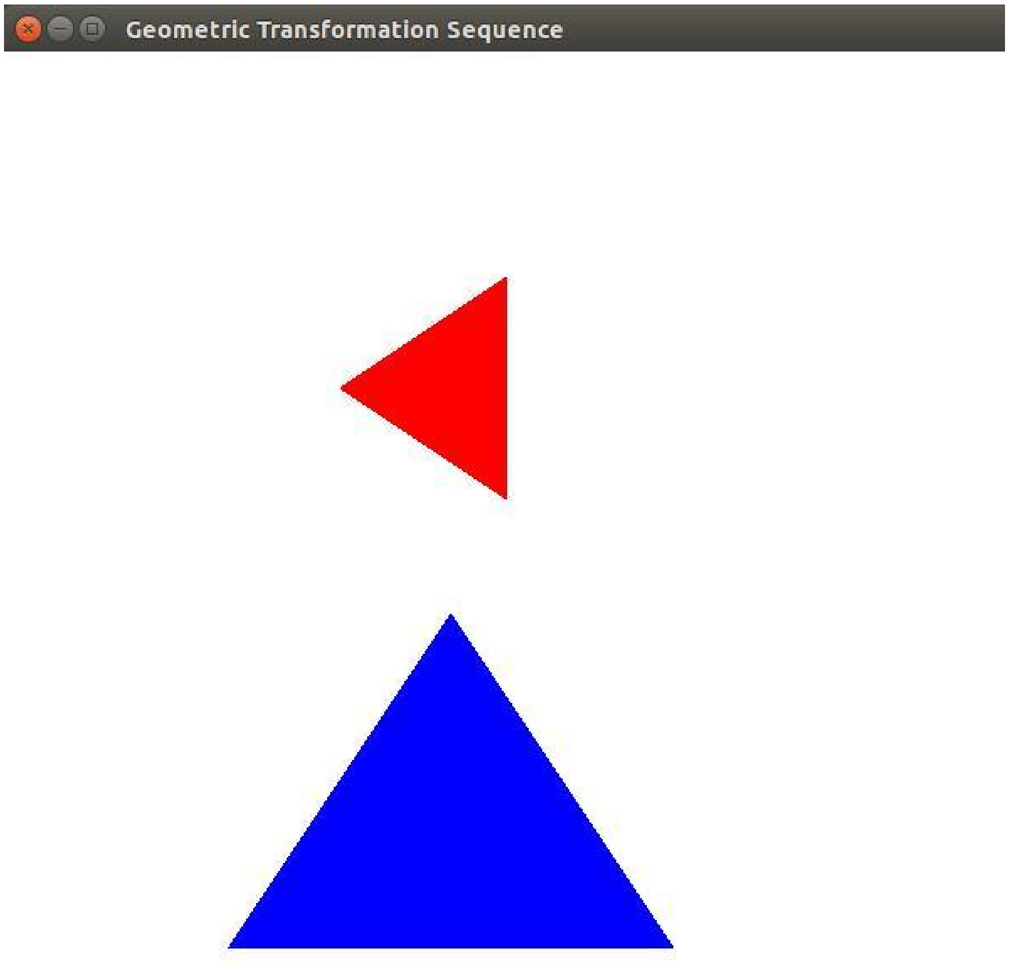
CODE:

```
#Triangle Rotation
#include<iostream>
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>
GLsizei winWidth = 600, winHeight = 600; /*   Set initial display-window size. */
GLfloat xwcMin = 0.0, xwcMax = 225.0; /*      Set range for world coordinates.        */
GLfloat ywcMin = 0.0, ywcMax = 225.0;
class wcPt2D {
public:
GLfloat x, y;
};
typedef GLfloat Matrix3x3 [3][3];
Matrix3x3 matComposite;
const GLdouble pi = 3.14159;
void init (void)
{
glClearColor (1.0, 1.0, 1.0, 0.0); }/* Set color of display window to white. */
void matrix3x3SetIdentity (Matrix3x3 matIdent3x3) /* Construct the 3 x 3 identity matrix. */
```

```
{
GLint row, col;
for (row = 0; row < 3; row++)
for (col = 0; col < 3; col++)
matIdent3x3 [row][col] = (row == col);
}
void matrix3x3PreMultiply (Matrix3x3 m1, Matrix3x3 m2)
{
GLint row, col;
Matrix3x3 matTemp;
for (row = 0; row < 3; row++)
for (col = 0; col < 3 ; col++)
matTemp [row][col] = m1 [row][0] * m2 [0][col] + m1 [row][1] *
m2 [1][col] + m1 [row][2] * m2 [2][col]; for (row = 0; row < 3;
row++)
for (col = 0; col < 3; col++)
m2 [row][col] = matTemp [row][col];
}
void translate2D (GLfloat tx, GLfloat ty)
{
Matrix3x3 matTransl;
matrix3x3SetIdentity (matTransl); /*  Initialize translation matrix to identity.   */
matTransl [0][2] = tx;
matTransl [1][2] = ty;
matrix3x3PreMultiply (matTransl, matComposite); /* Concatenate matTransl with the
composite matrix. */
}
void rotate2D (wcPt2D pivotPt, GLfloat theta)
{
Matrix3x3 matRot;
matrix3x3SetIdentity (matRot); matRot [0][0] = cos (theta);
matRot [0][1] = -sin (theta);
matRot [0][2] = pivotPt.x * (1 - cos (theta)) +
pivotPt.y * sin (theta); matRot [1][0] = sin (theta);
matRot [1][1] = cos (theta);
matRot [1][2] = pivotPt.y * (1 - cos (theta)) -
pivotPt.x * sin (theta);
/* Concatenate matRot with the composite matrix. */
matrix3x3PreMultiply (matRot, matComposite); }
void scale2D (GLfloat sx, GLfloat sy, wcPt2D fixedPt)
{
Matrix3x3 matScale;
```

```
/* Initialize scaling matrix to identity. */
matrix3x3SetIdentity (matScale); matScale [0][0]
= sx;
matScale [0][2] = (1 - sx) * fixedPt.x;
matScale [1][1] = sy;
matScale [1][2] = (1 - sy) * fixedPt.y;
/* Concatenate matScale with the composite matrix. */
matrix3x3PreMultiply (matScale, matComposite); }
/* Using the composite matrix, calculate transformed coordinates. */ void
transformVerts2D (GLint nVerts, wcPt2D * verts) {
GLint k;
GLfloat temp;
for (k = 0; k < nVerts; k++)
 {
temp = matComposite [0][0] * verts [k].x + matComposite [0][1] * verts [k].y +
matComposite [0][2];
verts [k].y = matComposite [1][0] * verts [k].x + matComposite [1][1] *
verts [k].y + matComposite [1][2];
verts [k].x = temp;
}
}
void triangle (wcPt2D *verts)
{
GLint k;
glBegin (GL_TRIANGLES);
for (k = 0; k < 3; k++)
glVertex2f (verts [k].x, verts [k].y);
glEnd ( );
}
void displayFcn (void)
{
/* Define initial position for triangle. */
GLint nVerts = 3;
wcPt2D verts [3] = { {50.0, 25.0}, {150.0, 25.0}, {100.0, 100.0} };
 /* Calculate position of triangle centroid. */
wcPt2D centroidPt;
GLint k, xSum = 0, ySum = 0;
for (k = 0; k < nVerts;      k++) {
xSum += verts [k].x;
ySum += verts [k].y;
}
centroidPt.x = GLfloat (xSum) / GLfloat (nVerts);
centroidPt.y = GLfloat (ySum) / GLfloat (nVerts);
/*   Set geometric transformation parameters.       */
wcPt2D pivPt,fixedPt;
pivPt = centroidPt;
fixedPt = centroidPt;
GLfloat tx = 0.0, ty = 100.0;
GLfloat sx = 0.5, sy = 0.5;
GLdouble theta = pi/2.0;
glClear
(GL_COLOR_BUFFER_BIT);      //     Clear display window.
glColor3f (0.0, 0.0, 1.0);             //  Set initial fill color to blue.
triangle (verts);                      //        Display blue triangle.
/*   Initialize composite matrix to identity.          */
matrix3x3SetIdentity (matComposite);
/*   Construct composite matrix for transformation sequence.  */
scale2D (sx, sy, fixedPt);          //  First transformation: Scale.
rotate2D (pivPt, theta);            //  Second transformation: Rotate
translate2D (tx, ty);               //  Final transformation: Translate.
/*   Apply composite matrix to triangle vertices.  */
transformVerts2D (nVerts, verts);
glColor3f (1.0, 0.0, 0.0);       // Set color for transformed triangle.
triangle (verts);
glFlush ( );
}
void winReshapeFcn (GLint newWidth, GLint newHeight)
{
glMatrixMode (GL_PROJECTION);
glLoadIdentity ( );
gluOrtho2D (xwcMin, xwcMax, ywcMin,
ywcMax); glClear
(GL_COLOR_BUFFER_BIT); }
int main (int argc, char ** argv)
{
glutInit (&argc, argv);
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
glutInitWindowPosition (50, 50); glutInitWindowSize
(winWidth, winHeight); glutCreateWindow ("Geometric
Transformation Sequence"); init ( );
glutDisplayFunc (displayFcn);
glutReshapeFunc (winReshapeFcn);
glutMainLoop ( );
return 0;
}
```

SAMPLE OUTPUT:



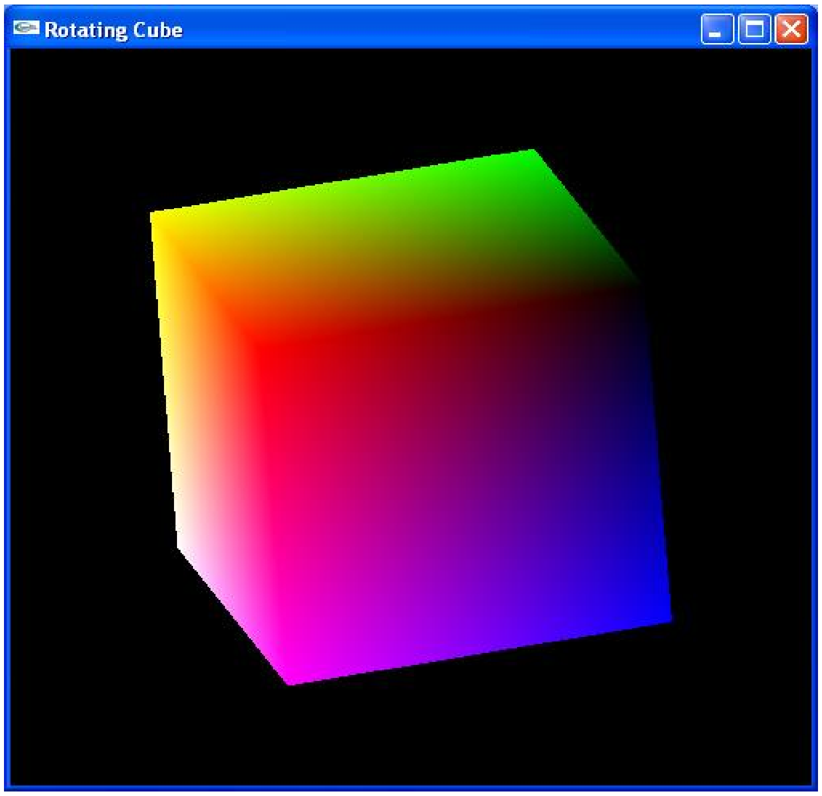
LAB PROGRAM-3:

Program to draw a color cube and spin it using OpenGL transformation matrices.

CODE:

```
#include<stdlib.h>
#include<GL/glut.h>
GLfloat vertices[][3]={{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},{1.0,1.0,-1.0},{-1.0,1.0,-1.0},{-
1.0,-1.0,1.0},{1.0,-1.0,1.0},{1.0,1.0,1.0},{-1.0,1.0,1.0}};
GLfloat normals[][3]={{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},{1.0,1.0,-1.0},{-1.0,1.0,-1.0},{-
1.0,-1.0,1.0},{1.0,-1.0,1.0},{1.0,1.0,1.0},{-1.0,1.0,1.0}};
GLfloat
colors[][3]={{0.0,0.0,0.0},{1.0,0.0,0.0},{1.0,1.0,0.0},{0.0,1.0,0.0},{0.0,0.0,1.0},{1.0,0.
0,1.0},{1.0,1.0,1.0},{0.0,1.0,1.0}};
void polygon(int a ,int b,int c, int d)
{
  glBegin(GL_POLYGON);
  glColor3fv(colors[a]);
  glNormal3fv(normals[a]);
  glVertex3fv(vertices[a]);
  glColor3fv(colors[b]);
  glNormal3fv(normals[b]);
  glVertex3fv(vertices[b]);
  glColor3fv(colors[c]);
  glNormal3fv(normals[c]);
  glVertex3fv(vertices[c]);
  glColor3fv(colors[d]);
  glNormal3fv(normals[d]);
  glVertex3fv(vertices[d]);
  glEnd();
}
void colorcube()
{
  polygon(0,3,2,1);
  polygon(2,3,7,6);
  polygon(0,4,7,3);
  polygon(1,2,6,5);
  polygon(4,5,6,7);
  polygon(0,1,5,4);
}
static GLfloat theta[]={0.0,0.0,0.0};
static GLint axis=2;
void display()
{
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
  glRotatef(theta[0],1.0,0.0,0.0);
  glRotatef(theta[1],0.0,1.0,0.0);
  glRotatef(theta[2],0.0,0.0,1.0);
  colorcube();
  glFlush();
  glutSwapBuffers();
}
void spincube()
{
     theta[axis]+=1.0;
    if(theta[axis]>360.0)theta[axis]-=360.0;
    glutPostRedisplay();
}
void mouse(int btn,int state,int x,int y)
{
  if(btn==GLUT_LEFT_BUTTON&&state==GLUT_DOWN)axis=0;
  if(btn==GLUT_MIDDLE_BUTTON&&state==GLUT_DOWN)axis=1;
  if(btn==GLUT_RIGHT_BUTTON&&state==GLUT_DOWN)axis=2;
}
void myreshape(int w,int h)
{
  glViewport(0,0,w,h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  if(w<=h)
          glOrtho(-2.0,2.0,-2.0*(GLfloat)h/(GLfloat)w,2.0*(GLfloat)h/(GLfloat)w,-
10.0,10.0);
  else
          glOrtho(-2.0*(GLfloat)w/(GLfloat)h,2.0*(GLfloat)w/(GLfloat)h,-2.0,2.0,-
10.0,10.0);
  glMatrixMode(GL_MODELVIEW);
}
void main(int argc, char **argv)
{
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
  glutInitWindowSize(500,500);
  glutInitWindowPosition(0,0);
  glutCreateWindow("Rotating Cube");
  glutDisplayFunc(display);
  glutIdleFunc(spincube);
  glutMouseFunc(mouse);
  glutReshapeFunc(myreshape);
  glEnable(GL_DEPTH_TEST);
  glutMainLoop();
}
```

SAMPLE OUTPUT:



LAB PROGRAM-4:

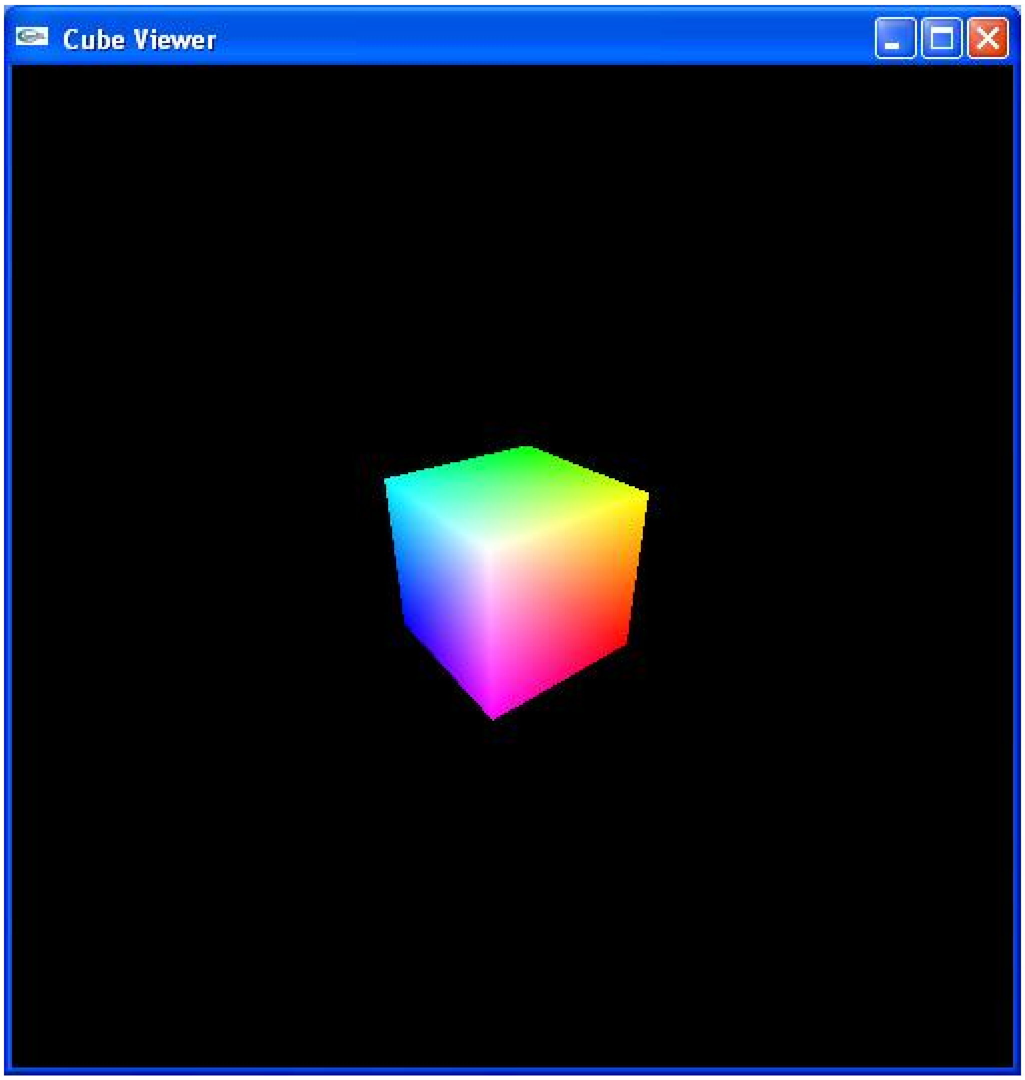
Program to draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing. Use OpenGL functions.

CODE:

```
#include<stdlib.h>
#include<GL/glut.h>
GLfloat vertices[][3]={{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},{1.0,1.0,-1.0},{-1.0,1.0,-1.0},{-
1.0,-1.0,1.0},{1.0,-1.0,1.0},{1.0,1.0,1.0},{-1.0,1.0,1.0}};
GLfloat normals[][3]={{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},{1.0,1.0,-1.0},{-1.0,1.0,-1.0},{-
1.0,-1.0,1.0},{1.0,-1.0,1.0},{1.0,1.0,1.0},{-1.0,1.0,1.0}};
GLfloat colors[][3]={{0.0,0.0,0.0},{1.0,0.0,0.0},{1.0,1.0,0.0},{0.0,1.0,0.0},{0.0,0.0,
1.0},{1.0,0.0,1.0},{1.0,1.0,1.0},{0.0,1.0,1.0}};
void polygon(int a ,int b,int c, int d)
{
  glBegin(GL_POLYGON);
  glColor3fv(colors[a]);
  glNormal3fv(normals[a]);
  glVertex3fv(vertices[a]);
  glColor3fv(colors[b]);
  glNormal3fv(normals[b]);
  glVertex3fv(vertices[b]);
  glColor3fv(colors[c]);
  glNormal3fv(normals[c]);
  glVertex3fv(vertices[c]);
  glColor3fv(colors[d]);
  glNormal3fv(normals[d]);
  glVertex3fv(vertices[d]);
  glEnd();
}
void colorcube()
{
  polygon(0,3,2,1);
  polygon(2,3,7,6);
  polygon(0,4,7,3);
  polygon(1,2,6,5);
  polygon(4,5,6,7);
  polygon(0,1,5,4);
}
static GLfloat theta[]={0.0,0.0,0.0};
static GLint axis=2;
static GLdouble viewer[]={0.0,0.0,5.0};
void display()
{
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
  gluLookAt(viewer[0],viewer[1],viewer[2],0.0,0.0,0.0,0.0,1.0,0.0);
  glRotatef(theta[0],1.0,0.0,0.0);
  glRotatef(theta[1],0.0,1.0,0.0);
  glRotatef(theta[2],0.0,0.0,1.0);
  colorcube();
  glFlush();
  glutSwapBuffers();
}
void mouse(int btn,int state,int x,int y)
{
  if(btn==GLUT_LEFT_BUTTON&&state==GLUT_DOWN)axis=0;
  if(btn==GLUT_MIDDLE_BUTTON&&state==GLUT_DOWN)axis=1;
  if(btn==GLUT_RIGHT_BUTTON&&state==GLUT_DOWN)axis=2;
  theta[axis]+=2.0;
    if(theta[axis]>360.0)theta[axis]-=360.0;
    glutPostRedisplay();
}
void keys(unsigned char key,int x,int y)
{
  if(key=='x') viewer[0]-=1.0;
  if(key=='X') viewer[0]+=1.0;
  if(key=='y') viewer[1]-=1.0;
  if(key=='Y') viewer[1]+=1.0;
  if(key=='z') viewer[2]-=1.0;
  if(key=='Z') viewer[2]+=1.0;
  glutPostRedisplay();
}
void myreshape(int w,int h)
{
  glViewport(0,0,w,h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  if(w<=h)
            glFrustum(-2.0,2.0,-
2.0*(GLfloat)h/(GLfloat)w,2.0*(GLfloat)h/(GLfloat)w,2.0,20.0);
  else
            glFrustum(-2.0*(GLfloat)w/(GLfloat)h,2.0*(GLfloat)w/(GLfloat)h,-
2.0,2.0,2.0,20.0);
  glMatrixMode(GL_MODELVIEW);
}
void main(int argc, char **argv)
{
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
  glutInitWindowSize(500,500);
```

```
        glutInitWindowPosition(0,0);
        glutCreateWindow(" Cube Viewer");
        glutReshapeFunc(myreshape);
        glutDisplayFunc(display);
        glutMouseFunc(mouse);
        glutKeyboardFunc(keys);
        glEnable(GL_DEPTH_TEST);
        glutMainLoop();
    }
```

SAMPLE OUTPUT:



LAB PROGRAM- 5:

Clip a line by using Cohen-Sutherland line-clipping algorithm.

PREAMBLE:

Line-Clipping

In computer graphics, **line clipping** is the process of removing lines or portions of lines outside of an area of interest. Typically, any line or part thereof which is outside of the viewing area is removed.

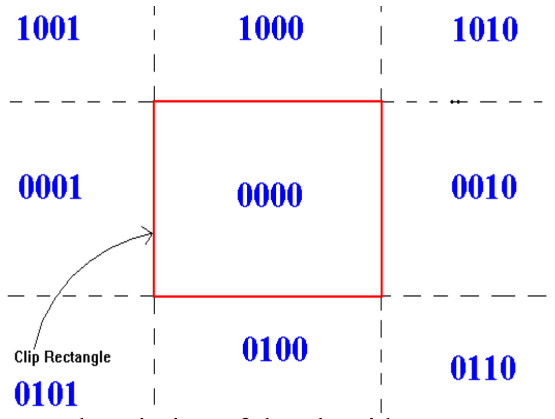
Cohen-Sutherland Line-Clipping algorithm:

This algorithm divides a 2D space into 9 parts, of which only the middle part (viewport) is visible. The algorithm includes, excludes or partially includes the line based on where the two endpoints are:

- Both endpoints are in the viewport (bitwise OR of endpoints == 0): trivial accept.
- Both endpoints are in the same part, which is not visible (bitwise AND of endpoints != 0): trivial reject.
- Both endpoints are in different parts: In case of this non trivial situation the algorithm finds one of the two points that are outside the viewport (there is at least one point outside). The intersection of the outpoint and extended viewport border is then calculated (i.e. with the parametric equation for the line) and this new point replaces the outpoint. The algorithm repeats until a trivial accept or reject occurs.

Steps for Cohen-Sutherland Algorithm

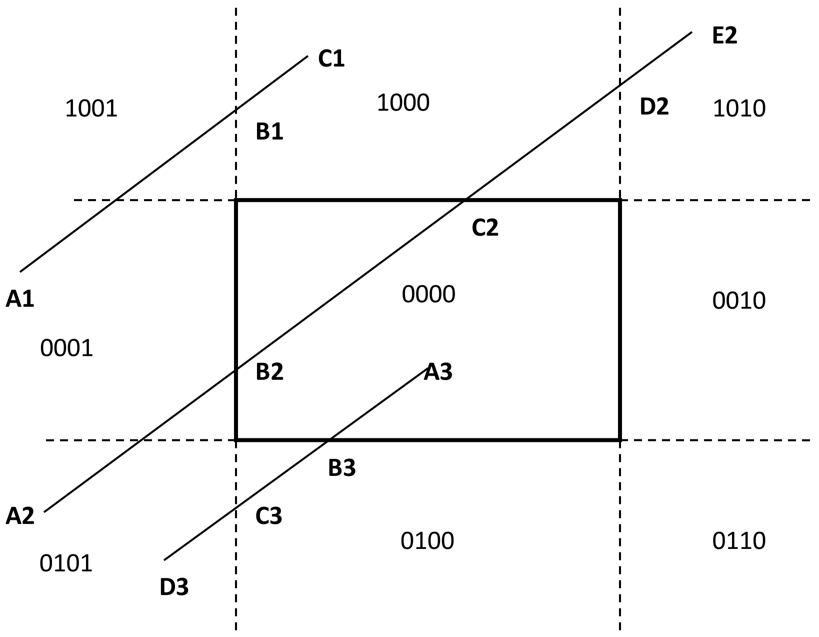
1. End-points pairs are checked for trivial acceptance or rejection using outcode (region code, each of the 9 parts are assigned a 4 bit code indicating their location with respect to the window/ region of interest).
2. If not trivially accepted or rejected, divide the line segment into two at a clip edge;
3. Iteratively clipped by test trivial-acceptance or trivial-rejection, and divided into two segments until completely inside or trivial-rejection.



Alternate description of the algorithm:

1. Encode end points

Bit 0 = point is left of window
Bit 1 = point is right of window
Bit 2 = point is below window
Bit 3 = point is above window
2. If C0 $\wedge$Cend $\neq$ 0 then P0Pend is trivially rejected
3. If C0 $\vee$ Cend = 0 thenP0Pend is trivially accepted
4. Otherwise subdivide and go to step 1 with new segment
C0 = Bit code of **P0**
Cend = Bit code of **Pend**



Clip order: Left, Right, Bottom, Top

1) **A1C1**	1) **A2E2**	1) **A3D3**
2) **B1C1**	2) **B2E2**	2) **A3C3**
3) **reject**	3) **B2D2**	3) **A3B3**
	4) **B2C2**	4) **accept**
	5) **accept**	

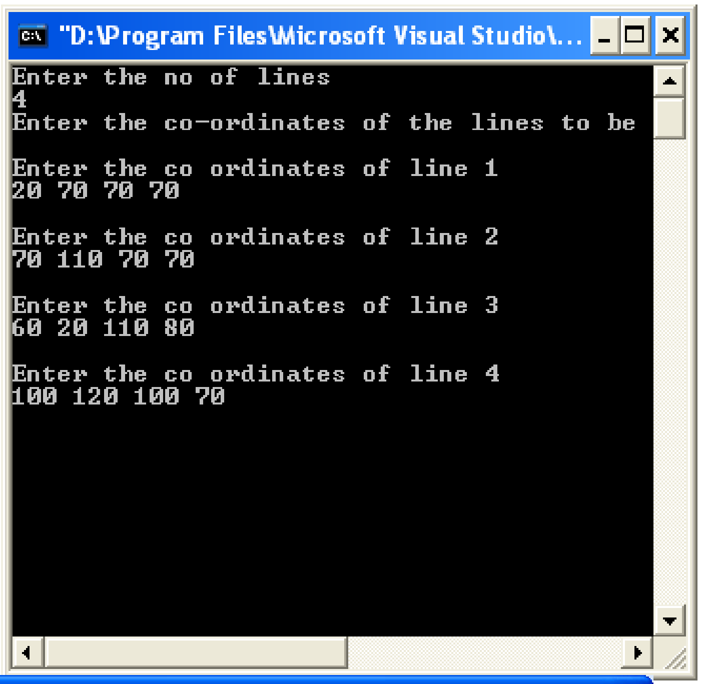
CODE:

```
#include<GL/glut.h>
#include<stdio.h>
double xmin=50,xmax=100,ymin=50,ymax=100;
double xvmin=200,xvmax=300,yvmin=200,yvmax=300;
float xc[10],yc[10];
int n;
typedef int outcode;
const int TOP=8;
const int BOTTOM=4;
const int RIGHT=2;
const int LEFT=1;
/*computing/ assigning region codes to end points of line*/
outcode ComputeCode(double x, double y)
{
  outcode code=0;
  if(y>ymax)
          code|=TOP;
  else if(y<ymin)
          code|=BOTTOM;
  if(x>xmax)
          code|=RIGHT;
  else if(x<xmin)
          code|=LEFT;
  return code;
}
void CohenSutherlandLineClipper(double x0,double y0,double x1,double y1)
{
  outcode outcode0,outcode1,outcodeout;
  double x,y;
  bool accept=false,done=false;
  outcode0=ComputeCode(x0,y0);
  outcode1=ComputeCode(x1,y1);
  do
  {
          if(!(outcode0|outcode1))
          {accept=true;done=true;}
          else if(outcode0& outcode1)done=true;
          else
          {
                  outcodeout=outcode0?outcode0:outcode1;
                  if(outcodeout&TOP)
                  {
                          y=ymax;
                          x=x0+(x1-x0)*(ymax-y0)/(y1-y0);
                  }
                  else if(outcodeout&BOTTOM)
                  {
                          y=ymin;
                          x=x0+(x1-x0)*(ymin-y0)/(y1-y0);
                  }
                  else if(outcodeout & RIGHT)
                  {
                          x=xmax;
                          y=y0+(y1-y0)*(xmax-x0)/(x1-x0);
                  }
```

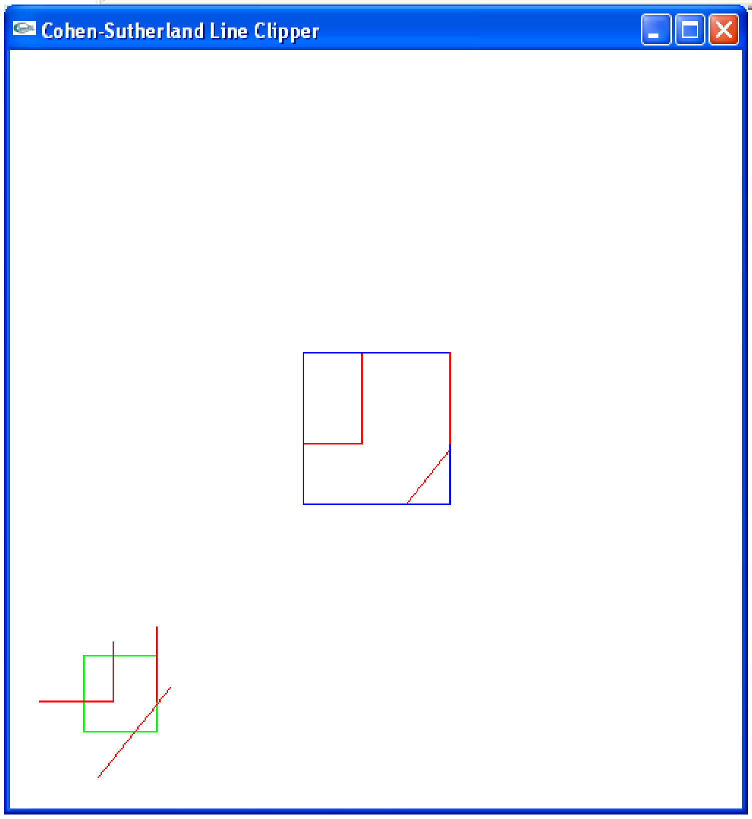
```
                else
                {
                        x=xmin;
                        y=y0+(y1-y0)*(xmin-x0)/(x1-x0);
                }
                if(outcodeout==outcode0)
                {
                        x0=x;
                        y0=y;
                        outcode0=ComputeCode(x0,y0);
                }
                else
                {
                        x1=x;
                        y1=y;
                        outcode1=ComputeCode(x1,y1);
                }
        }
  }while(done==false);
  if(accept)
  {
        double sx=(xvmax-xvmin)/(xmax-xmin);
        double sy=(yvmax-yvmin)/(ymax-ymin);
        double vx0=xvmin+(x0-xmin)*sx;
        double vy0=yvmin+(y0-ymin)*sy;
        double vx1=xvmin+(x1-xmin)*sx;
        double vy1=yvmin+(y1-ymin)*sy;
        glColor3f(0.0,0.0,1.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(xvmin,yvmin);
        glVertex2f(xvmax,yvmin);
        glVertex2f(xvmax,yvmax);
        glVertex2f(xvmin,yvmax);
        glEnd();
        glColor3f(1.0,0.0,0.0);
        glBegin(GL_LINES);
        glVertex2f(vx0,vy0);
        glVertex2f(vx1,vy1);
        glEnd();
  }
}
void display()
{
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(0.0,1.0,0.0);
  glBegin(GL_LINE_LOOP);
  glVertex2f(xmin,ymin);
  glVertex2f(xmax,ymin);
  glVertex2f(xmax,ymax);
  glVertex2f(xmin,ymax);
  glEnd();
  glColor3f(1.0,0.0,0.0);
  glBegin(GL_LINES);
  for(int i=0;i<n*2;i=i+2)
  {
  glVertex2f(xc[i],yc[i]);
  glVertex2f(xc[i+1],yc[i+1]);
  }
  glEnd();
  for( i=0;i<n*2;i=i+2)
  CohenSutherlandLineClipper(xc[i],yc[i],xc[i+1],yc[i+1]);
  glFlush();
}

void myinit()
{
  glClearColor(1.0,1.0,1.0,1.0);
  gluOrtho2D(0.0,499.0,0.0,499.0);
}

void main(int argc, char **argv)
{
  int c=0;
  printf("Enter the no of lines\n");
  scanf("%d",&n);
  printf("Enter the co-ordinates of the lines to be clipped\n");
for(int i=0;i<n;i++)
  {
          printf("\nEnter the co ordinates of line %d\n",i+1);
          for(int j=0;j<2;j++)
          {
                  scanf("%f%f",&xc[c],&yc[c]);c++;
          }
}
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
  glutInitWindowPosition(0,0);
  glutInitWindowSize(500,500);
  glutCreateWindow("Cohen-Sutherland Line Clipper");
  myinit();

glutDisplayFunc(display);
```

```
    glutMainLoop();
}
```

SAMPLE OUTPUT:



```
Enter the no of lines
4
Enter the co-ordinates of the lines to be

Enter the co ordinates of line 1
20 70 70 70

Enter the co ordinates of line 2
70 110 70 70

Enter the co ordinates of line 3
60 20 110 80

Enter the co ordinates of line 4
100 120 100 70
```



LAB PROGRAM-6:

To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the surfaces of the solid object used in the scene.

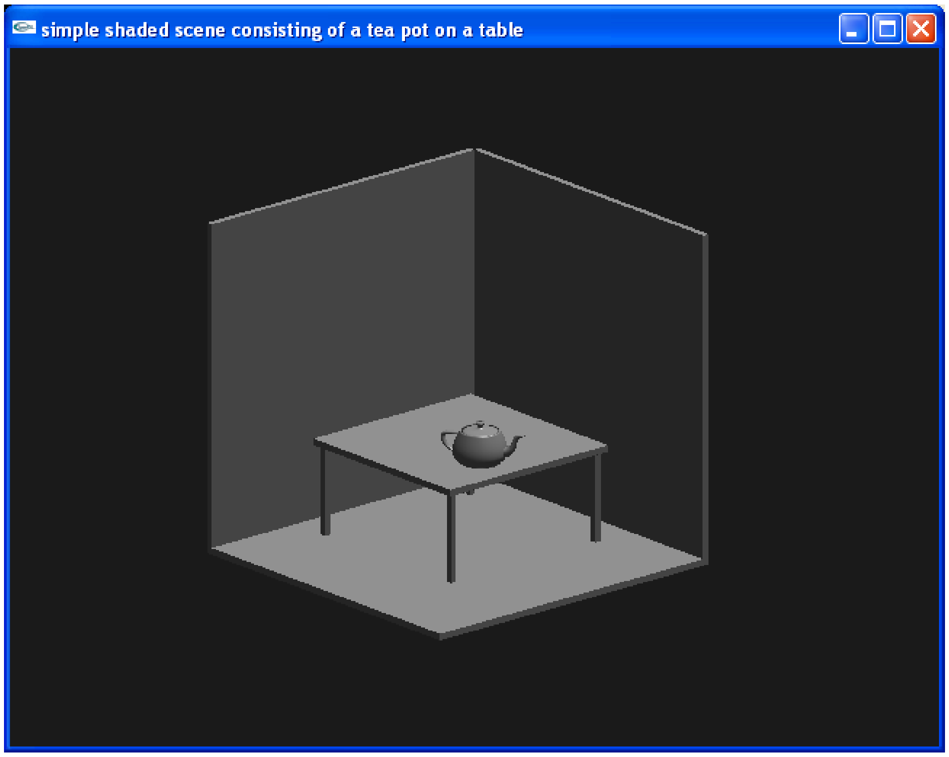
CODE:

```
#include<GL/glut.h>
#include<stdio.h>
void wall(double thickness)
{
  glPushMatrix();
  glTranslated(0.5,0.5*thickness,0.5);
  glScaled(1.0,thickness, 1.0);
  glutSolidCube(1.0);
  glPopMatrix();
}
void tableLeg(double thick, double len)
{
  glPushMatrix();
  glTranslated(0,len/2,0);
  glScaled(thick,len,thick);
  glutSolidCube(1.0);
  glPopMatrix();
}
void table(double topWid, double topThick,double legThick, double legLen)
{
  glPushMatrix();
  glTranslated(0,legLen,0);
  glScaled(topWid,topThick,topWid);
  glutSolidCube(1.0);
  glPopMatrix();
  double dist=0.95*topWid/2.0-legThick/2.0;
  glPushMatrix();
  glTranslated(dist,0,dist);
  tableLeg(legThick,legLen);
  glTranslated(0.0,0.0,-2*dist);
  tableLeg(legThick,legLen);
  glTranslated(-2*dist,0,2*dist);
  tableLeg(legThick,legLen);
  glTranslated(0,0,-2*dist);
  tableLeg(legThick,legLen);
  glPopMatrix();
}
```

```
void displaySolid(void)
{
  glLoadIdentity();
  GLfloat mat_ambient[] = {0.7f, 0.7f, 0.7f, 1.0f}; // gray
  GLfloat mat_diffuse[] = {.5f, .5f, .5f, 1.0f};
  GLfloat mat_specular[] = {1.0f, 1.0f, 1.0f, 1.0f};
  GLfloat mat_shininess[] = {50.0f};
  glMaterialfv (GL_FRONT, GL_AMBIENT, mat_ambient);
  glMaterialfv (GL_FRONT, GL_DIFFUSE, mat_diffuse);
  glMaterialfv (GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialfv (GL_FRONT, GL_SHININESS, mat_shininess);
  GLfloat lightIntensity[] = {0.9f, 0.9f, 0.9f, 1.0f};
  GLfloat light_position[] = {2.0f, 6.0f, 3.0f, 0.0f};
  glLightfv (GL_LIGHT0, GL_POSITION, light_position);
  glLightfv (GL_LIGHT0, GL_DIFFUSE, lightIntensity);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity();
  double winHt = 1.0; //half-height of window
  glOrtho (-winHt * 64/48.0, winHt*64/48.0, -winHt, winHt, 0.1, 100.0);
  glMatrixMode (GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt (2.3, 1.3, 2.0, 0.0, 0.25, 0.0, 0.0, 1.0, 0.0);
  glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glPushMatrix();
  glTranslated (0.6, 0.38, 0.5);
  glRotated (30, 0, 1, 0);
  glutSolidTeapot (0.08);
  glPopMatrix ();
  glPushMatrix();
  glTranslated (0.4, 0, 0.4);
  table (0.6, 0.02, 0.02, 0.3);
  glPopMatrix();
  wall (0.02);
  glPushMatrix();
  glRotated (90.0, 0.0, 0.0, 1.0);
  wall (0.02);
  glPopMatrix();
  glPushMatrix();
  glRotated (-90.0, 1.0, 0.0, 0.0);
  wall (0.02);
  glPopMatrix();
  glFlush();
}
void main(int argc, char ** argv)
{
  glutInit (&argc, argv);
```

```
    glutInitDisplayMode (GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
    glutInitWindowSize (640, 480);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("simple shaded scene consisting of a tea pot on a table");
    glutDisplayFunc (displaySolid);
    glEnable (GL_LIGHTING);
    glEnable (GL_LIGHT0);
    glShadeModel (GL_SMOOTH);
    glEnable (GL_DEPTH_TEST);
    glEnable (GL_NORMALIZE);
    glClearColor (0.1, 0.1, 0.1, 0.0);
    glViewport (0, 0, 640, 480);
    glutMainLoop();
}
```

SAMPLE OUTPUT:



LAB PROGRAM-7:

Design, develop and implement recursively subdivided a tetrahedron to form a 3D sierpinski gasket. The number of recursively steps is to be specified by the user.

Sierpinski's Triangle is a very famous fractal that's been seen by most advanced math students. This fractal consists of one large triangle, which contains an innite amount of smaller triangles within. The innite amount of triangles is easily understood if the fractal is zoomed in many levels. Each zoom will show yet more previously unseen triangles embedded in the visible ones.

Creating the fractal requires little computational power. Even simple graphing calcu-lators can easily make this image. The fractal is created pixel by pixel, using random numbers; the fractal will be slightly dierent each time due to this. Although, if you were to run the program repeatedly, and allow each to use an innite amount of time, the results would be always identical. No one has an innite amount of time, but the dierences in the nite versions are very small.

A fractal is generally "a rough or fragmented geometric shape that can be split into parts, each of which is (at least approximately) a reduced-size copy of the whole".

To generate this fractal, a few steps are involved.

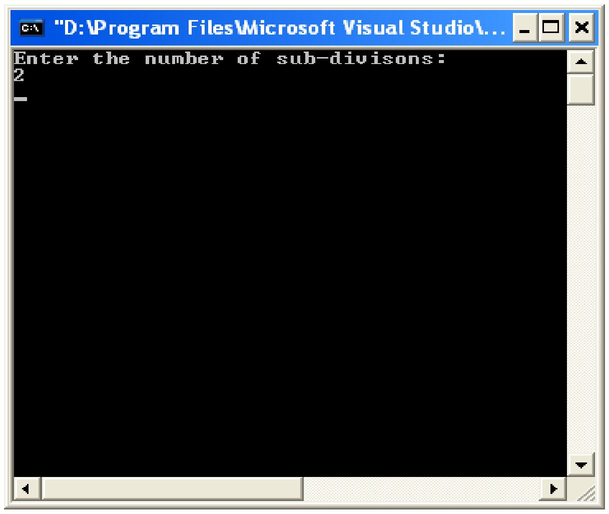
We begin with a triangle in the plane and then apply a repetitive scheme of operations to it (when we say triangle here, we mean a blackened, 'lled-in' triangle). Pick the midpoints of its three sides. Together with the old verticies of the original triangle, these midpoints dene four congruent triangles of which we drop the center one. This completes the basic construction step. In other words, after the rst step we have three congruent triangles whose sides have exactly half the size of the original triangle and which touch at three points which are common verticies of two contiguous trianges. Now we follow the same procedure with the three remaining triangles and repeat the basic step as often as desired. That is, we start with one triangle and then produce 3, 9, 27, 81, 243, . . . triangles, each of which is an exact scaled down version of the triangles in the preceeding step.

CODE:

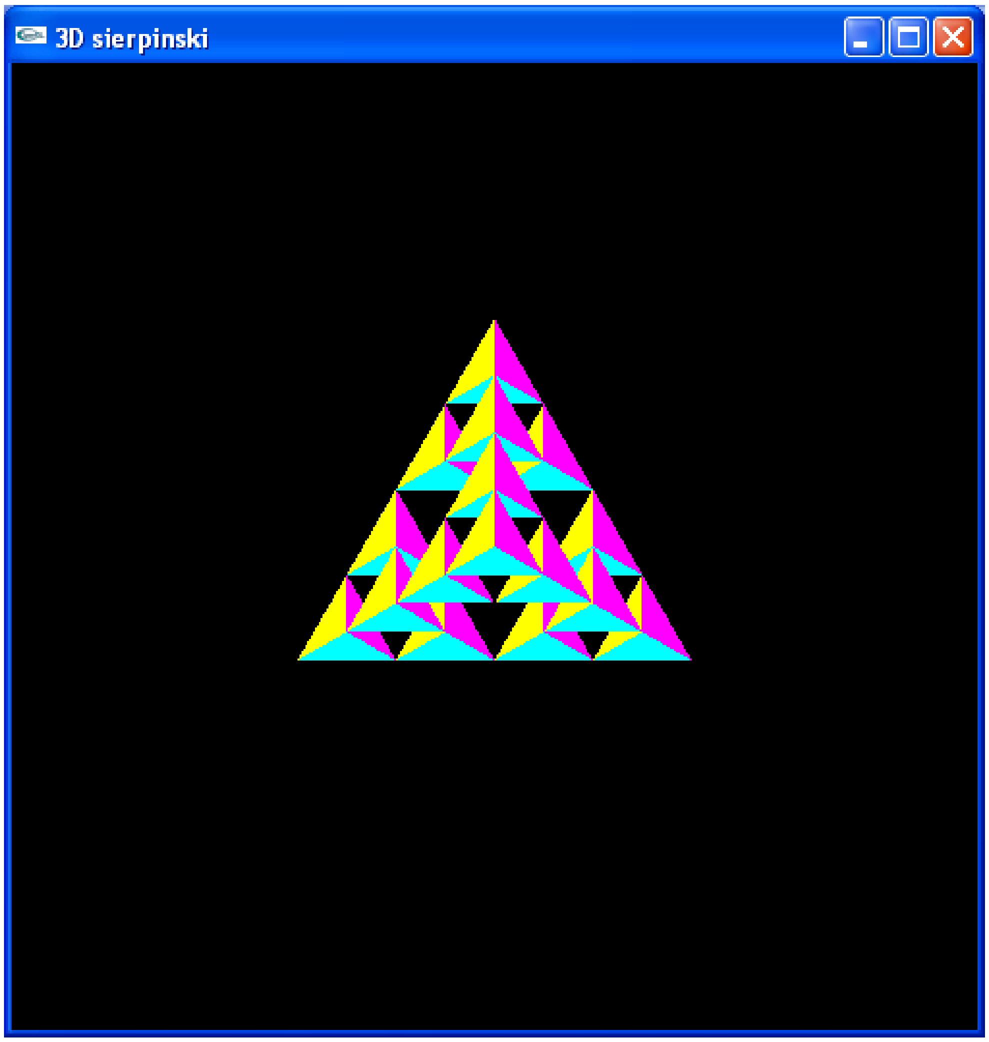
```
#include<GL/glut.h>
#include<stdio.h>
typedef float point[3];
point v[4]={{0.0,0.0,1.0},{0.0,0.942809,-0.33333},{-0.816497,-0.471405,-
0.333333},{0.816497,-0.471405,-0.333333}};
int n;/*recursive steps*/
void triangle(point a,point b,point c)
{
  glBegin(GL_TRIANGLES);
  glNormal3fv(a);
  glVertex3fv(a);
  glVertex3fv(b);
  glVertex3fv(c);
  glEnd();
}
void tetrahedron(point a, point b, point c, point d)
{
            glColor3f(1.0,1.0,0.0);
  triangle(a,b,c);
            glColor3f(0.0,1.0,1.0);
  triangle(a,c,d);
            glColor3f(1.0,0.0,1.0);
  triangle(a,d,b);
            glColor3f(0.0,0.0,0.0);
  triangle(b,d,c);
}
void divide_tetrahedron(point a, point b, point c, point d, int n)
{
  int j;
  point v1,v2,v3,v4,v5,v6;/*variables to store six mid points*/

if(n>0)
  { /*the six mid-points of the six edges of a tetrahedron*/
  for(j=0;j<3;j++)v1[j]=(a[j]+b[j])/2;/*mid point of edge ab*/
  for(j=0;j<3;j++)v2[j]=(a[j]+c[j])/2;/*mid point of edge ac*/
  for(j=0;j<3;j++)v3[j]=(a[j]+d[j])/2;/*mid point of edge ad*/
  for(j=0;j<3;j++)v4[j]=(b[j]+c[j])/2;/*mid point of edge bc*/
  for(j=0;j<3;j++)v5[j]=(c[j]+d[j])/2;/*mid point of edge cd*/
  for(j=0;j<3;j++)v6[j]=(b[j]+d[j])/2;/*mid point of edge bd*/

  divide_tetrahedron(a, v1,v2,v3,n-1);/*a tetrahedron formed from vertices a,mid point
of ab,ac,ad edge*/
  divide_tetrahedron( v1,b,v4,v6,n-1);/*a tetrahedron formed from vertices b,mid point
of ab,bc,bd edge*/
  divide_tetrahedron( v2,v4,c,v5,n-1);/*a tetrahedron formed from vertices c,mid point
of ac,bc,cd edge*/
  divide_tetrahedron( v3,v6,v5,d,n-1);/*a tetrahedron formed from vertices d,mid point
of ad,cd,bd edge*/
  }
  else
  tetrahedron(a,b,c,d);/*drawing the tetrahedrons*/
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glEnable(GL_DEPTH_TEST);
divide_tetrahedron(v[0],v[1],v[2],v[3],n);
```

```
glFlush();
}
void myreshape(int w,int h)
{
glViewport(0,0,w,h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
if(w<=h)
{

glOrtho
(-2.0,2.0,-2.0*(GLfloat)h/(GLfloat)w,2.0*(GLfloat)h/(GLfloat)w,-10.0,10.0);}
  else
glOrtho(-2.0*(GLfloat)w/(GLfloat)h,2.0*(GLfloat)w/(GLfloat)h,-2.0,2.0,-10.0,10.0);
  glMatrixMode(GL_MODELVIEW);
  glutPostRedisplay();
}
void main(int argc,char **argv)
{
  printf("Enter the number of sub-divisons:\n");
  scanf("%d",&n);
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE|GLUT_DEPTH);
  glutInitWindowPosition(0,0);
  glutInitWindowSize(500,500);
  glutCreateWindow("3D sierpinski");
  glutReshapeFunc(myreshape);

  glutDisplayFunc(display);
  glClearColor(0.0,0.0,0.0,1.0);
  glutMainLoop();
}
```

Output:



```
Enter the number of sub-divisons:
2
_
```


3D sierpinski

LAB PROGRAM-8:

Develop a menu driven program to animate a ag using Bezier Curve algorithm.

PREAMBLE

Bezier curve is discovered by the French engineer Pierre Bezier. These curves can be generated under the control of other points. Approximate tangents by using control points are used to generate curve. The Bezier curve can be represented mathemati-cally as

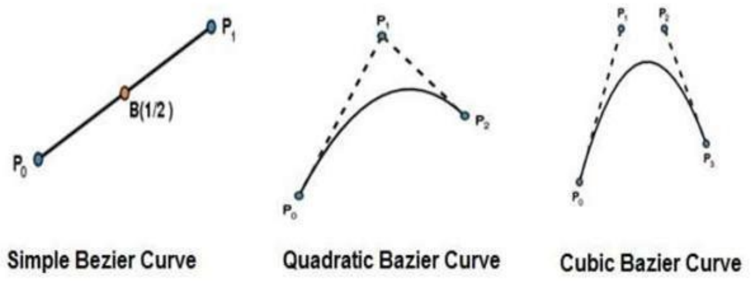
$$\sum_{k=0}^{n} P_i B_i^n(t)$$

Where pi is the set of points and Bni(t) represents the Bernstein polynomials which are given by

$$B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i$$

Where n is the polynomial degree, i is the index, and t is the variable.

The simplest Bezier curve is the straight line from the point P0 to P1. A quadratic Bezier curve is determined by three control points. A cubic Bezier curve is deter-mined by four control points.



Bezier curves generally follow the shape of the control polygon, which consists of the segments joining the control points and always pass through the rst and last control points. They are contained in the convex hull of their dening control points. The degree of the polynomial dening the curve segment is one less that the number of dening polygon point. Therefore, for 4 control points, the degree of the polynomial is 3, i.e. cubic polynomial. A Bezier curve generally follows the shape of the dening polygon. The direction of the tangent vector at the end points is same as that of the vector determined by rst and last segments. The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points. No straight line intersects a Bezier curve more times than it intersects its control polygon.

They are invariant under an a-ne transformation. Bezier curves exhibit global control means moving a control point alters the shape of the whole curve. A given Bezier curve can be subdivided at a point t=t0 into two Bezier segments which join together at the point corresponding to the parameter value t=t0.

Concepts Used: Algorithm: OpenGL Commands Familiarized :

+ glMatrixMode
+ glLoadIdentity
+ gluOrtho2D
+ glFlush
+ glColor3f
+ glBegin

1.8.3 CODE:

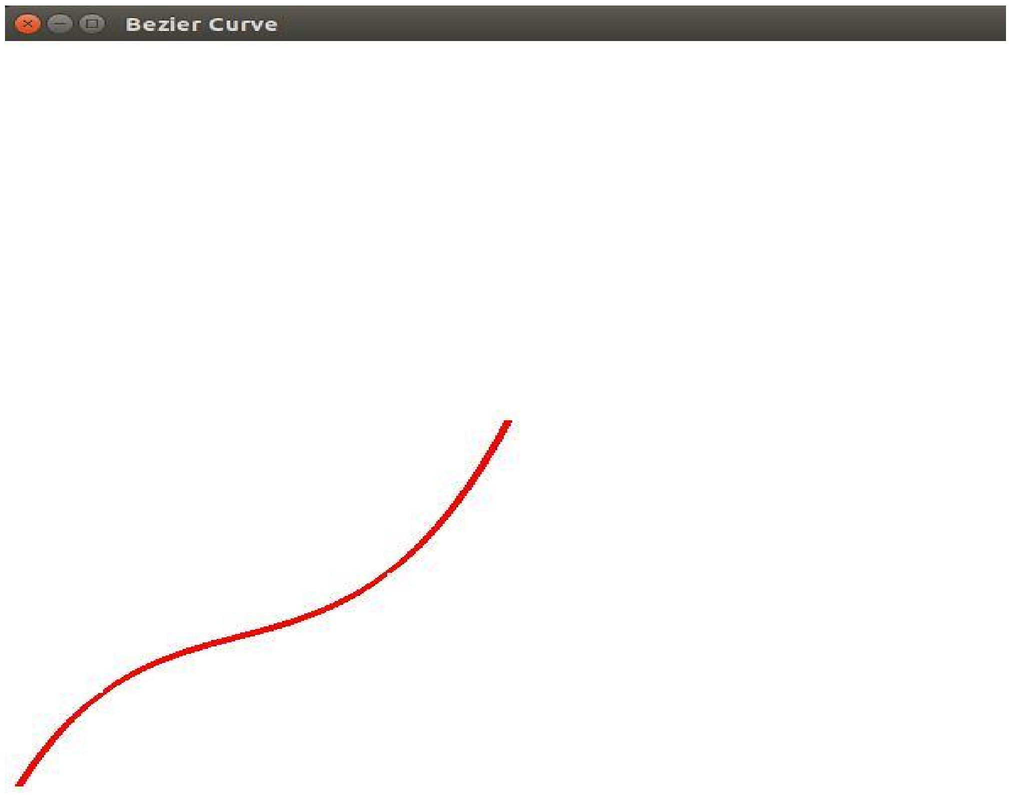
# Bezier Curve

```
#include<GL/glut.h>
#include<math.h>
#include<stdio.h>
void bezierCoefficients(int n,int *c)
{
    int k,i;
    for(k=0;k<=n;k++)
    {
        c[k]=1;
        for(i=n;i>=k+1;i--)
        c[k]*=i;
        for(i=n-k;i>=2;i--)
            c[k]/=i;

    }
}

void display()
{

    int cp[4][2]={{10,10},{100,200},{200,50},{300,300}};
int c[4],k,n=3;
float x,y,u,blend;
bezierCoefficients(n,c);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0,0.0,0.0);
    glLineWidth(5.0);
    glBegin(GL_LINE_STRIP);

    for(u=0;u<1.0;u+=0.01)
    {x=0;y=0;
        for(k=0;k<4;k++)
        {
            blend=c[k]*pow(u,k)*pow(1-u,n-k);
            x+=cp[k][0]*blend;
            y+=cp[k][1]*blend;
        }
        glVertex2f(x,y);

    }
    glEnd();
    glFlush();
}
void myinit()
{
    glClearColor(1.0,1.0,1.0,1.0);
        glColor3f(1.0,0.0,0.0);
        glPointSize(5.0);
        gluOrtho2D(0.0,600,0.0,600.0);
}
int main(int argc, char ** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(600,600);
    glutCreateWindow("Bezier Curve");
    glutDisplayFunc(display);
    myinit();

    glutMainLoop();
    return 0;

}
```

RUN:

```
gcc 8.cpp -lglut -lGL -lGLUT
./a.out
```

SAMPLE OUTPUT:



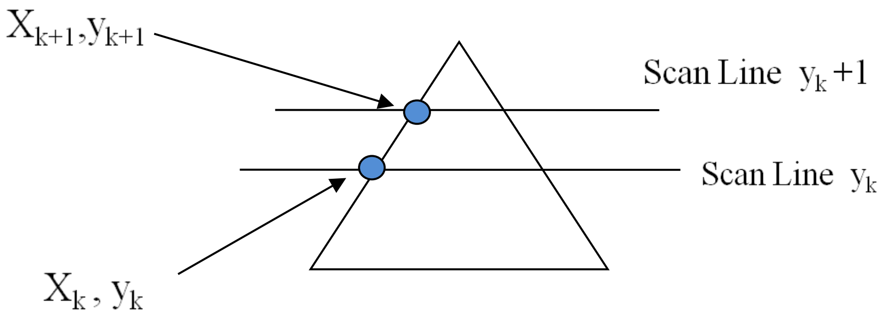
LAB PROGRAM-9:

Program to fill any given polygon using scan-line area filling algorithm.(Use appropriate data structures.)

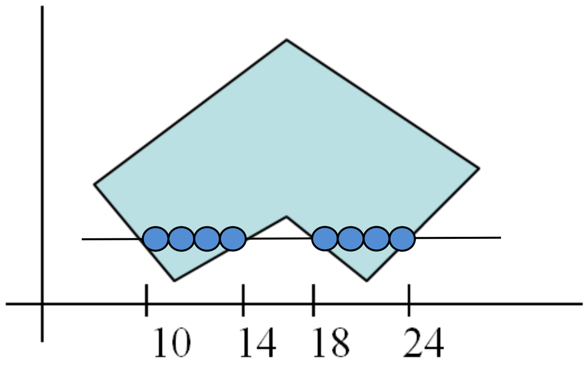
Algorithm at work:

Scan Line Polygon Fill Algorithms

- A standard output primitive in general graphics package is a solid color or patterned polygon area:
- There are two basic approaches to filling on raster systems.
- Determine overlap Intervals for scan lines that cross that area.
- Start from a given interior point and paint outward from this point until we encounter the boundary
- The first approach is mostly used in general graphics packages, however second approach is used in applications having complex boundaries and interactive painting systems



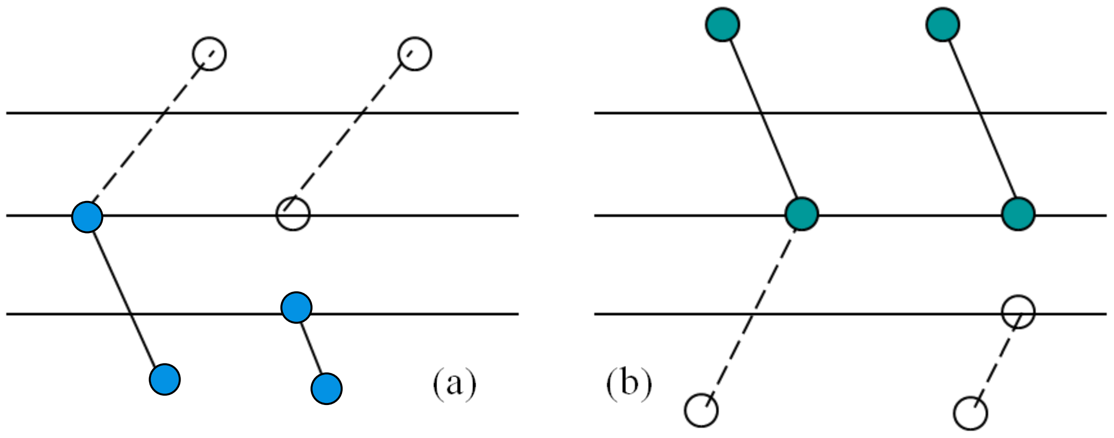
Scan Line Polygon Fill Algorithm:



Interior pixels along a scan line passing through a polygon area

- For each scan line crossing a polygon are then sorted from left to right, and the corresponding frame buffer positions between each intersection pair are set to the specified color.

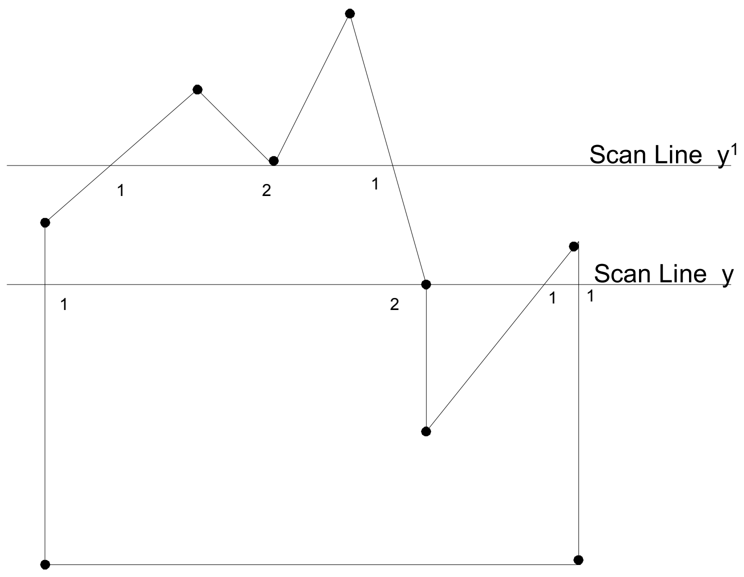
- These intersection points are then sorted from left to right , and the corresponding frame buffer positions between each intersection pair are set to specified color
- *In the given example (previous slide) , four pixel intersections define stretches from x=10 to x=14 and x=18 to x=24*
- Some scan-Line intersections at polygon vertices require special handling:
- A scan Line passing through a vertex intersects two polygon edges at that position, adding two points to the list of intersections for the scan Line
- In the given example , scan Line y intersects five polygon edges and the scan Line y‘ intersects 4 edges although it also passes through a vertex
- y‘ correctly identifies internal pixel spans ,but need some extra processing
- One way to resolve this is also to shorten some polygon edges to split those vertices that should be counted as one intersection
- When the end point y coordinates of the two edges are increasing , the y value of the upper endpoint for the current edge is decreased by 1
- When the endpoint y values are monotonically decreasing, we decrease the y coordinate of the upper endpoint of the edge following the current edge



Adjusting endpoint values for a polygon, as we process edges in order around the polygon perimeter. The edge currently being processed is indicated as a solid like. In (a), the y coordinate of the upper endpoint of the current edge id decreased by 1. In (b), the y coordinate of the upper end point of the next edge is decreased by 1

- The topological difference between scan line y and scan line y’ is …
- For Scan line y, the two intersecting edges sharing a vertex are on opposite sides of the scan line …!
  But for scan line y’, the two intersecting edges are both above the scan line.
- Thus, the vertices that require additional processing are those that have connecting edges on opposite sides of scan line.
- We can identify these vertices by tracing around the polygon boundary either in clock-wise or anti-clockwise order and observing the relative changes in vertex y coordinates as we move from one edge to the next.

- If the endpoint y values of two consecutive edges monotonically increase or decrease, we need to count the middle vertex as a single intersection point for any scan line passing through that vertex.
- Otherwise, the shared vertex represents a local extremum (min. or max.) on the polygon boundary, and the two edge intersections with the scan line passing through that vertex can be added to the intersection list



Intersection points along the scan lines that intersect polygon vertices. Scan line y generates an odd number of intersections, but scan line y generates an even number of intersections that can be paired to identify correctly the interior pixel spans.

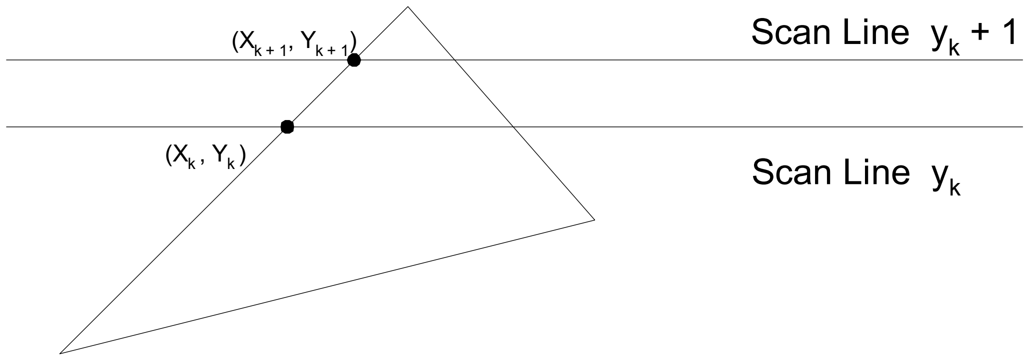
The scan conversion algorithm works as follows

i. Intersect each scanline with all edges
ii. Sort intersections in x
iii. Calculate parity of intersections to determine in/out
iv. Fill the "in" pixels

Special cases to be handled:

i. Horizontal edges should be excluded
ii. For vertices lying on scanlines,
    i. count twice for a change in slope.
    ii. Shorten edge by one scanline for no change in slope

- Coherence between scanlines tells us that
    - Edges that intersect scanline y are likely to intersect y + 1
    - X changes predictably from scanline y to y + 1



We have 2 data structures: Edge Table and Active Edge Table

- Traverse Edges to construct an Edge Table
    1. Eliminate horizontal edges
    2. Add edge to linked-list for the scan line corresponding to the lower vertex.

Store the following:

- y_upper: last scanline to consider
- x_lower: starting x coordinate for edge
- 1/m: for incrementing x; compute

- Construct Active Edge Table during scan conversion. AEL is a linked list of active edges on the current scanline, y. Each active edge line has the following information
    - y_upper: last scanline to consider
    - x_lower: edge's intersection with current y
    - 1/*m*: x increment

The active edges are kept sorted by x

Algorithm:

1. Set y to the smallest y coordinate that has an entry in the ET; i.e, y for the first nonempty bucket.
2. Initialize the AET to be empty.
3. Repeat until the AET and ET are empty:
   3.1 Move from ET bucket y to the AET those edges whose y_min = y (entering edges).
   3.2 Remove from the AET those entries for which y = y_max (edges not involved in the next scanline), the sort the AET on x (made easier because ET is presorted).

   3.3 Fill in desired pixel values on scanline y by using pairs of x coordinates from AET.
   3.4 Increment y by 1 (to the coordinate of the next scanline).
   3.5 For each nonvertical edge remaining in the AET, update x for the new y.

CODE:

```
#include<stdio.h>
#include<GL/glut.h>
#include<stdlib.h>
```

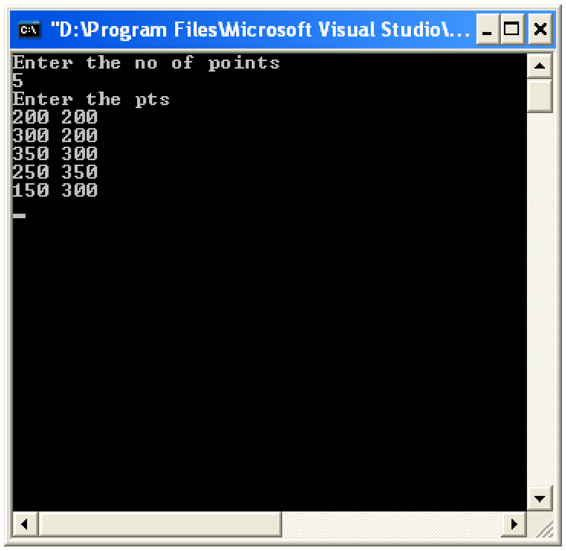
```
#define WINDOW_HEIGHT 500
/*The edge data structure*/
typedef struct tEdge {
int yUpper;
float xIntersect, dxPerScan;
struct tEdge * next;
} Edge;
typedef struct tdcPt {
int x;
int y;
} dcPt;
int cnt;
dcPt pts[60];
/* Inserts edge into list in order of increasing xIntersect field. */
void insertEdge (Edge * list, Edge * edge)
{
Edge * p, * q = list;
p = q->next;
while (p != NULL)
{
if (edge->xIntersect < p->xIntersect)
p = NULL;
else
{
q = p;
p = p->next;
}
}
edge->next = q->next;
q->next = edge;
}



void makeEdgeRec(dcPt lower, dcPt upper, int yComp, Edge * edge, Edge * edges[])
{
//Edge *q;
edge->dxPerScan =(float) (upper.x - lower.x) / (upper.y - lower.y);
edge->xIntersect = lower.x;
if (upper.y < yComp)
edge->yUpper = upper.y - 1;
else
edge->yUpper = upper.y;
insertEdge (edges[lower.y], edge);
/*checking the values inserted into edge records uncomment if you want to check
q=edges[lower.y]->next;
while(q!=NULL)
```

```
{
printf("xi=%f\n",q->xIntersect);
q=q->next;
}*/
}

/* For an index, return y-coordinate of next nonhorizontal line */
int yNext (int k, int cnt, dcPt * pts)
{
int j;
if ((k+1) > (cnt-1))
j = 0;
else
j = k + 1;
while (pts[k].y == pts[j].y)
if ((j+1) > (cnt-1))
j = 0;
else
j++;
return (pts[j].y);
}
void buildEdgeList (int cnt, dcPt * pts, Edge * edges[])
{
Edge * edge;
dcPt v1, v2;
int i, yPrev = pts[cnt - 2].y;
v1.x = pts[cnt-1].x; v1.y = pts[cnt-1].y;

for (i=0; i<cnt; i++) {
v2 = pts[i];
if (v1.y != v2.y) { /* nonhorizontal line */
edge = (Edge *) malloc (sizeof (Edge));
if (v1.y < v2.y) /* up-going edge */
makeEdgeRec (v1, v2, yNext (i, cnt, pts), edge, edges);
else /* down-going edge */
makeEdgeRec (v2, v1, yPrev, edge, edges);
}
yPrev = v1.y;
v1 = v2;
}
}
void buildActiveList (int scan, Edge * active, Edge * edges[])
{
Edge * p, * q;
p = edges[scan]->next;
while (p) {
q = p->next;
insertEdge (active, p);
p = q;
}
}
void fillScan (int scan, Edge * active)
{
Edge * p1, * p2;
int i;
p1 = active->next;
while (p1)
{
p2 = p1->next;
glColor3f(0.0,1.0,0.0);
glBegin(GL_POINTS);
for (i=p1->xIntersect; i<p2->xIntersect; i++)
glVertex2i((int) i, scan);
glEnd();
p1 = p2->next;
}
}
void deleteAfter (Edge * q)
{
Edge * p = q->next;
q->next = p->next;
free (p);
}

/* Delete completed edges. Update 'xIntersect' field for others */
void updateActiveList (int scan, Edge * active)
{
Edge * q = active, * p = active->next;
while (p)
if (scan >= p->yUpper)
{
p = p->next;
deleteAfter (q);
}
else
{
p->xIntersect = p->xIntersect + p->dxPerScan;/*x=x+1/m*/
q = p;
p = p->next;
}
}

void resortActiveList (Edge * active)
{
```

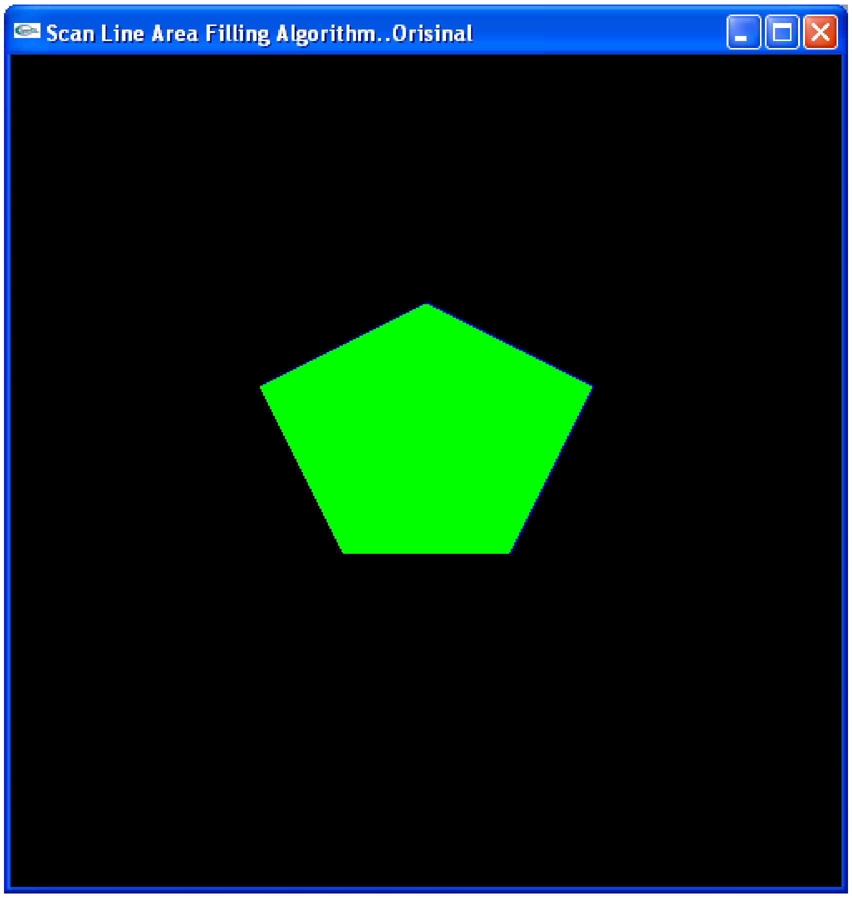
```
Edge * q, * p = active->next;
active->next = NULL;
while (p)
{
q = p->next;
insertEdge (active, p);
p = q;
}
}
void scanFill (int cnt, dcPt * pts)
{
Edge * edges[WINDOW_HEIGHT], * active;
int i, scan;
for (i=0; i<WINDOW_HEIGHT; i++)
{
edges[i] = (Edge *) malloc (sizeof (Edge));
edges[i]->next = NULL;
}

buildEdgeList (cnt, pts, edges);
active = (Edge *) malloc (sizeof (Edge));
active->next = NULL;
for (scan=0; scan<WINDOW_HEIGHT; scan++)
{
buildActiveList (scan, active, edges);
if (active->next)
{
fillScan (scan, active);
updateActiveList (scan, active);
resortActiveList (active);
}
}
/* Free edge records that have been malloc'ed ... */
free(active);
}

void display()
{
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(0.0,0.0,1.0);
  glBegin(GL_LINE_LOOP);
  for(int i=0;i<cnt; i++)
  {
          glVertex2i(pts[i].x,pts[i].y);
  }
  glEnd();
  scanFill(cnt,pts);
  glFlush();

}
void myinit()
{
glClearColor(0.0,0.0,0.0,1.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0,499.0,0.0,499.0);
glMatrixMode(GL_MODELVIEW);
}

void main(int argc, char **argv)
{
  printf("Enter the no of points\n");
  scanf("%d",&cnt);
  printf("Enter the pts\n");
  for(int i=0;i<cnt; i++)
          scanf("%d%d",&pts[i].x,&pts[i].y);
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
  glutInitWindowPosition(0,0);
  glutInitWindowSize(500,500);
  glutCreateWindow("Scan Line Area Filling Algorithm..Orisinal");
  myinit();
  glutDisplayFunc(display);
  glutMainLoop();
}
```

NOTE: Skeleton code(pseudo) obtained from
http://www.cs.uvm.edu/~snapp/teaching/CS274/lectures/scanlinefill.pdf

Output:



```
Enter the no of points
5
Enter the pts
200 200
300 200
350 300
250 350
150 300
_
```



PART B

Develop a suitable Graphics package to implement the skills learnt in the theory and the exercises indicated in PART A. Use the OpenGL.

Criteria for CG Project

Students per batch should be THREE or less.

One Three-Dimensional OpenGL Graphics Project using features from at least THREE CATEGORIES listed below:

Category I

ˆ Input and Interaction

ˆ Menus, Display Lists

Category II

ˆ Transformations

ˆ Camera Movement

Category III

ˆ Coloring

ˆ Texturing

ˆ Lighting/ Shading

Category IV

ˆ Animation

ˆ Hidden Surface Removal

1.1 OpenGL:

Introduction

OpenGL, or the Open Graphics Library, is a 3D graphics language developed by Sili-con Graphics. Before OpenGL was available, software developers had to write unique 3D graphics code for each operating system platform as well as dierent graphics hard-ware. However, with OpenGL, developers can create graphics and special eects that will appear nearly identical on any operating system and any hardware that supports OpenGL. This makes it much easier for developers of 3D games and programs to port their software to multiple platforms.

When programmers write OpenGL code, they specify a set of commands. Each command executes a drawing action or creates a special eect. Using hundreds or even thousands of these OpenGL commands, programmers can create 3D worlds which can include special eects such as texture mapping, transparency (alpha blending), hidden surface removal, antialiasing, fog, and lighting eects. An unlimited amount of viewing and modeling transformations can be applied to the OpenGL objects, giving developers an innite amount of possibilities.

GLUT gives you the ability to create a window, handle input and render to the screen without being Operating System dependent.

The rst things you will need are the OpenGL and GLUT header les and libraries for your current Operating System.

Once you have them setup on your system correctly, open your rst c++ le and include them at the start of your le like so:

```
#include <GL/gl.h> //include the gl header le #include
<GL/glut.h> //include the GLUT header le
```

Now, just double check that your system is setup correctly, and try compiling your current le.

If you get no errors, you can proceed. If you have any errors, try your best to x them. Once you are ready to move onto the next step, create a main() method in your current le.

Inside this is where all of your main GLUT calls will go.

The rst call we are going to make will initialize GLUT and is done like so:

```
glutInit(&argc, argv); //initialize the program.
```

Keep in mind for this, that argc and argv are passed as parameters to your main method. You can see how to do this below.

```
int main(int argc,char **argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(500,500);
glutCreateWindow("");
glutDisplayFunc(display);
myinit();
glutMainLoop();
}
```

Once we have GLUT initialized, we need to tell GLUT how we want to draw. There are several parameters we can pass here, but we are going to stick the with most basic GLUT_SINGLE, which will give use a single buered window.

```
glutInitDisplayMode(GLUT_SINGLE);//set up a basic display buer (only singular for now)
```

The next two methods we are going to use, simply set the size and position of the GLUT window on our screen:

```
glutInitWindowSize (500, 500); //set whe width and height of the window
glutInitWindowPosition (100, 100); //set the position of the window
```

And then we give our window a caption/title, and create it.

```
glutCreateWindow ("A basic OpenGL Window"); //set the caption for the window
```

We now have a window of the size and position that we want. But we need to be able to draw to it. We do this, by telling GLUT which method will be our main drawing method. In this case, it is a void method called display()

```
glutDisplayFunc(display); //call the display function to draw our world
```

Finally, we tell GLUT to start our program. It does this by executing a loop that will continue until the program ends.

glutMainLoop(); //initialize the OpenGL loop cycle

So thus far, we have a window. But the display method that I mentioned is needed. Lets take a look at this and dissect it.

```
void display(void)
{
glClearColor (0.0,0.0,0.0,1.0); //clear the color of the window
glClear (GL_COLOR_BUFFER_BIT); //Clear teh Color Buer (more buers later on)

glLoadIdentity(); //load the Identity Matrix
gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0); //set the view
glFlush(); //ush it all to the screen }
```

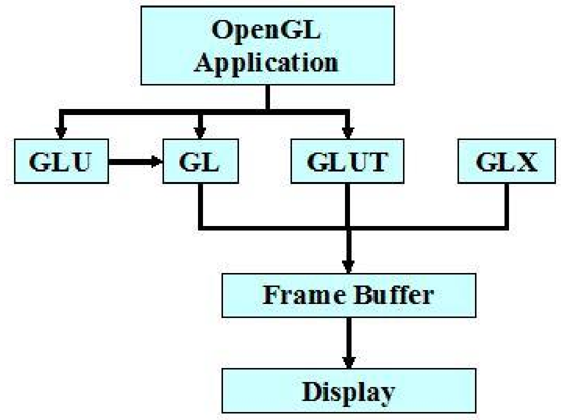
The rst method, glClearColor will set the background color of our window. In this example, we are setting the background to black (RGB 0, 0, 0). The 1.0 value on the end is an alpha value and makes no dierence at this stage as we don't have alpha enabled.

The next thing we want to do, is erase everything currently stored by OpenGL. We need to do this at the start of the method, as the method keeps looping over itself and if we draw something, we need to erase it before we draw our next frame. If we don't do this, we can end up with a big mess inside our buffer where frames have been drawn over each other.

The third method, glLoadIdentity resets our model view matrix to the identity matrix, so that our drawing transformation matrix is reset. From then, we will set the 'camera' for our scene. I am placing it 5 units back into the user so that anything we draw at 0,0,0 will be seen infront of us. The final thing we need to do is then flush the current buer to the screen so we can see it. This can be done with glFlush() as we are only using a single buer.

OpenGL is an API. OpenGL is nothing more than a set of functions you call from your program (think of as collection of .h les.

OpenGL Libraries



OpenGL Hierarchy: Several levels of abstraction are provided GL

- Lowest level: vertex, matrix manipulation
- glVertex3f(point.x, point.y, point.z)

GLU

- ˆ Helper functions for shapes, transformations
- ˆ gluPerspective(fovy, aspect, near, far)

1.2 Introduction to frequently used OpenGL commands

glutInit

ˆ glutInit is used to initialize the GLUT library.

Usage
void glutInit(int *argcp, char **argv);
argcp

ˆ A pointer to the program's unmodied argc variable from main. Upon return, the value pointed to by argcp will be updated, because glutInit extracts any command line options intended for the GLUT library.

argv

ˆ The program's unmodied argv variable from main. Like argcp, the data for argv will be updated because glutInit extracts any command line options understood by the GLUT library.

Description

ˆ glutInit will initialize the GLUT library. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized.

glutInitWindowPosition, glutInitWindowSize

ˆ glutInitWindowPosition and glutInitWindowSize set the initial window position and size respectively.

Usage
void glutInitWindowSize(int width, int height);
void glutInitWindowPosition(int x, int y); width

ˆ Width in pixels.

height

ˆ Height in pixels.

x

ˆ Window X location in pixels.

y

ˆ Window Y location in pixels.

Description

ˆ Windows created by glutCreateWindow will be requested to be created with the current initial window position and size.

glutInitDisplayMode

ˆ glutInitDisplayMode sets the initial display mode.

Usage

void glutInitDisplayMode(unsigned int mode);

mode

ˆ Display mode, normally the bitwise OR-ing of GLUT display mode bit masks. See values below:

GLUT_RGB
An alias for GLUT_RGBA.
GLUT_INDEX
Bit mask to select a color index mode window. This overrides GLUT RGBA if it is also specied.
GLUT_SINGLE
Bit mask to select a single buered window. This is the default if neither GLUT_DOUBLE or GLUT_SINGLE are specied.
GLUT_DOUBLE
Bit mask to select a double buered window. This overrides GLUT_SINGLE if it is also specied.
GLUT_DEPTH
Bit mask to select a window with a depth buer.
Description

ˆ The initial display mode is used when creating top-level windows.

glutMainLoop

ˆ glutMainLoop enters the GLUT event processing loop.

Usage

void glutMainLoop(void);

Description

ˆ glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

glutCreateWindow

ˆ glutCreateWindow creates a top-level window.

Usage

int glutCreateWindow(char *name);

name

ˆ ASCII character string for use as window name.

Description

- glutCreateWindow creates a top-level window. The name will be provided to the window system as the window's name. The intent is that the window system will label the window with the name.

glutPostRedisplay
glutPostRedisplay marks the current window as needing to be redisplayed.

Usage
void glutPostRedisplay(void);
Description

- Mark the normal plane of current window as needing to be redisplayed. The next iteration through glutMainLoop, the window's display callback will be called to redisplay the window's normal plane. Multiple calls to glutPostRedisplay before the next display callback opportunity generates only a single redisplay callback.

glutReshapeWindow

- glutReshapeWindow requests a change to the size of the current window.

Usage
void glutReshapeWindow(int width, int height);
width

- New width of window in pixels.

height

- New height of window in pixels.

Description

- glutReshapeWindow requests a change in the size of the current window. The width and height parameters are size extents in pixels. The width and height must be positive values. The requests by glutReshapeWindow are not processed immediately. The request is executed after returning to the main event loop. This allows multiple glutReshapeWindow, glutPositionWindow, and glut-FullScreen requests to the same window to be coalesced.

glutDisplayFunc

- glutDisplayFunc sets the display callback for the current window.

Usage
void glutDisplayFunc(void (*func)(void));
func

- The new display callback function.

Description

ˆ glutDisplayFunc sets the display callback for the current window. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called. Before the callback, the current window is set to the window needing to be redisplayed and (if no overlay display callback is registered) the layer in use is set to the normal plane. The display callback is called with no parameters. The entire normal plane region should be redisplayed in response to the callback (this includes ancillary buers if your program depends on their state).

glutReshapeFunc

ˆ glutReshapeFunc sets the reshape callback for the current window.

Usage

void glutReshapeFunc(void (*func)(int width, int height));

func

ˆ The new reshape callback function.

Description

ˆ glutReshapeFunc sets the reshape callback for the current window. The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's rst display callback after a window is created or whenever an overlay for the window is established. The width and height parameters of the callback specify the new window size in pixels. Before the callback, the current window is set to the window that has been reshaped.

glFlush

ˆ glFlush - force execution of GL commands in nite time

Description

ˆ Dierent GL implementations buer commands in several dierent locations, including network buers and the graphics accelerator itself. glFlush empties all of these buers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in nite time.

glMatrixMode

ˆ glMatrixMode - specify which matrix is the current matrix

Usage

void glMatrixMode(GLenum mode)

Parameters

ˆ mode- Species which matrix stack is the target for subsequent matrix operations. Three values are accepted: GL_MODELVIEW, GL_PROJECTION, and GL_TEXTURE. The default value is GL_MODELVIEW.

Description

ˆ glMatrixMode sets the current matrix mode. mode can assume one of three values: GL_MODELVIEW - Applies subsequent matrix operations to the mode matrix stack. GL_PROJECTION - Applies subsequent matrix operations to the projection matrix stack.

gluOrtho2D
Usage
gluOrtho2D(left, right, bottom, top)

ˆ Specifies the 2D region to be projected into the viewport. Any drawing outside the region will be automatically clipped away.

Viva questions :

1. Explain all the OpenGL functions used in this program?
2. What is the principle of Sierpinski gasket?
3. Difference between additive and subtractive color?
4. What is the Graphics Architecture used in OpenGL?
5. What is Rasterisation?
6. Explain sierpinski Gasket (using points).
7. Explain sierpinski Gasket (using polye)
8. Difference between 2D Tetrohedron and 3D Tetrahedron.
9. What do you mean by clipping?
10. How is Liang-Barsky clipping different from CohenSutherland Clipping Algorithm?
11. How do you set the color attributes in Opengl?
12. What is the command for clearscreen?
13. What is Event callback function?
14. Explain Liang-Barsky line clipping algorithm?
15. Explain window to viewpoint mapping?
16. What are vertex arrays?
17. How are the faces of the color cube modeled?
18. How do you consider the inward and outward pointing of the faces?
19. Explain the OpenGL function used to rotate the color cube?
20. What is the difference between 2D and 3D orthographic projection statements?
21. Explain the Inward and Outward pointing face?
22. Explain the data structure for object representation?
23. Explain the vertex list representation of a cube?
24. What is transformation?
25. Explain the OpenGL functions used for translation, rotation and scaling?
26. What is the order of transformation?
27. State the difference between modelview and projection?
28. What is Homogeneous-coordinate representation?
29. Define the rotation matrix and object matrix.
30. Explain the procedure to obtain the resultant matrix using rotation matrix and object matrix.
31. What is the principle of Cohen-Sutherland Algorithm?
32. State the advantages and disadvantages of Cohen-Sutherland Algorithm?

33. What is an outcode?
34. What is Synthetic Camera Model?
35. What are the Camera Specifications?
36. Explain the cases of outcodes in Cohen-Sutherland algorithm.
37. Explain the cohen-sutherland line clipping algorithm
38. Mention the difference between Liang-Barsky and Cohen-Sutherland line clipping algorithm.
39. Explain about gluLookAt(...), glFrustum(...), gluPerspective?
40. Explain the different types of projections?
41. Explain Z-buffer algorithm?
42. What is antialiasing?
43. What is Center Of Projection(COP), Direction Of Projection(DOP)?
44. What is midpoint circle drawing algorithm
45. How do you get the equation d+=2x+3, d+=2(x-y)+5
46. What is gluOrtho2D function
47. Explain plot pixel function
48. Why do we use GLFlush function in Display
49. Explain Specular, Diffuse and Translucent surfaces.
50. What is ambient light?
51. What is umbra, penumbra?
52. Explain Phong lighting model.
53. Explain glLightfv(...), glMaterialfv(...).
54. What is glutSolidCube function ? what are its Parameters
55. What are the parameters to glScale function
56. Explain Push & Pop matrix Functions
57. What is Materialfv function & its Parameters
58. Explain GLULookAt Function
59. Explain the keyboard and mouse events used in the program?
60. What is Hidden surface Removal? How do you achieve this in OpenGL?
61. What are the functions for creating Menus in OpenGL?
62. Explain about fonts in GLUT?
63. Explain about glutPostRedisplay()?
64. Explain how the cube is constructed
65. Explain rotate function
66. What is GLFrustum function what are its Parameters
67. What is viewport
68. What is glutKeyboard Function what are its Parameters
69. Explain scanline filling algorithm?
70. What is AspectRatio,Viewport?
71. How do you use timer?
72. What are the different frames in OpenGL?
73. What is fragment processing?
74. Explain Polygon Filling algorithm

75. What is slope
76. How the edges of polygon are detected
77. Why you use GL_PROJECTION in MatrixMode Function
78. What is dx & dy
79. What is maxx & maxy
80. What is glutPostRedisplay
81. Why do we use glutMainLoop function
82. What do you mean by GL_LINE_LOOP in GL_Begin function
83. Define Computer Graphics.
84. Explain any 3 uses of computer graphics applications.
85. What are the advantages of DDA algorithm?
86. What are the disadvantages of DDA algorithm?
87. Define Scan-line Polygon fill algorithm.
88. What are Inside-Outside tests?
89. Define Boundary-Fill algorithm.
90. Define Flood-Fill algorithm.
91. Define attribute parameter. Give examples.
92. What is a line width? What is the command used to draw the thickness of lines.
93. What are the three types of thick lines? Define.
94. What are the attribute commands for a line color?
95. What is color table? List the color codes.
96. What is a marker symbol and where it is used?
97. Discuss about inquiry functions.
98. Define translation and translation vector.
99. Define window and view port.
100. Define viewing transformation.
101. Give the equation for window to viewport transformation.
102. Define view up vector.
103. What is meant by clipping? Where it happens?
104. What is point clipping and what are its inequalities?
105. What is line clipping and what are their parametric representations?
106. How is translation applied?
107. What is referred to as rotation?
108. Write down the rotation equation and rotation matrix.
109. Write the matrix representation for scaling, translation and rotation.
110. Draw the block diagram for 2D viewing transformation pipeline.
111. Mention the equation for homogeneous transformation.
112. What is known as composition of matrix?
113. Write the composition transformation matrix for scaling, translation and Rotation.
114. Discuss about the general pivot point rotation?
115. Discuss about the general fixed point scaling.
116. Explain window, view port and window - to - view port transformation

117. Mention the three raster functions available in graphics packages.
118. What is known as region codes?
119. Why Cohen Sutherland line clipping is popular?
120. Mention the point clipping condition for the liang-barsky line clipping.
121. What is called as an exterior clipping?
122. How is the region code bit values determined?
123. Why liang-barsky line clipping is more efficient than Cohen Sutherland line Clipping?
124. Differentiate uniform and differential scaling
125. Explain soft fill procedures.
126. Explain the three primary color used in graphics
127. Explain in detail about color and grey scale levels?
128. Explain color and grey scale levels.
129. Explain the area fill attributes and character attributes.
130. Explain character attributes in detail.
131. Briefly discuss about basic transformations.
132. Explain matrix representations.
133. Discuss about composite transformations.
134. Explain about reflection and shear.
135. Explain the following transformation with the matrix representations.
Give suitable diagram for illustration translation .ii scaling.iii rotation.
136. How the rotation of an object about the pivot point is performed?
137. How window-to-viewport coordinate transformation happens.
138. Explain clipping with its operation in detail.
139. Explain Cohen- Sutherland line clipping.
140. Discuss the logical classifications of input devices.
141. Explain the details of 2d viewing transformation pipeline.
142. Explain point, line, curve, text, exterior clipping?
143. Discuss the properties of light.
144. Define chromaticity, complementary colors, color gamut and primary colors.
145. What is color model?
146. Define hue, saturation and value.
147. Explain XYZ color model.
148. Explain RGB color model.
149. Explain YIQ color model.
150. Explain CMY color model.
151. Explain HSV color model.
152. Give the procedure to convert HSV & RGB color model.
153. What is the use of chromaticity diagram?
154. What is illumination model?
155. What are the basic illumination models?
156. What is called as lightness?
157. Explain the conversion of CMY to RGB representation.

158. What is animation?
159. Define Morphing.
160. What are the steps involved in designing an animation sequence?
161. How to draw dots using OPENGL?
162. How to draw lines using OPENGL?
163. How to draw convex polygons using OPENGL?
164. What is the command used in OPENGL to clear the screen?
165. What are the various OPENGL data types?

Viva questions and answers

1. What is Computer Graphics?
   Answer: Computer graphics are graphics created using computers and, more generally, the representation and manipulation of image data by a computer.
2. What is OpenGL?
   Answer: OpenGL is the most extensively documented 3D graphics API (Application Pro-gram Interface) to date. It is used to create Graphics.
3. What is GLUT?
   Answer: The OpenGL Utility Toolkit (GLUT) is a library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system.
4. What are the applications of Computer Graphics?
   Answer: Gaming Industry, Animation Industry and Medical Image Processing Industries. The sum total of these industries is a Multi Billion Dollar Market. Jobs will continue to increase in this arena in the future.
5. Explain in breif 3D Sierpinski gasket?
   Answer: The Sierpinski triangle (also with the original orthography Sierpinski), also called the Sierpinski gasket or the Sierpinski Sieve, is a fractal named after the Polish mathemati-cian Waclaw Sierpinski who described it in 1915. Originally constructed as a curve, this is one of the basic examples of self-similar sets, i.e. it is a mathematically generated pattern that can be reproducible at any magnication or reduction.
6. What is Liang-Barsky line clipping algorithm?
   Answer: In computer graphics, the Liang-Barsky algorithm is a line clipping algorithm. The Liang-Barsky algorithm uses the parametric equation of a line and inequalities describing the range of the clipping box to determine the intersections between the line and the clipping box. With these intersections it knows which portion of the line should be drawn.
7. Explain in brief Cohen-Sutherland line-clipping algorithm?
   Answer: The Cohen-Sutherland line clipping algorithm quickly detects and dispenses with two common and trivial cases. To clip a line, we need to consider only its endpoints. If both endpoints of a line lie inside the window, the entire line lies inside the window. It is trivially accepted and needs no clipping. On the other hand, if both endpoints of a line

lie entirely to one side of the window, the line must lie entirely outside of the window. It is trivially rejected and needs to be neither clipped nor displayed.

8. Explain in brief scan-line area lling algorithm?

   Answer: The scanline ll algorithm is an ingenious way of lling in irregular polygons. The algorithm begins with a set of points. Each point is connected to the next, and the line between them is considered to be an edge of the polygon. The points of each edge are adjusted to ensure that the point with the smaller y value appears rst. Next, a data structure is created that contains a list of edges that begin on each scanline of the image. The program progresses from the rst scanline upward. For each line, any pixels that contain an intersection between this scanline and an edge of the polygon are lled in. Then, the algorithm progresses along the scanline, turning on when it reaches a polygon pixel and turning o when it reaches another one, all the way across the scanline.

9. Explain Midpoint Line algorithm

   Answer: The Midpoint line algorithm is an algorithm which determines which points in an n-dimensional raster should be plotted in order to form a close approximation to a straight line between two given points. It is commonly used to draw lines on a computer screen, as it uses only integer addition, subtraction and bit shifting, all of which are very cheap operations in standard computer architectures.

10. What is a Pixel?

    Answer: In digital imaging, a pixel (or picture element) is a single point in a raster image. The Pixel is the smallest addressable screen element; it is the smallest unit of picture which can be controlled. Each Pixel has its address. The address of Pixels corresponds to its coordinate. Pixels are normally arranged in a 2-dimensional grid, and are often represented using dots or squares.

11. What is Graphical User Interface?

    Answer: A graphical user interface (GUI) is a type of user interface item that allows people to interact with programs in more ways than typing such as computers; hand-held devices such as MP3 Players, Portable Media Players or Gaming devices; household appliances and o-ce equipment with images rather than text commands.

12. What is the general form of an OpenGL program?

    Answer: There are no hard and fast rules. The following pseudocode is generally recognized as good OpenGL form.

```
program_entrypoint
{
// Determine which depth or pixel format should be used.
// Create a window with the desired format.
// Create a rendering context and make it current with the window.
// Set up initial OpenGL state.
// Set up callback routines for window resize and window refresh.
}
handle_resize
{
glViewport(...);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
// Set projection transform with glOrtho, glFrustum, gluOrtho2D, gluPerspective, etc.
}
handle_refresh
{
glClear(...);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
// Set view transform with gluLookAt or equivalent
// For each object (i) in the scene that needs to be rendered:
// Push relevant stacks, e.g., glPushMatrix, glPushAttrib.
// Set OpenGL state specic to object (i).
// Set model transform for object (i) using glTranslatef, glScalef, glRotatef, and/or
equiv-alent.
// Issue rendering commands for object (i).
// Pop relevant stacks, (e.g., glPopMatrix, glPopAttrib.)
// End for loop.
// Swap buers.
}
```

13. What support for OpenGL does Open,Net,FreeBSD or Linux provide?

    Answer: The X Windows implementation, XFree86 4.0, includes support for OpenGL using Mesa or the OpenGL Sample Implementation. XFree86 is released under the XFree86 license. http://www.xfree86.org/

14. What is the AUX library?

    Answer: The AUX library was developed by SGI early in OpenGL's life to ease creation of small OpenGL demonstration programs. It's currently neither supported nor maintained. Developing OpenGL programs using AUX is strongly discouraged. Use the GLUT in-stead. It's more exible and powerful and is available on a wide range of platforms. Very important: Don't use AUX. Use GLUT instead.

15. How do I implement a zoom operation?

    Answer: A simple method for zooming is to use a uniform scale on the ModelView matrix. However, this often results in clipping by the zNear and zFar clipping planes if the model is scaled too large. A better method is to restrict the width and height of the view volume in the Projection matrix.

16. What are OpenGL coordinate units?

    Answer: Depending on the contents of your geometry database, it may be convenient for your application to treat one OpenGL coordinate unit as being equal to one millimeter or one parsec or anything in between (or larger or smaller). OpenGL also lets you specify your geometry with coordinates of diering values. For example, you may nd it convenient to model an airplane's controls in centimeters, its fuselage in meters, and a world to y around in kilometers. OpenGL's ModelView matrix can then scale these dierent coordinate sys-tems into the same eye coordinate space. It's the application's responsibility to ensure that the Projection and ModelView matrices are constructed to provide an image that keeps the viewer at an appropriate distance, with an appropriate eld of view, and keeps the zNear and zFar clipping planes at an appropriate range. An application that displays molecules in micron scale, for example, would probably not want to place the viewer at a distance of 10 feet with a 60 degree eld of view.

17. What is Microsoft Visual Studio?

    Answer: Microsoft Visual Studio is an integrated development environment (IDE) for devel-oping windows applications. It is the most popular IDE for developing windows applications or windows based software.

18. What does the .gl or .GL file format have to do with OpenGL?
    Answer: .gl files have nothing to do with OpenGL, but are sometimes confused with it. .gl is a file format for images, which has no relationship to OpenGL.

20. Who needs to license OpenGL? Who doesn't? Is OpenGL free software?

    Answer: Companies which will be creating or selling binaries of the OpenGL library will need to license OpenGL. Typical examples of licensees include hardware vendors, such as Digital Equipment, and IBM who would distribute OpenGL with the system software on their workstations or PCs. Also, some software vendors, such as Portable Graphics and Template Graphics, have a business in creating and distributing versions of OpenGL, and they need to license OpenGL. Applications developers do NOT need to license OpenGL. If a developer wants to use OpenGL that developer needs to obtain copies of a linkable OpenGL library for a particular machine. Those OpenGL libraries may be bundled in with the development and/or run-time options or may be purchased from a third-party software vendor, without licensing the source code or use of the OpenGLtrademark.

21. How do we make shadows in OpenGL? Answer: There are no individual routines to control neither shadows nor an OpenGL state for shadows. However, code can be written to render shadows.

22. What is the use of Glutinit?
    Answer: void glutInit(int *argcp, char **argv);

    glutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized.

23. Describe the usage of glutInitWindowSize and
    glutInitWindowPosition? Answer: void glutInitWindowSize(int width,
    int height);

    void glutInitWindowPosition(int x, int y);

    Windows created by glutCreateWindow will be requested to be created with the current initial window position and size. The intent of the initial window position and size values is to provide a suggestion to the window system for a window's initial size and position. The window system is not obligated to use this information. Therefore, GLUT programs should not assume the window was created at the specied size or position. A GLUT program should use the window's reshape callback to determine the true size of the window.

24. Describe the usage of glutMainLoop?
    Answer: void glutMainLoop(void);

glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

25. Define Computer Animation?

Computer animation is the art of creating moving images via the use of computers. It is a subfield of **computer graphics** and animation

26. Define Raster graphics?

Raster Graphics, which is the representation of images as an array of pixels, as it is typically used for the representation of photographic images.

27. Define Image?

An **image** (from Latin *imago*) is an artifact, usually two-dimensional (a **picture**), that has a similar appearance to some subject—usually a physical object or a person.

28. Define Rendering?

Rendering is the process of generating an image from a model, by means of computer programs.

29. Define Ray Tracing?

Ray tracing is a technique for generating an image by tracing the path of light through pixels in an image plane.

30. Define Projection?

Projection is a method of mapping points to a plane.

31. Define 3D Projection?

3D projection is a method of mapping three dimensional points to a two dimensional plane.

32. Define Texture Mapping?

Texture mapping is a method for adding detail, surface texture, or color to a computer-generated graphic or 3D model.

33. Define Vector Graphics?

Vector graphics formats are complementary to raster graphics, which is the representation of images as an array of pixels, as it is typically used for the representation of photographic images.

34. Define Pinhole camera model?

The **pinhole camera model** describes the mathematical relationship between the coordinates of a 3D point and its projection onto the image plane of an *ideal* pinhole camera, where the camera aperture is described as a point and no lenses are used to focus light.

35. Define digital image?

A **digital image** is a representation of a two-dimensional image using ones and zeros (binary). Without qualifications, the term "digital image" usually refers to raster images.

36. Define Grayscale?

A **grayscale** or **greyscale** digital image is an image in which the value of each pixel is a single sample, that is, it carries only intensity information.

37. Define Pinhole Camera?

A **pinhole camera** is a very simple camera with no lens and a single very small aperture. Simply explained, it is a light-proof box with a small hole in one side.

38. Define CCD?

A **charge-coupled device** (**CCD**) is an analog shift register that enables the transportation of analog signals (electric charges) through successive stages (capacitors), controlled by a clock signa.

39. Define Image Resolution?

Image resolution describes the detail an image holds. The term applies equally to digital images, film images, and other types of images. Higher resolution means more image detail.

40. Define Wavelength?

Wavelength is the distance between repeating units of a propagating wave of a given frequency.

41. What is an Electromagnetic Spectrum?

The **electromagnetic (EM) spectrum** is the range of all possible electromagnetic radiation frequencies. The "electromagnetic spectrum" (usually just *spectrum*) of an object is the characteristic distribution of electromagnetic radiation from that particular object.

42. Define Halftone?

Halftone is the reprographic technique that simulates continuous tone imagery through the use of dots, varying either in size or in spacing. 'Halftone' can also be used to refer specifically to the image that is produced by this process.

43. Define WCS?

World Coordinate Systems. While every point in the FITS data array can be located in the coordinate system determined by the array axes. **World Coordinate** Systems (WCS) are any **coordinate** systems that describe the physical **coordinate** associated with a data array, such as sky **coordinates.**

44. What are the benefits of OpenGL for hardware and software developers?

Industry standard

Reliable and portable.

Evolving.

Scalable.

Easy to use.

Well-documented.

45. What is the GLUT Toolkit?

GLUT is a portable toolkit which performs window and event operations to support OpenGL rendering

46. How does the camera work in OpenGL?

As far as OpenGL is concerned, there is no camera. More specifically, the camera is always located at the eye space coordinate (0.0, 0.0, 0.0). To give the appearance of moving the camera, your OpenGL application must move the scene with the inverse of the camera transformation by placing it on the MODELVIEW matrix. This is commonly referred to as the viewing transformation. In practice this is mathematically equivalent to a camera transformation but more efficient because model transformations and camera transformations are concatenated to a single matrix. As a result though, certain operations must be performed when the camera and only the camera is on the MODELVIEW matrix. For example to position a light source in world space it most be positioned while the viewing transformation and only the viewing transformation is applied to the MODELVIEW matrix.

47. How do I implement a zoom operation?

A simple method for zooming is to use a uniform scale on the ModelView matrix. However, this often results in clipping by the zNear and zFar clipping planes if the model is scaled too large. A better method is to restrict the width and height of the view volume in the Projection matrix. For example, your program might maintain a zoom factor based on user input, which is a floating-point number. When set to a value of 1.0, no zooming takes place. Larger values result in greater zooming or a more restricted field of view, while smaller values cause the opposite to occur.

52. How do I get a specified point (XYZ) to appear at the center of the scene?

gluLookAt() is the easiest way to do this. Simply set the X, Y, and Z values of your point as the fourth, fifth, and sixth parameters to gluLookAt ().

53. Define Instruction Pipeline?

An **instruction pipeline** is a technique used in the design of computers and other digital electronic devices to increase their instruction throughput (the number of instructions that can be executed in a unit of time).

54.Define Geometric Pipeline?

Geometric manipulation of modeling primitives, such as that performed by a **Geometry Pipeline**, is the first stage in computer graphics systems which perform image generation based on geometric models.

55.Define Perspective Projection?

Perspective (from Latin *perspicere*, to see through) in the graphic arts, such as drawing, is an approximate representation, on a flat surface (such as paper), of an image as it is perceived by the eye. The two most characteristic features of perspective are that objects are drawn.

CONTENT BEYOND SYLLABUS

Program to create a house like figure and rotate it about a given fixed point using OpenGL functions.

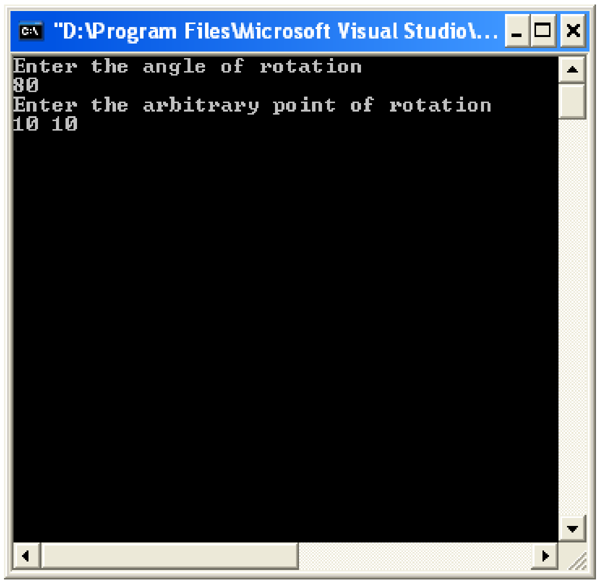
Program Code:

```
#include<GL/glut.h>
#include<math.h>
#include<stdio.h>
#define pi 3.142
GLfloat house[3][9]={{10.0,10.0,35.0,60.0,60.0,20.0,20.0,50.0,50.0},
                                  {10.0,35.0,50.0,35.0,10.0,10.0,20.0,20.0,10.0},
                                  {1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0}};
GLfloat rot_mat[3][3]={{0.0},{0.0},{0.0}};
GLfloat result[3][9]={{0},{0},{0}};
GLfloat h=10.0;
GLfloat k=10.0;
GLfloat theta=0.0;
float c=pi/180.0;
void multiply()
{
  int i,j,l;
  for(i=0;i<3;i++)
          for(j=0;j<9;j++)
          {
                  result[i][j]=0;
                  for(l=0;l<3;l++)
                          result[i][j]=result[i][j]+rot_mat[i][l]*house[l][j];
          }
}
void rotate()
{
  float thetar=theta*c;
  GLfloat m,n;
  m=h*(1-cos(thetar))+k*(sin(thetar));
  n=k*(1-cos(thetar))-h*(sin(thetar));
  rot_mat[0][0]=cos(thetar);
  rot_mat[0][1]=-sin(thetar);
  rot_mat[0][2]=m;
  rot_mat[1][0]=sin(thetar);
  rot_mat[1][1]=cos(thetar);
  rot_mat[1][2]=n;
  rot_mat[2][0]=0;
  rot_mat[2][1]=0;
  rot_mat[2][2]=1;
  multiply();
}
void drawhouse()
{
  glColor3f(0.0,0.0,1.0);
  glBegin(GL_LINE_LOOP);
  glVertex2f(house[0][0],house[1][0]);
  glVertex2f(house[0][1],house[1][1]);
  glVertex2f(house[0][3],house[1][3]);
```

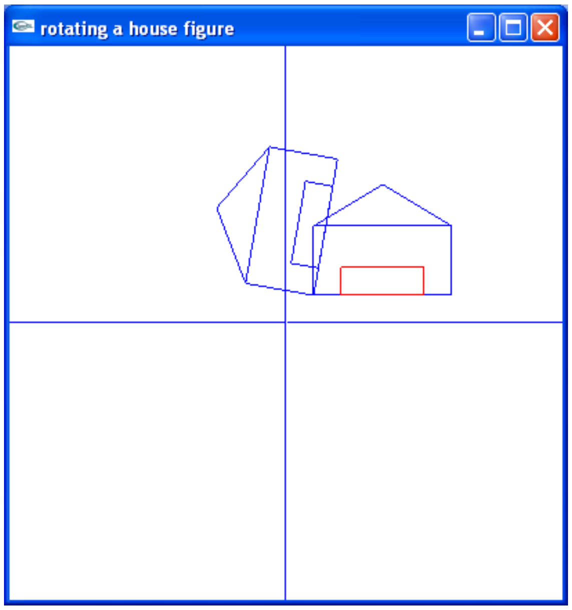
```
    glVertex2f(house[0][4],house[1][4]);
    glEnd();
    glColor3f(1.0,0.0,0.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(house[0][5],house[1][5]);
    glVertex2f(house[0][6],house[1][6]);
    glVertex2f(house[0][7],house[1][7]);
    glVertex2f(house[0][8],house[1][8]);
    glEnd();
    glColor3f(0.0,0.0,1.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(house[0][1],house[1][1]);
    glVertex2f(house[0][2],house[1][2]);
    glVertex2f(house[0][3],house[1][3]);
    glEnd();
}
void drawrotatehouse()
{
    glColor3f(0.0,0.0,1.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(result[0][0],result[1][0]);
    glVertex2f(result[0][1],result[1][1]);
    glVertex2f(result[0][3],result[1][3]);
    glVertex2f(result[0][4],result[1][4]);
    glEnd();
    glColor3f(0.0,0.0,1.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(result[0][5],result[1][5]);
    glVertex2f(result[0][6],result[1][6]);
    glVertex2f(result[0][7],result[1][7]);
    glVertex2f(result[0][8],result[1][8]);
    glEnd();
    glColor3f(0.0,0.0,1.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(result[0][1],result[1][1]);
    glVertex2f(result[0][2],result[1][2]);
    glVertex2f(result[0][3],result[1][3]);
    glEnd();
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINES);
    glVertex2f(0.0,100.0);
    glVertex2f(0.0,-100.0);
    glVertex2f(100.0,0.0);
    glVertex2f(-100.0,0.0);
    glEnd();
    glColor3f(1.0,1.0,1.0);
    glBegin(GL_POINTS);
    glVertex2f(0.0,0.0);
    glVertex2f(0.0,0.0);
    glEnd();
    drawhouse();
    rotate();
    drawrotatehouse();
    glFlush();
```

```
}
void myinit()
{
  glClearColor(1.0,1.0,1.0,1.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(-100.0,100.0,-100.0,100.0);
  glMatrixMode(GL_MODELVIEW);
}
void main(int argc, char **argv)
{
  printf("Enter the angle of rotation\n");
  scanf("%f",&theta);
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
  glutInitWindowSize(400,400);
  glutInitWindowPosition(0,0);
  glutCreateWindow("rotating a house figure");
  myinit();
  glutDisplayFunc(display);
  glutMainLoop();
}
```

Output:



```
Enter the angle of rotation
80
Enter the arbitrary point of rotation
10 10
```

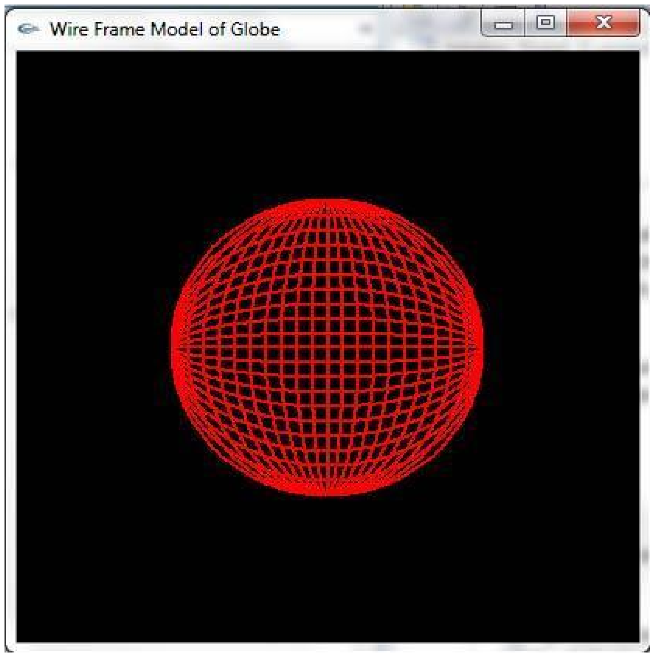


1.Write a Program to create a wire frame model of globe using equation of ellipse.

```
#include <GL/glut.h>
#include<math.h>
 #define PI 3.1416

      void setPixel(GLint, GLint);
      void ellipse(GLint, GLint, GLint, GLint, Lfloat);
      void display();
      void resize(int, int);

      void main(int argc, char**argv)
      {
            glutInit(&argc,argv);                        //initialize GLUT
            glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);                  //initialize display mode
            glutInitWindowSize(400,400);                 //set display-window width & height
            glutInitWindowPosition(50,50);               //set display-window upper-left position

            glutCreateWindow("Wire Frame Model of Globe");           //create display-window
            glutDisplayFunc(display);                //call graphics to be displayed on the window
            glutReshapeFunc(resize);                  //calls whenever frame buffer window is resized

            glutMainLoop();          //display everything and wait
}
    void resize(int w, int h)
    {
          //set projection paramaters
          glMatrixMode(GL_PROJECTION);
          glLoadIdentity();
          gluOrtho2D(0.0,w,0.0,h); glViewport(0.0,
          0.0, w, h);
    }

    void display()
    {
          glClear(GL_COLOR_BUFFER_BIT);
          glColor3f(1.0,0.0,0.0); glPointSize(2.0);

          GLint xc=200;             // setting the center of globe

          GLint yc=200;

          GLint ry=100; // setting the radius_accross_y-axis of ellipse as constant

          //change angle (0 – 360) and radius_accross_x-axis (0 to 100 in steps of 10) // and draw ellipses

          for(GLint theta =1;theta<=360;theta++) for(GLint
                  rx=100;rx>=0;rx-=10)
                          ellipse(xc, yc, rx, ry, theta);

          GLint rx=100;  //setting the radius_accross_x-axis of ellipse as constant
```

```
        //change angle (0 – 360) and radius_accross_y-axis (0 to 100 in steps of 10) // and draw ellipses

        for(GLint theta =1;theta<=360;theta++) for(GLint
                ry=100;ry>=0;ry-=10)
                        ellipse(xc, yc, rx, ry, theta);

        glFlush();
    }

    void ellipse(GLint xc, GLint yc, GLint rx, GLint ry, GLfloat theta)
    {
        GLint x = xc + rx *cos(theta* PI/180.0); GLint y = yc +
        ry * sin(theta* PI/180.0); setPixel(x,y);
    }

     void setPixel(GLint xCoordinate, GLint yCoordinate)
     {
          glBegin(GL_POINTS);
          glVertex2i(xCoordinate,yCoordinate); glEnd();
     }
```

OUTPUT:



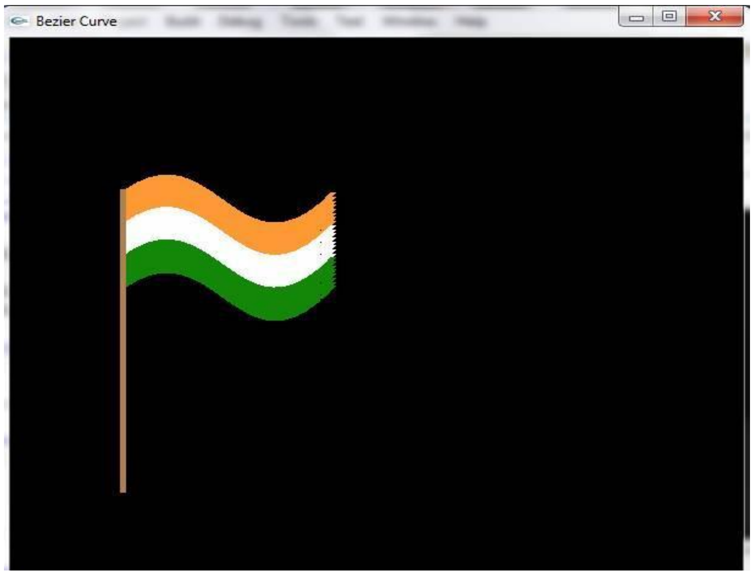
2.Write a program to animate a flag using Bezier curve algorithm.

```
#include<GL/glut.h>
#include<stdio.h>
#include<math.h>
#define PI 3.1416

    GLsizei winWidth = 600, winHeight = 600;
    GLfloat xwcMin = 0.0, xwcMax = 130.0;
    GLfloat ywcMin = 0.0, ywcMax = 130.0;

    typedef struct wcPt3D
    {
        GLfloat x, y, z;
    };

    void bino(GLint n, GLint *C)
    {
        GLint k, j;

        for(k=0;k<=n;k++)
        {
                C[k]=1;
   for(j=n;j>=k+1; j--)
                        C[k]*=j;

 for(j=n-k;j>=2;j--) C[k]/=j;

        }

    }

void computeBezPt(GLfloat u, wcPt3D *bezPt, GLint nCtrlPts, wcPt3D *ctrlPts, GLint *C)
{

        GLint k, n=nCtrlPts-1;
        GLfloat bezBlendFcn;

        bezPt ->x =bezPt ->y = bezPt->z=0.0;

        for(k=0; k< nCtrlPts; k++)
        {
                bezBlendFcn = C[k] * pow(u, k) * pow( 1-u,

                n-k); bezPt ->x += ctrlPts[k].x * bezBlendFcn;
                bezPt ->y += ctrlPts[k].y * bezBlendFcn; bezPt ->z +=
                ctrlPts[k].z * bezBlendFcn;
        }
}

    void bezier(wcPt3D *ctrlPts, GLint nCtrlPts, GLint nBezCurvePts)
    {
        wcPt3D bezCurvePt;
        GLfloat u;
    GLint *C, k;
    C= new GLint[nCtrlPts];
    bino(nCtrlPts-1, C);
  glBegin(GL_LINE_STRIP);
          for(k=0; k<=nBezCurvePts; k++)
          {
                  u=GLfloat(k)/GLfloat(nBezCurvePts); computeBezPt(u, &bezCurvePt,
                  nCtrlPts, ctrlPts, C); glVertex2f(bezCurvePt.x, bezCurvePt.y);
          }
  glEnd();
  delete[]C;
}
void displayFcn()
{
  GLint nCtrlPts = 4, nBezCurvePts =20; static float
  theta = 0;
  wcPt3D ctrlPts[4] = { {20,
          100, 0}, {30, 110,
          0}, {50, 90, 0}, {60,
          100, 0}};
  ctrlPts[1].x +=10*sin(theta * PI/180.0); ctrlPts[1].y +=5*sin(theta
  * PI/180.0);
  ctrlPts[2].x -= 10*sin((theta+30) * PI/180.0); ctrlPts[2].y -=
  10*sin((theta+30) * PI/180.0); ctrlPts[3].x-= 4*sin((theta) *
  PI/180.0); ctrlPts[3].y += sin((theta-30) * PI/180.0);
  theta+=0.1;
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 1.0, 1.0); glPointSize(5);
  glPushMatrix();
  glLineWidth(5);
  glColor3f(255/255, 153/255.0, 51/255.0); //Indian flag: Orange color code
  for(int i=0;i<8;i++)
  {
          glTranslatef(0, -0.8, 0);
          bezier(ctrlPts, nCtrlPts, nBezCurvePts);
  }


      glColor3f(1, 1, 1);                        //Indian flag: white color code
      for(int i=0;i<8;i++)
      {
              glTranslatef(0, -0.8, 0);
              bezier(ctrlPts, nCtrlPts, nBezCurvePts);
      }

      glColor3f(19/255.0, 136/255.0, 8/255.0); //Indian flag: green color code for(int i=0;i<8;i++)

      {
              glTranslatef(0, -0.8, 0);
              bezier(ctrlPts, nCtrlPts, nBezCurvePts);
      }
      glPopMatrix();

      glColor3f(0.7, 0.5,0.3);
      glLineWidth(5);
      glBegin(GL_LINES);
```

```
                    glVertex2f(20,100);
                    glVertex2f(20,40);

            glEnd();

            glFlush();
            glutPostRedisplay();
            glutSwapBuffers();
        }
        void winReshapeFun(GLint newWidth, GLint newHeight)
        {
            glViewport(0, 0, newWidth, newHeight);
            glMatrixMode(GL_PROJECTION);
            glLoadIdentity();
            gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);
            glClear(GL_COLOR_BUFFER_BIT);
        }
        void main(int argc, char **argv)
        {
            glutInit(&argc, argv);
            glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
            glutInitWindowPosition(50, 50);
            glutInitWindowSize(winWidth, winHeight);
            glutCreateWindow("Bezier  Curve");

            glutDisplayFunc(displayFcn);
            glutReshapeFunc(winReshapeFun);
         glutMainLoop();
     }
```

OUTPUT:

Once you execute the above program, you will find a waving flag. The below given figure is a still-image of the same.


Bezier Curve