Computer Graphics and Visualization Laboratory
Subject Code: 06CSL67

1.INTRODUCTION

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

OpenGL (**Open 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.

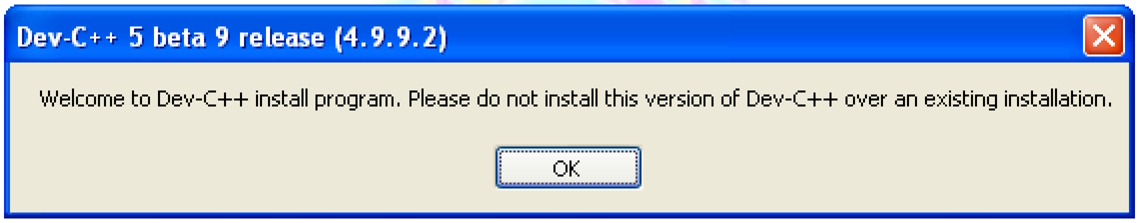
2.1 On Windows:

2.1.1 Working with Visual C++ →

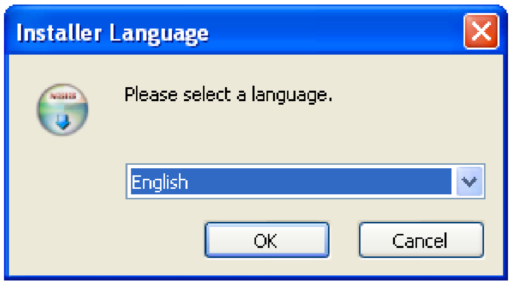
- Get glut.h, glut32.lib and glut32.dll from web.
- Copy and paste the glut.h file in the C:\ProgramFiles\MicrosoftVisual studio\VC98\Include\GL folder.
- Copy and paste the glut32.lib file in the C:\ProgramFiles\MicrosoftVisual studio\VC98\Lib folder.
- Copy and paste the glut32.dll file in the C:\WINDOWS\system32
- 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++ →

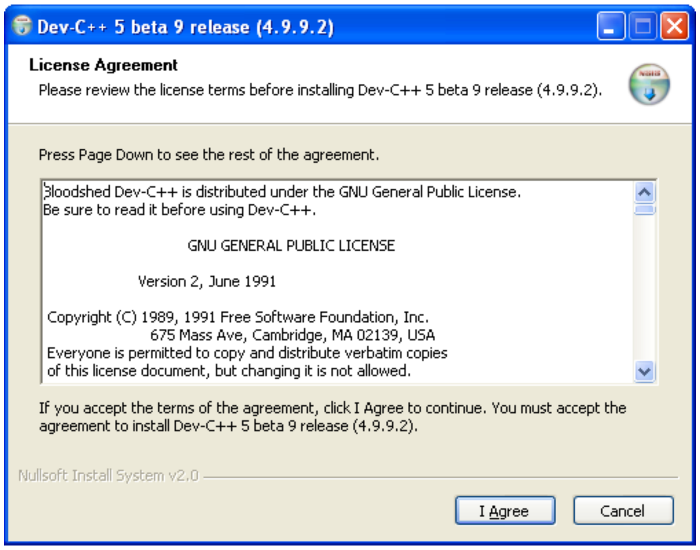
Find Dev C++ 's Beta version at this site>>
http://bloodshed-dev-c.en.softonic.com/
Click on the dev C++ set up icon the download will begin.
Here is the step by step guide to installation of Dev.



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



Please select a language.
English


Dev-C++ 5 beta 9 release (4.9.9.2)
I Agree
Cancel
Nullsoft Install System v2.0
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++.

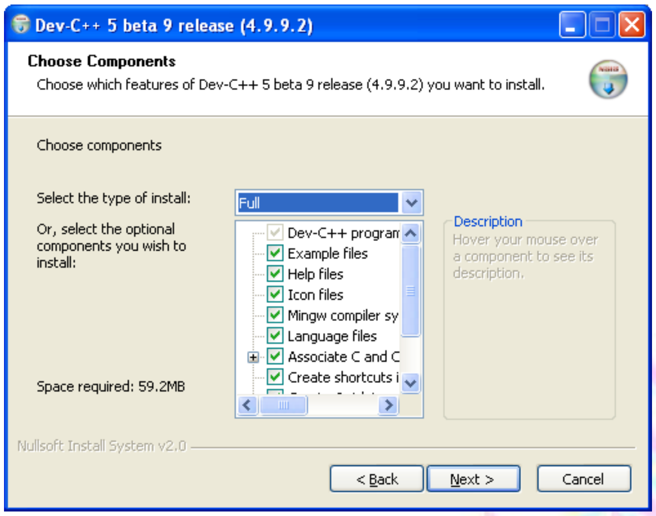
GNU GENERAL PUBLIC LICENSE

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).

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

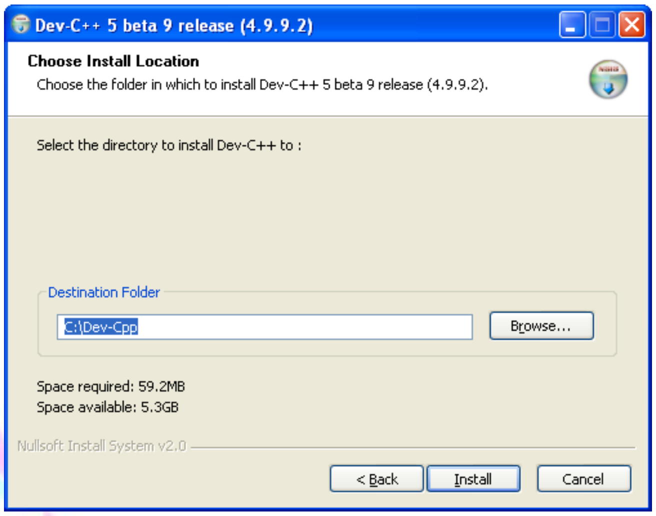
Or, select the optional components you wish to install:

- 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

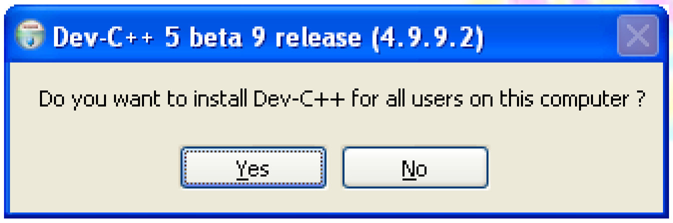
Choose the folder in which to install Dev-C++ 5 beta 9 release (4.9.9.2).

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

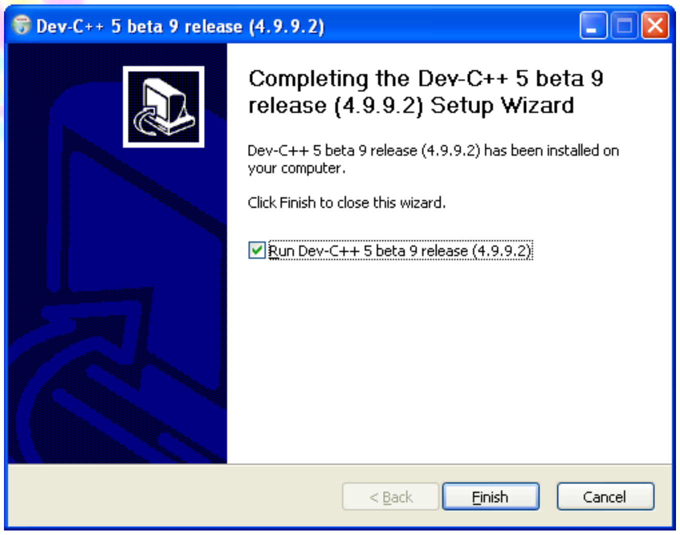
Destination Folder: C:\Dev-Cpp

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 ?

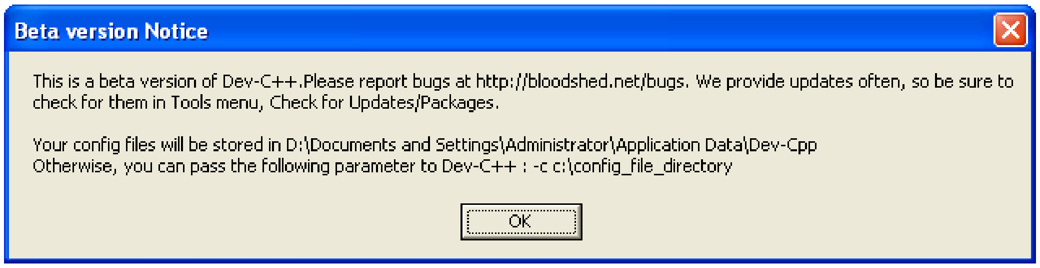

Dev-C++ 5 beta 9 release (4.9.9.2)

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.

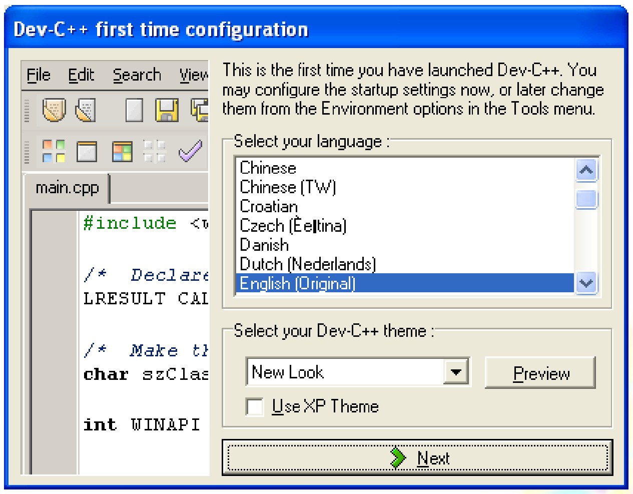
[✓] Run Dev-C++ 5 beta 9 release (4.9.9.2)


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



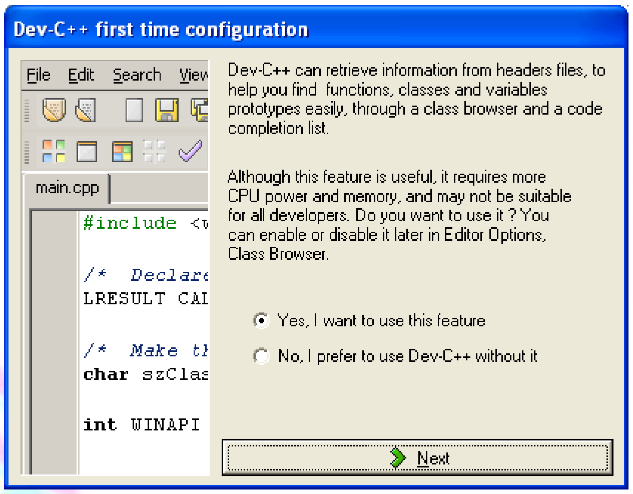
This is the first time you have launched Dev-C++. You may configure the startup settings now, or later change them from the Environment options in the Tools menu.

Select your language : Chinese / Chinese (TW) / Croatian / Czech (Èeltina) / Danish / Dutch (Nederlands) / [✓ English (Original)]

Select your Dev-C++ theme : New Look

Use XP Theme ☐

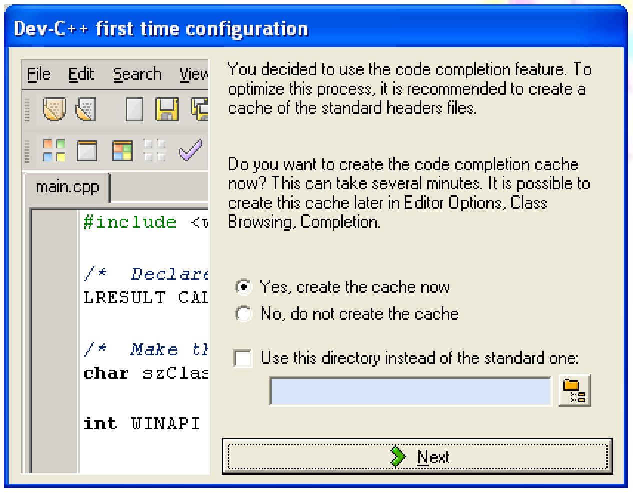
```
#include <w
/* Declare
LRESULT CAL
/* Make th
char szClas
int WINAPI
```



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

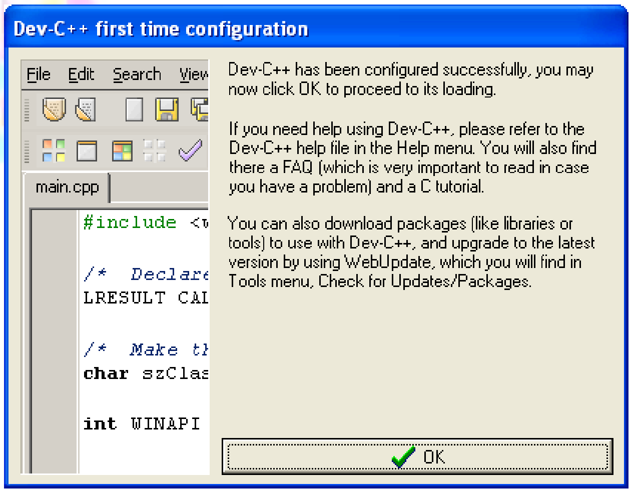


You decided to use the code completion feature. To optimize this process, it is recommended to create a cache of the standard headers files.

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:

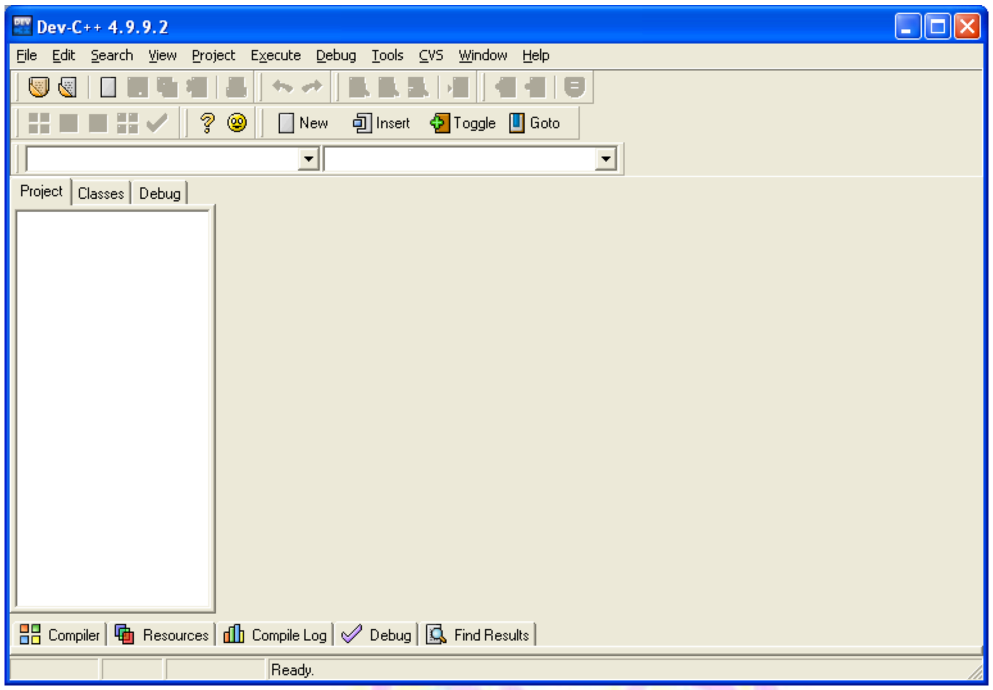


Dev-C++ has been configured successfully, you may now click OK to proceed to its loading.

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.

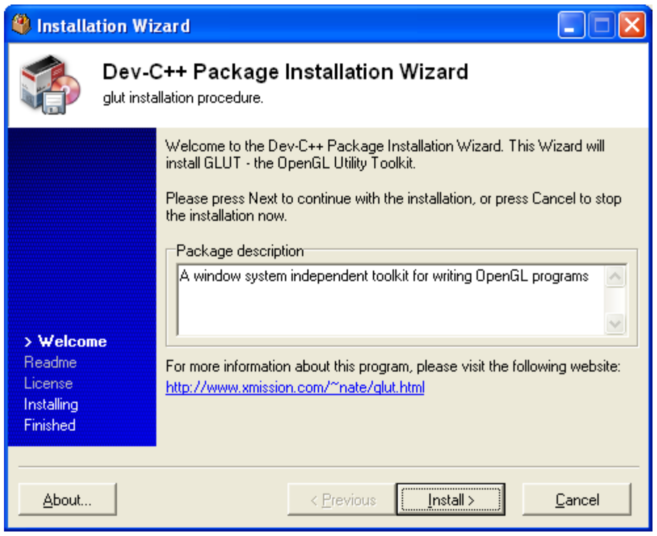
You can also download packages (like libraries or tools) to use with Dev-C++, and upgrade to the latest version by using WebUpdate, which you will find in Tools menu, Check for Updates/Packages.

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!


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Compiler Resources Compile Log Debug Find Results
Ready.

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 .



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

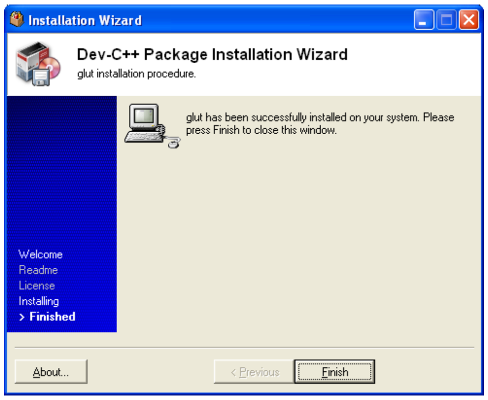
Welcome to the Dev-C++ Package Installation Wizard. This Wizard will install GLUT - the OpenGL Utility Toolkit.

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

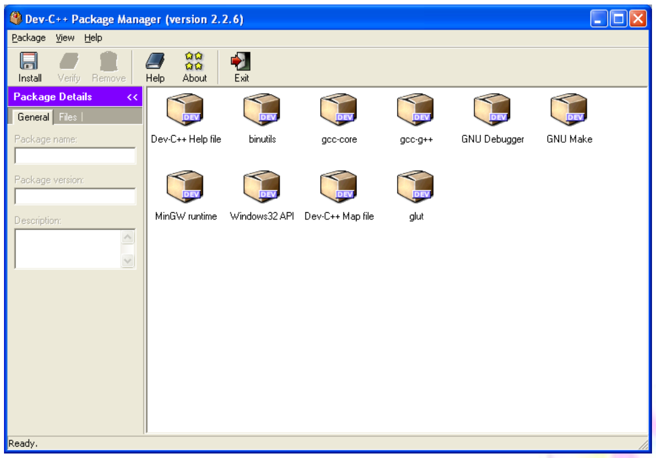
For more information about this program, please visit the following website:
http://www.xmission.com/~nate/glut.html

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


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.


Dev-C++ Package Manager (version 2.2.6)

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→

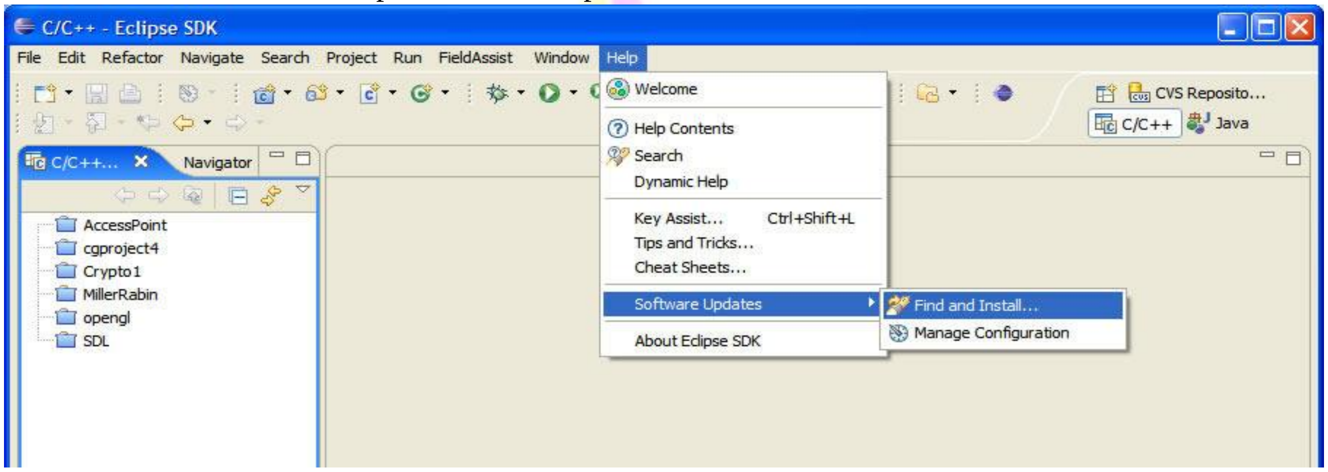
1. Download Eclipse SDK 3.2.2 or newer at Eclipse.org/downloads
2. Install MinGW and the gcc/g++ compiler

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 filesin step 4.

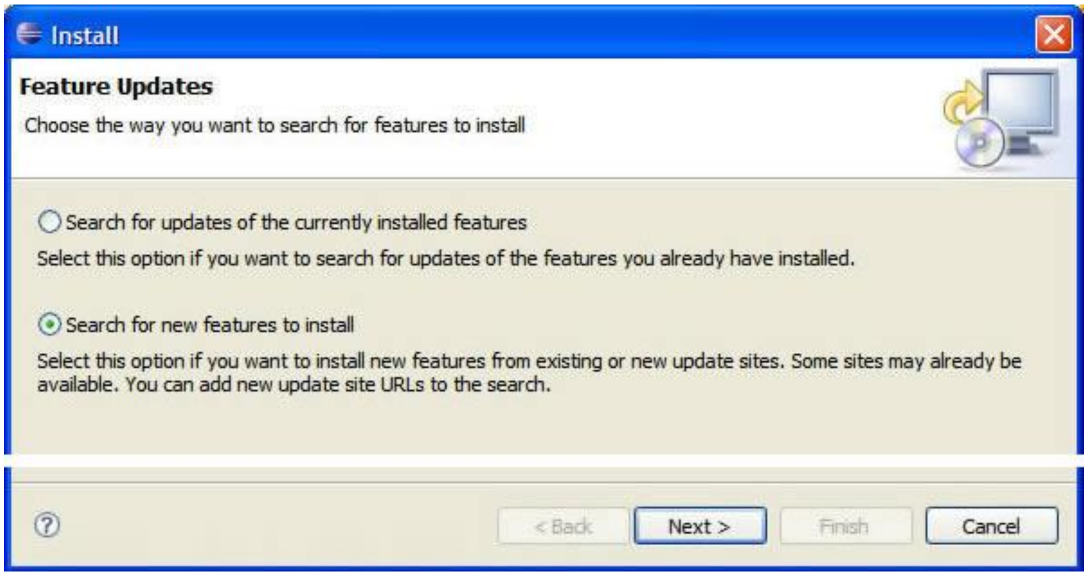
3. Install the C/C++ Development Tooling (CDT)

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 Install



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



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.

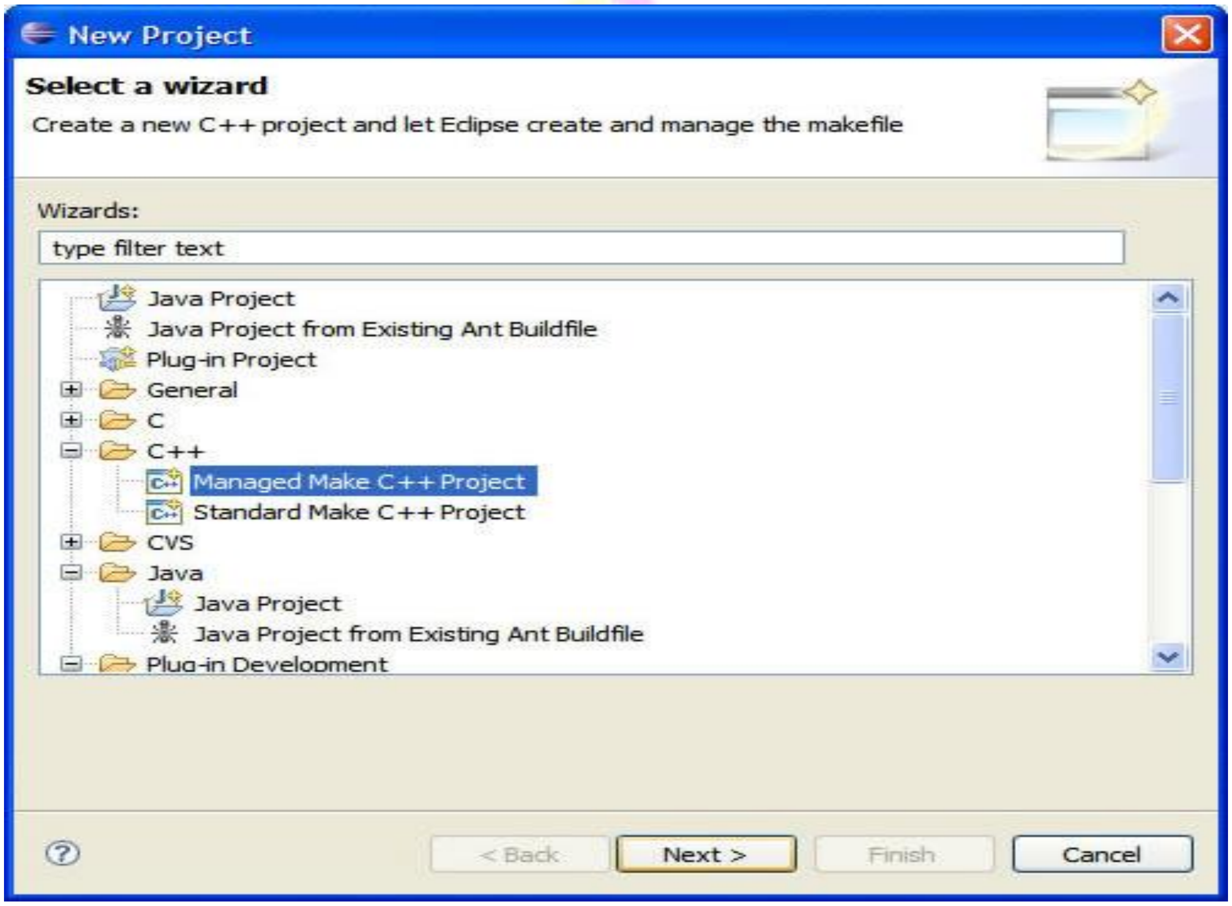
● Search for new features to install

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

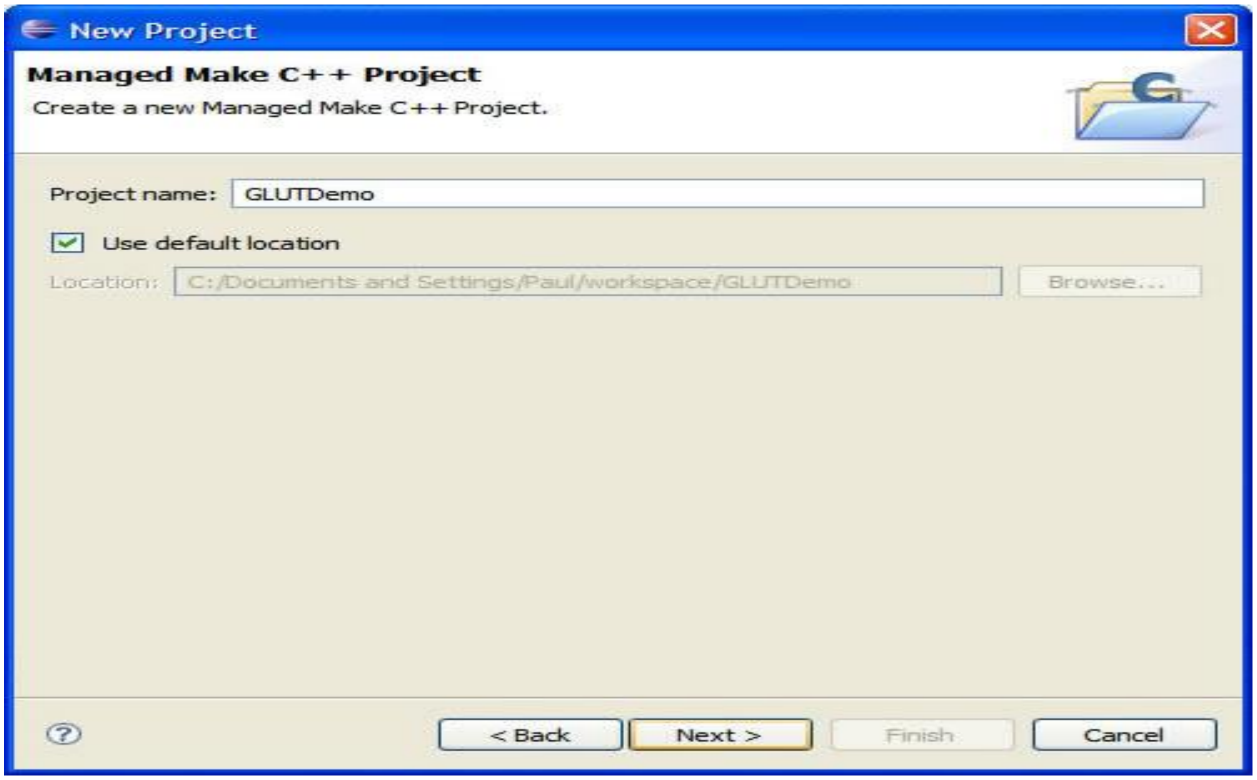
Create a new C++ project and let Eclipse create and manage the makefile

Wizards:

type filter text

Java Project
Java Project from Existing Ant Buildfile
Plug-in Project
General
C
C++
Managed Make C++ Project
Standard Make C++ Project
CVS
Java
Java Project
Java Project from Existing Ant Buildfile
Plug-in Development

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


New Project

Managed Make C++ Project

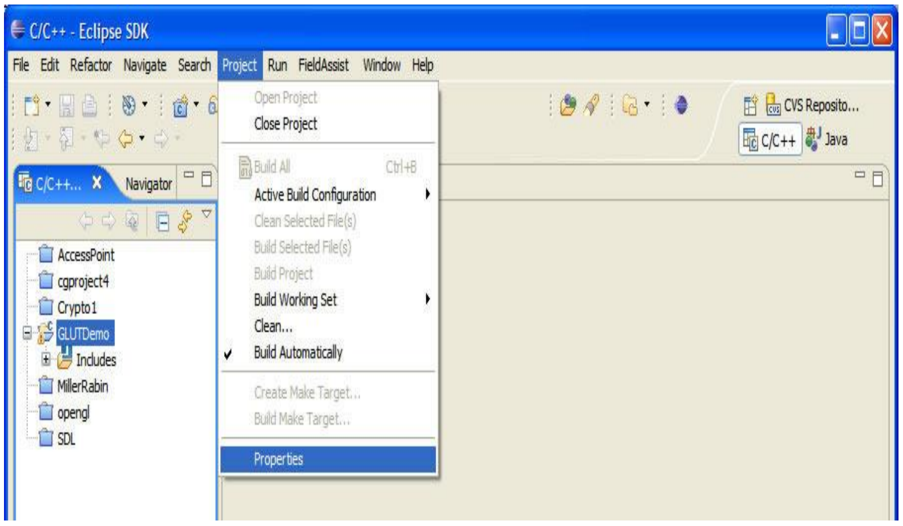
Create a new Managed Make C++ Project.

Project name: GLUTDemo

Use default location

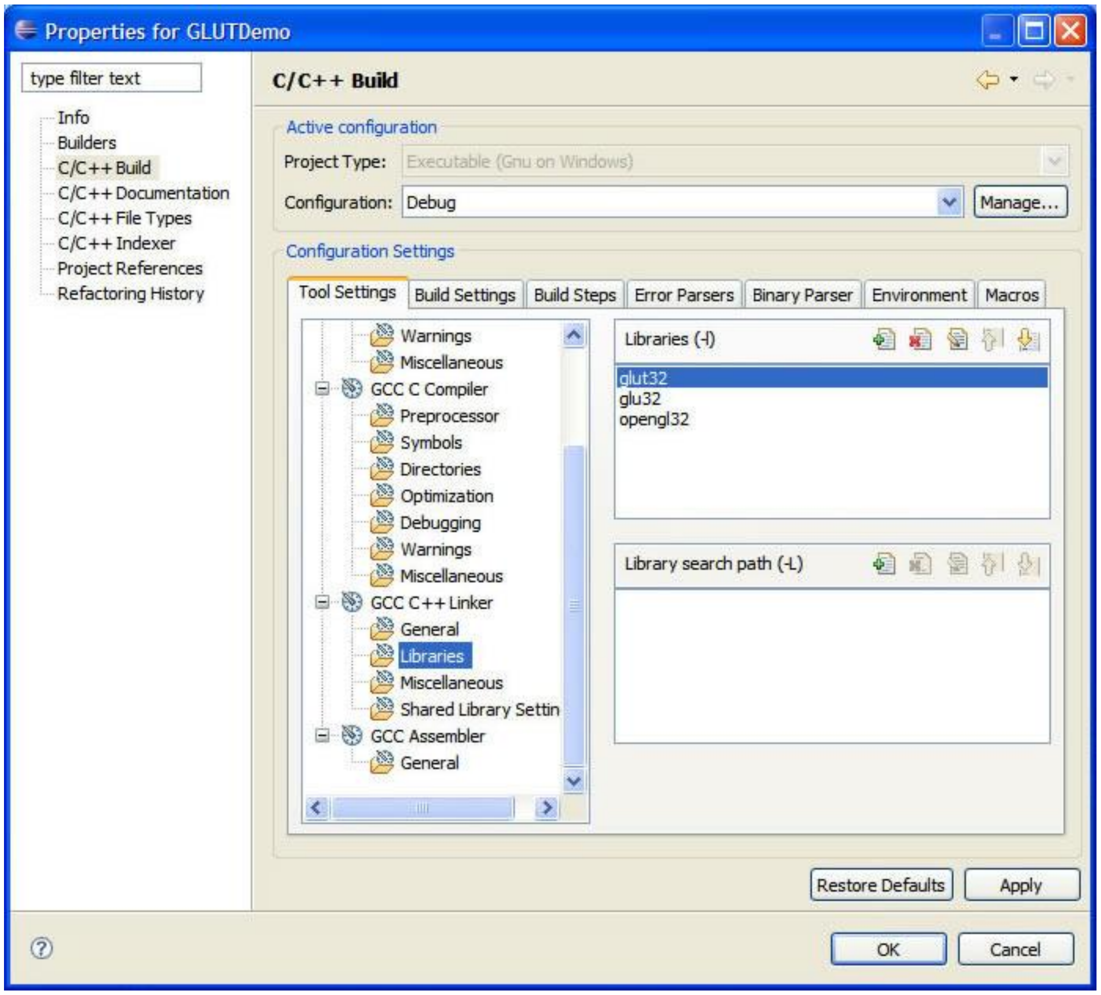
Location: C:/Documents and Settings/Paul/workspace/GLUTDemo

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

Library search path (-L)

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 Redhat 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:

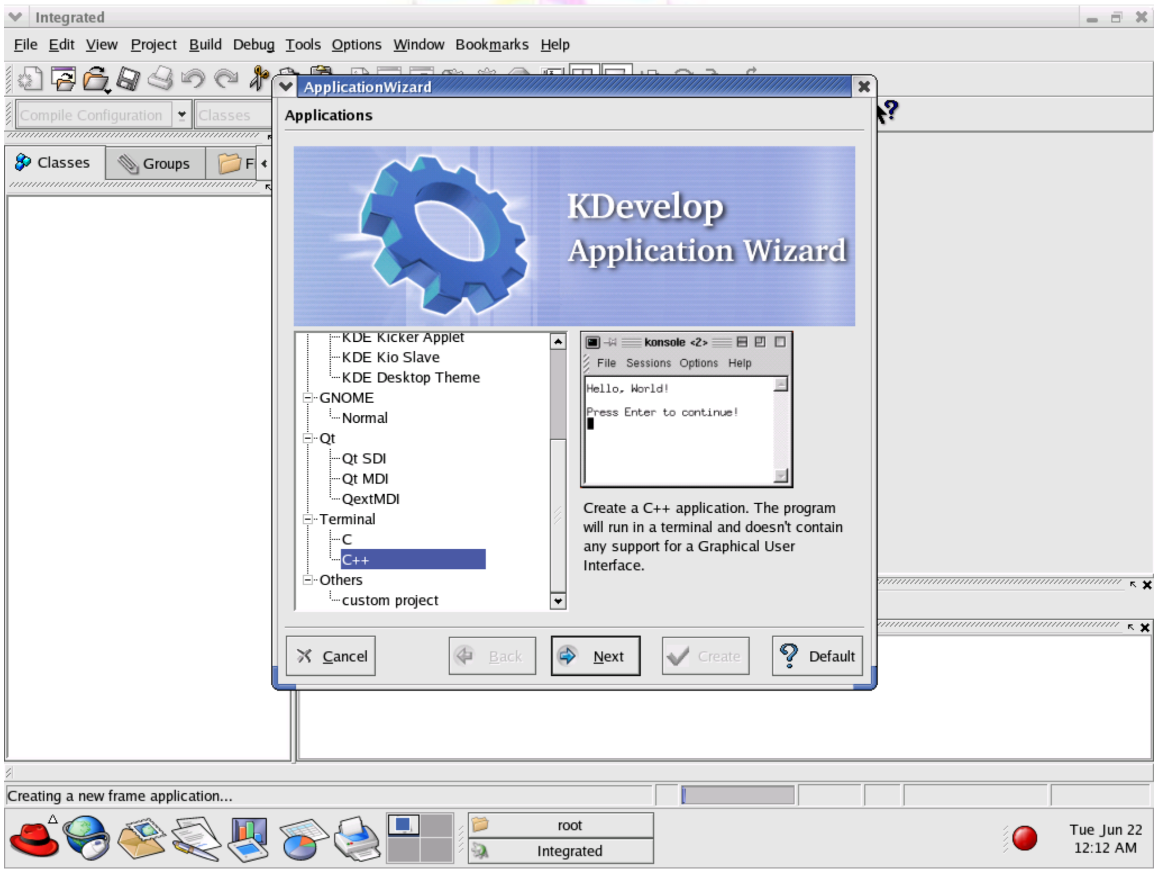
2.2.1: Running a program on the command prompt->

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++

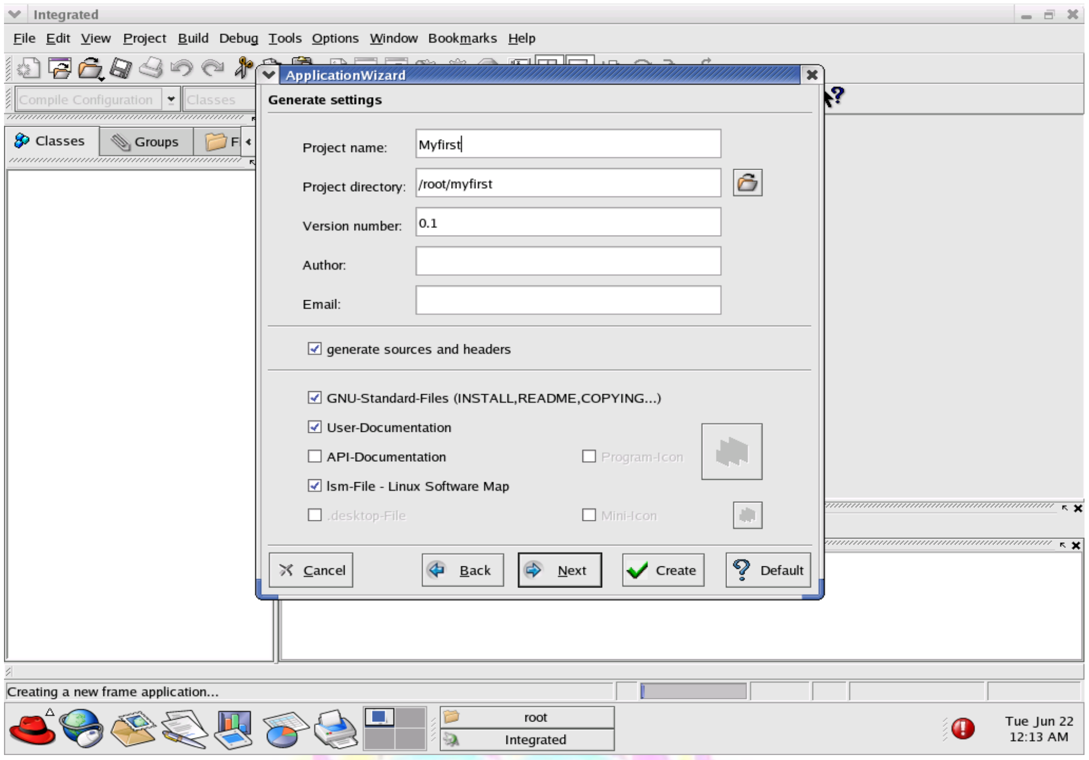


```
Hello, World!
Press Enter to continue!
```

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

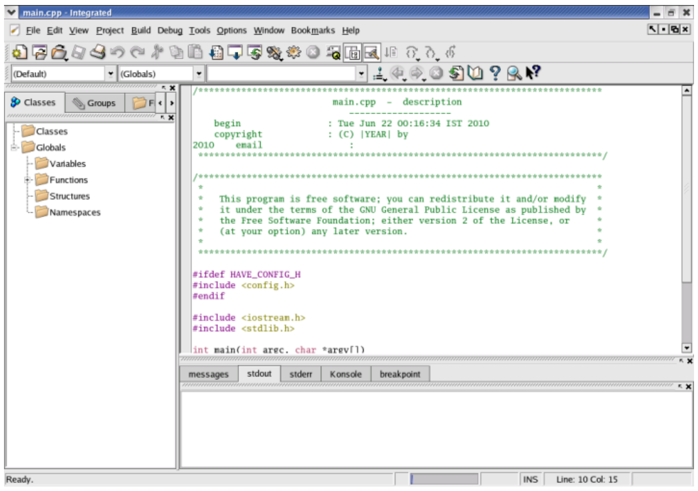
Project name: Myfirst
Project directory: /root/myfirst
Version number: 0.1
Author:
Email:

[✓] generate sources and headers

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

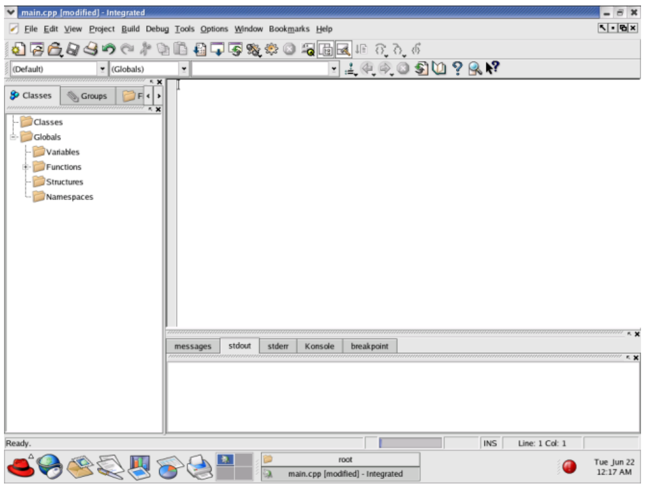
Creating a new frame application...

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



```
/***************************************************************************
                          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[])
```

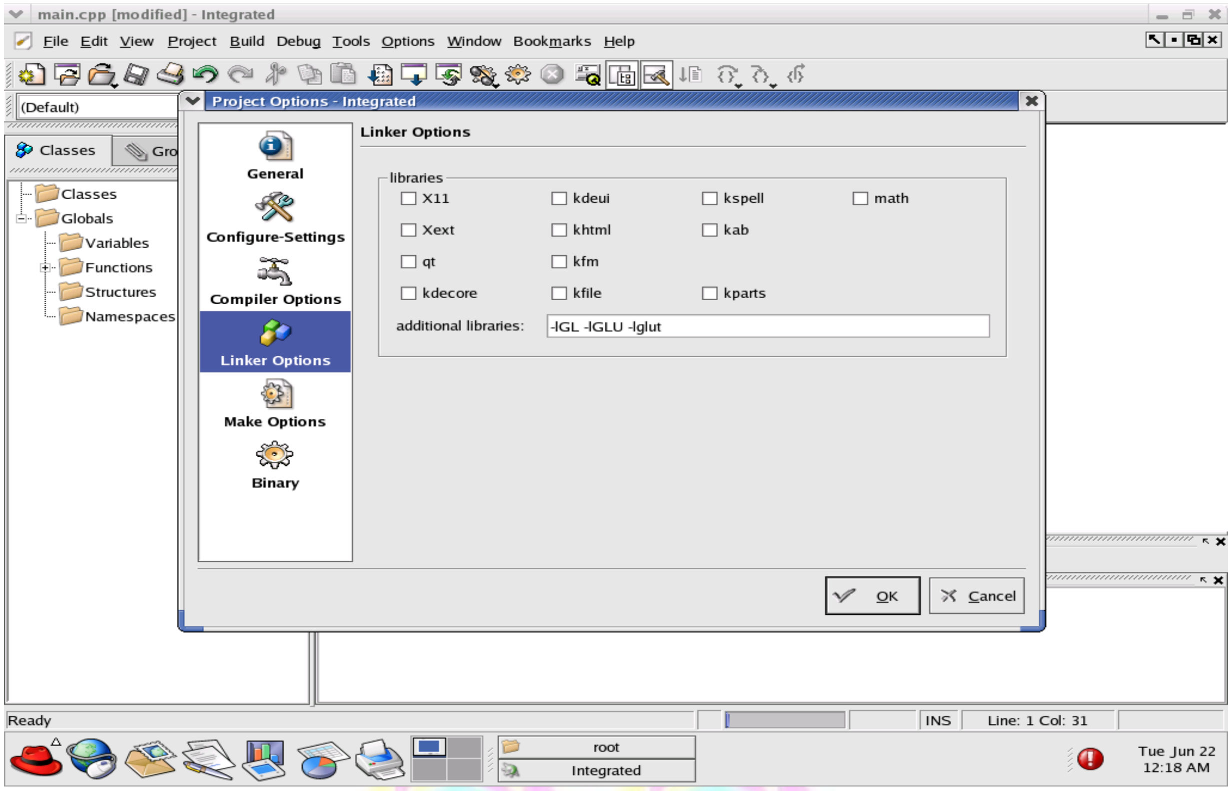
Ready. INS Line: 10 Col: 15



Ready. INS Line: 1 Col: 1

Prepared By: Aslam J K & Shonali R

5. Go to project→ options
6. In the options window click on linker options
7. In linker options, in additional libraries field type –lGL –lGLU –lglut, click ok


main.cpp [modified] - Integrated
File Edit View Project Build Debug Tools Options Window Bookmarks Help
Project Options - Integrated
General
Configure-Settings
Compiler Options
Linker Options
Make Options
Binary
OK
Cancel
Ready
root
Integrated
Tue Jun 22
12:18 AM

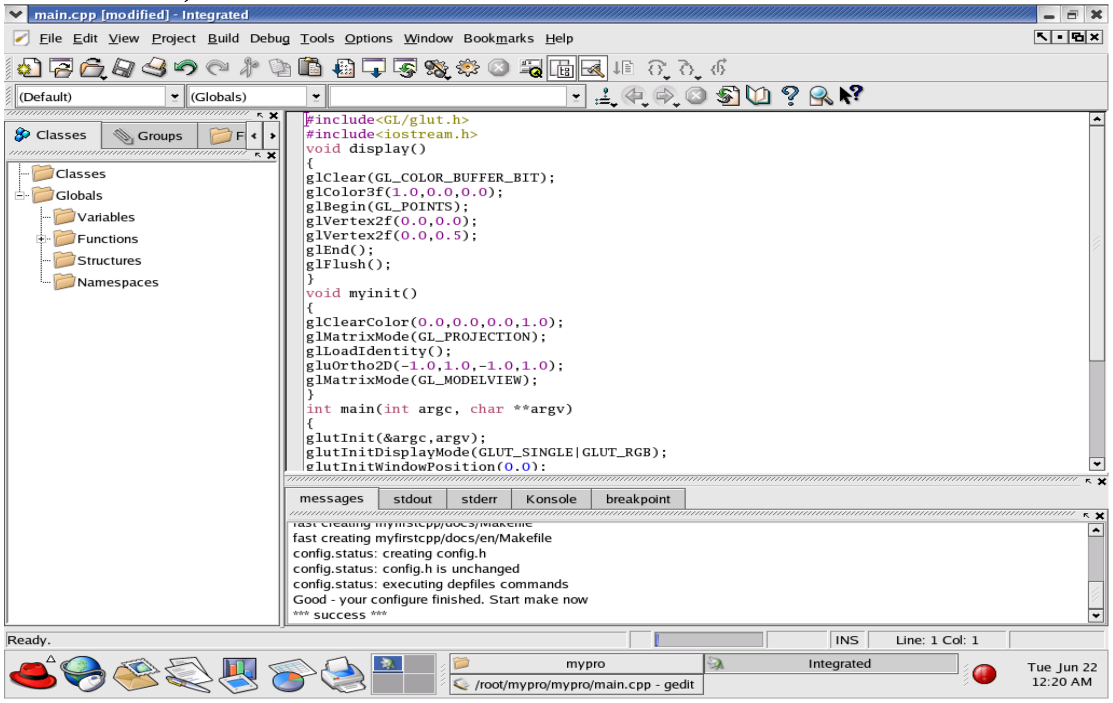
Linker Options

libraries: ☐ X11 ☐ kdeui ☐ kspell ☐ math ☐ Xext ☐ khtml ☐ kab ☐ qt ☐ kfm ☐ kdecore ☐ kfile ☐ kparts

additional libraries: -lGL -lGLU -lglut

INS Line: 1 Col: 31

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

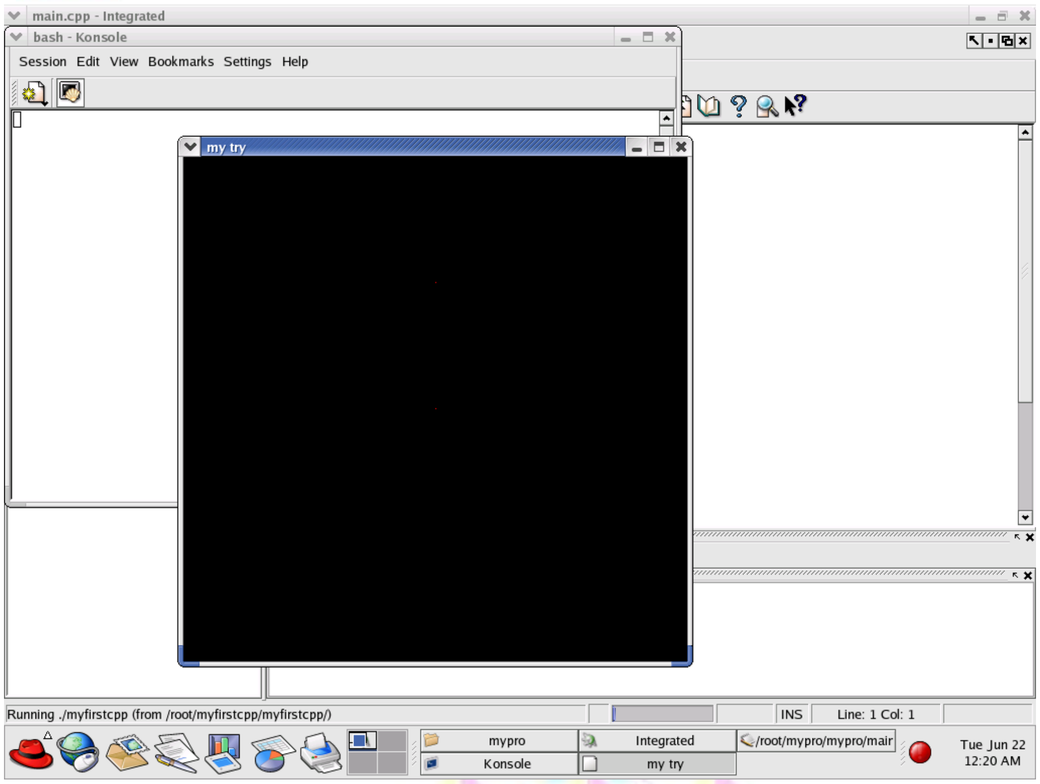


```
#include<GL/glut.h>
#include<iostream.h>
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0,0.0,0.0);
glBegin(GL_POINTS);
glVertex2f(0.0,0.0);
glVertex2f(0.0,0.5);
glEnd();
glFlush();
}
void myinit()
{
glClearColor(0.0,0.0,0.0,1.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(-1.0,1.0,-1.0,1.0);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

```
fast creating myfirstcpp/docs/Makefile
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 ***
```

INS Line: 1 Col: 1

Prepared By: Aslam J K & Shonali R

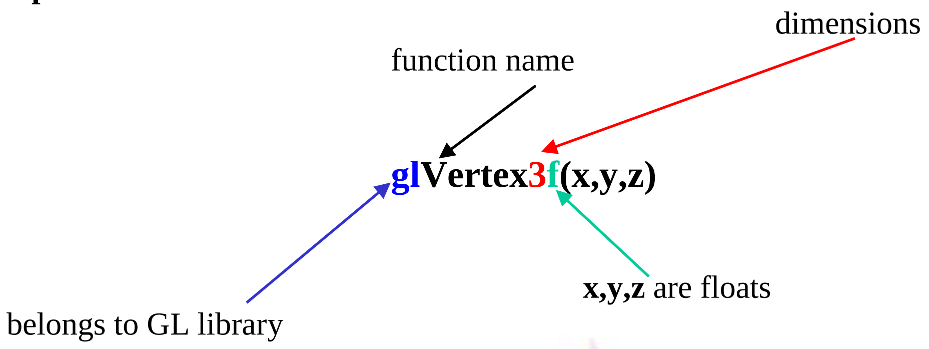


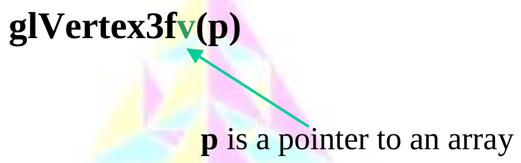
```
Running ./myfirstcpp (from /root/myfirstcpp/myfirstcpp/)
```

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>  <---------------------------  includes gl.h
#include<stdio.h>
void display()
{
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); <------------------
glutInitWindowPosition(0,0);                    define window properties
glutCreateWindow("Simple demo");
myinit(); <-------------------------------
glutDisplayFunc(display); <----------------  set OpenGL state
                                              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

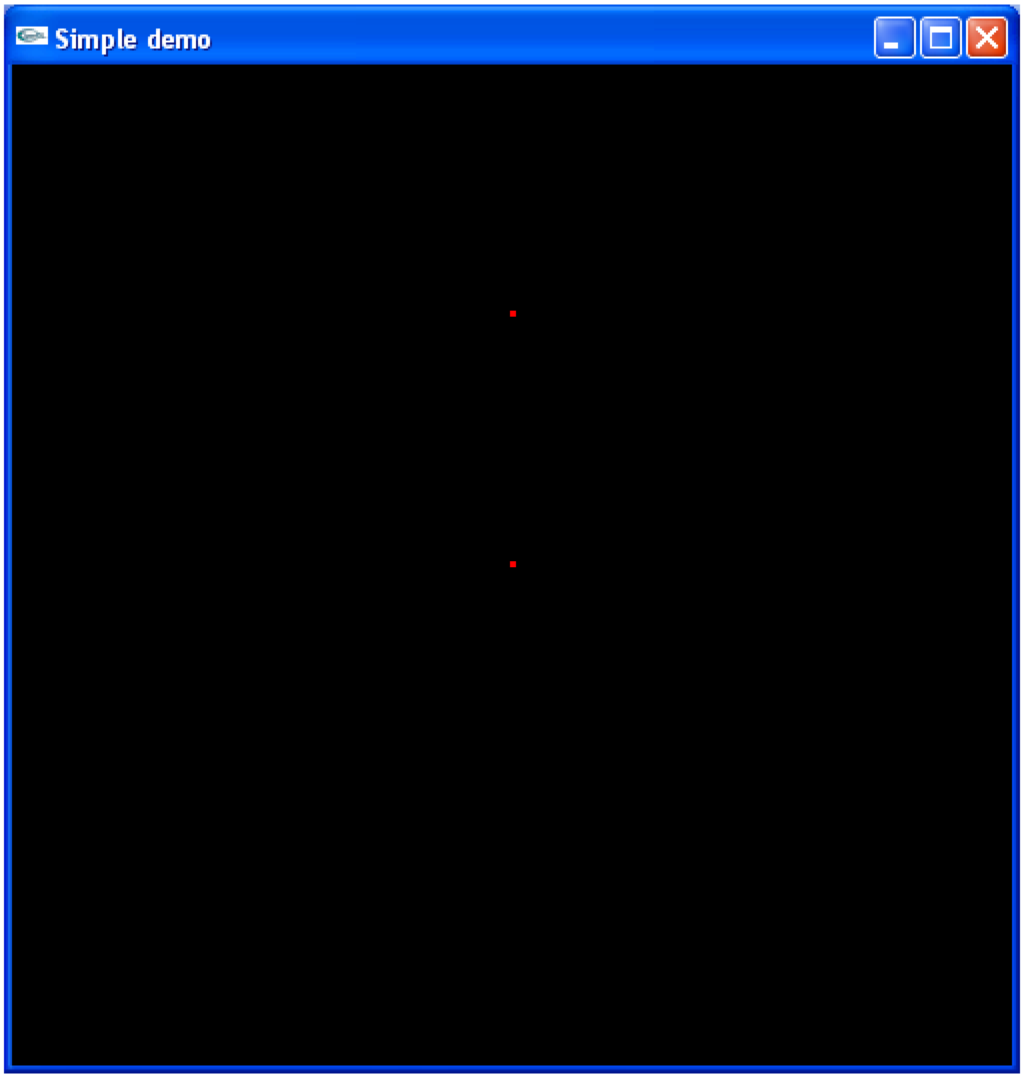
GLUT functions

- **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

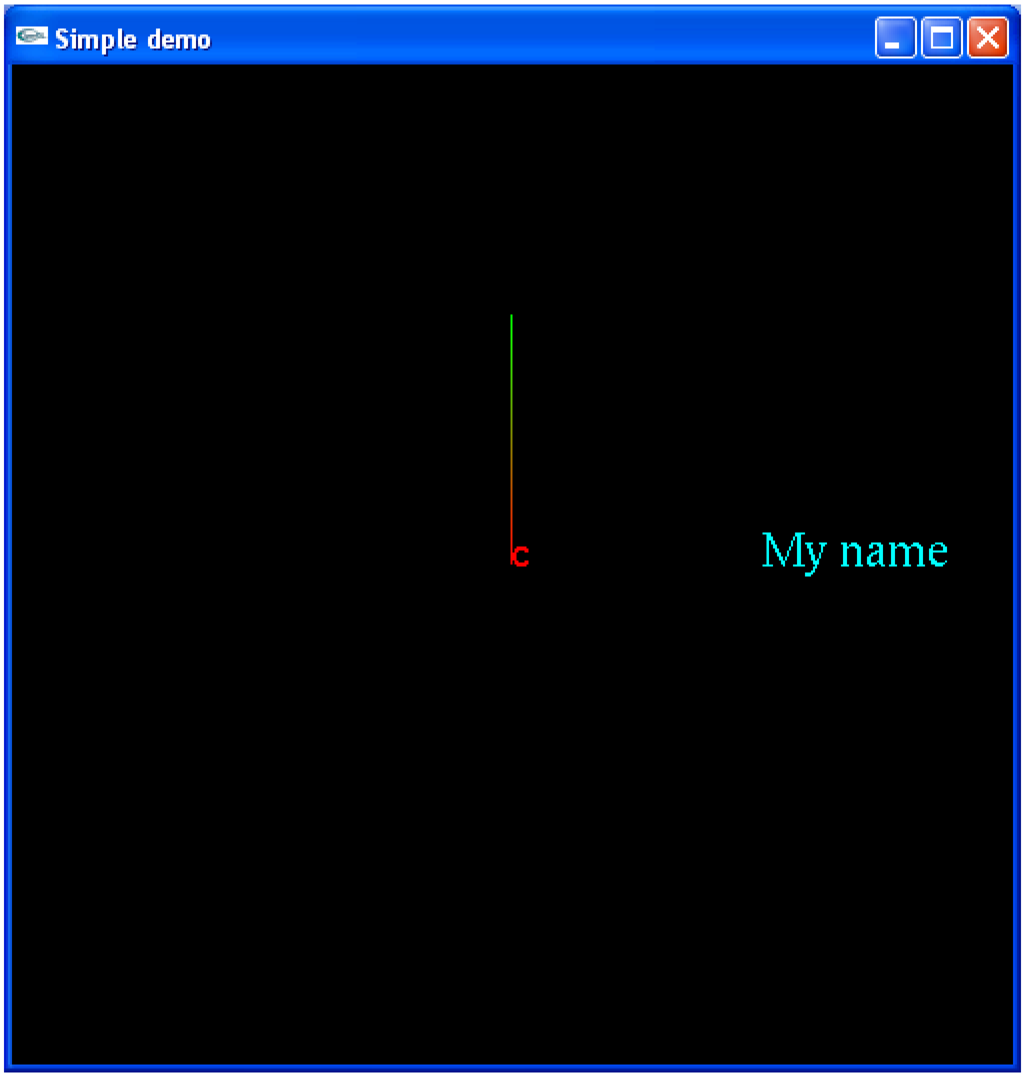
<u>Output:</u>



<u>Basics 2:</u> 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:



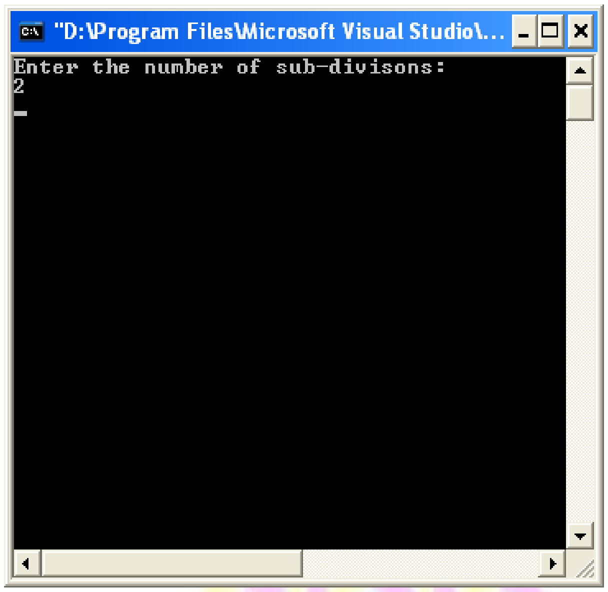
LAB1: Program to recursively subdivide a tetrahedron to from 3D Sierpinski gasket. The number of recursive steps is to be specified by the user.

Program Code:

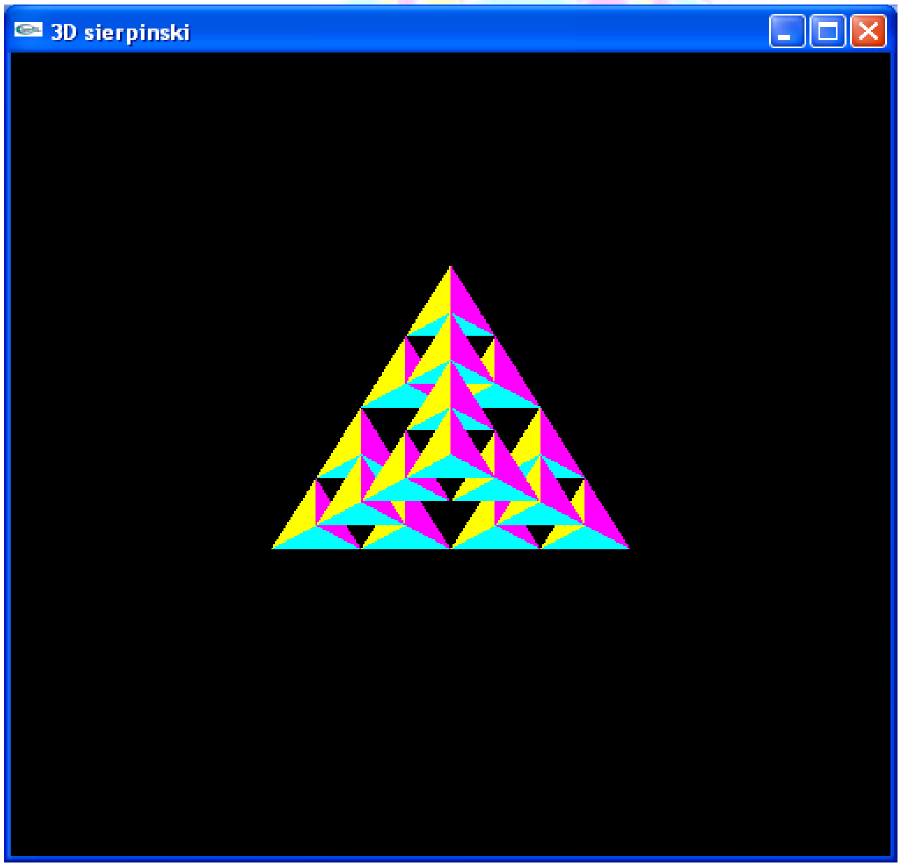
```
#include<GL/glut.h>
#include<stdio.h>
typedef float point[3];
point v[4]={{0.0,0.0,1.0},{0.0,0.942809,-0.33333},{-0.816497,-0.471405,-
0.333333},{0.816497,-0.471405,-0.333333}};
int n;/*recursive steps*/
void triangle(point a,point b,point c)
{
  glBegin(GL_TRIANGLES);
  glNormal3fv(a);
  glVertex3fv(a);
  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
```



LAB2: Program to implement liang barsky line clipping algorithm.

Algorithm at work:
Parametric form

- A line segment with endpoints (x0, y0) and (xend, yend)
  we can describe in the parametric form
  - $x = x0 + u\Delta x$
  - $y = x0 + u\Delta y \quad 0 \le u \le 1$
  - where
  - $\Delta x = xend – x0$
  - $\Delta y = yend – y0$
- The parametric equation of the line segment:
  - Defines a starting and ending point
  - Defines a direction
  - Can be easily extended to 3D
  - Is better than the line equation for many computer graphics applications

Liang-Barsky Line-Clipping

- More efficient than Cohen-Sutherland
- A line is inside the clipping region for values of u such that:
  $xw\text{min} \le x0 + u\Delta x \le xw\text{max} \qquad \Delta x = x\text{end} – x0$
  $yw\text{min} \le y0 + u\Delta y \le yw\text{max} \qquad \Delta y = y\text{end} – y0$

- Can be described as
  $u\, p\text{k} \le q\text{k}$, k = 1, 2, 3, 4

The infinitely line intersects the clip region edges when:

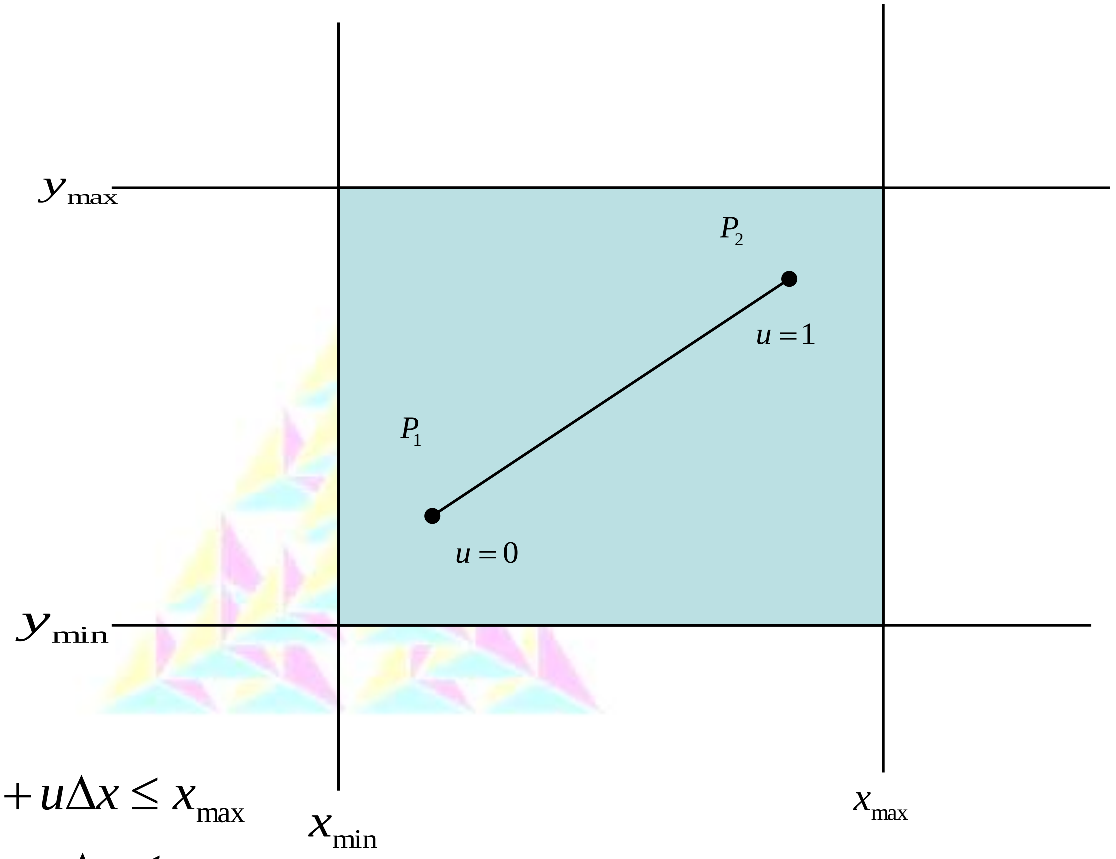
- Where p and q are defined as:

$p1 = -\Delta x$,	$q1 = x1 – xwmin$	(left , k=1)
$p2 = \Delta x$,	$q2 = xwmax - x1$	(right, k=2)
$p3 = -\Delta y$,	$q3 = y1 – ywmin$	(bottom, k=3)
$p4 = -\Delta y$,	$q4 = ywmax - y1$	(top, k=4)

The 4 in equalities are

- $xwmin <= x1 + u\Delta x$
- $x1 + u\Delta x <= xwmax$
- $ywmin <= y1 + u\Delta y$
- $y1 + u\Delta y <= ywmax$

Each of these four inequalities can be expressed as
upk <= qk, k = 1, 2, 3, 4
-uΔx= x1 - xwmin
uΔx = xwmax –x1
Thus, *u p*k ≤ *q*k



$$x_{\min} \le x_1 + u\Delta x \le x_{\max}$$
$$y_{\min} \le y_1 + u\Delta y \le y_{\max}$$
$$\forall 0 \le u \le 1$$

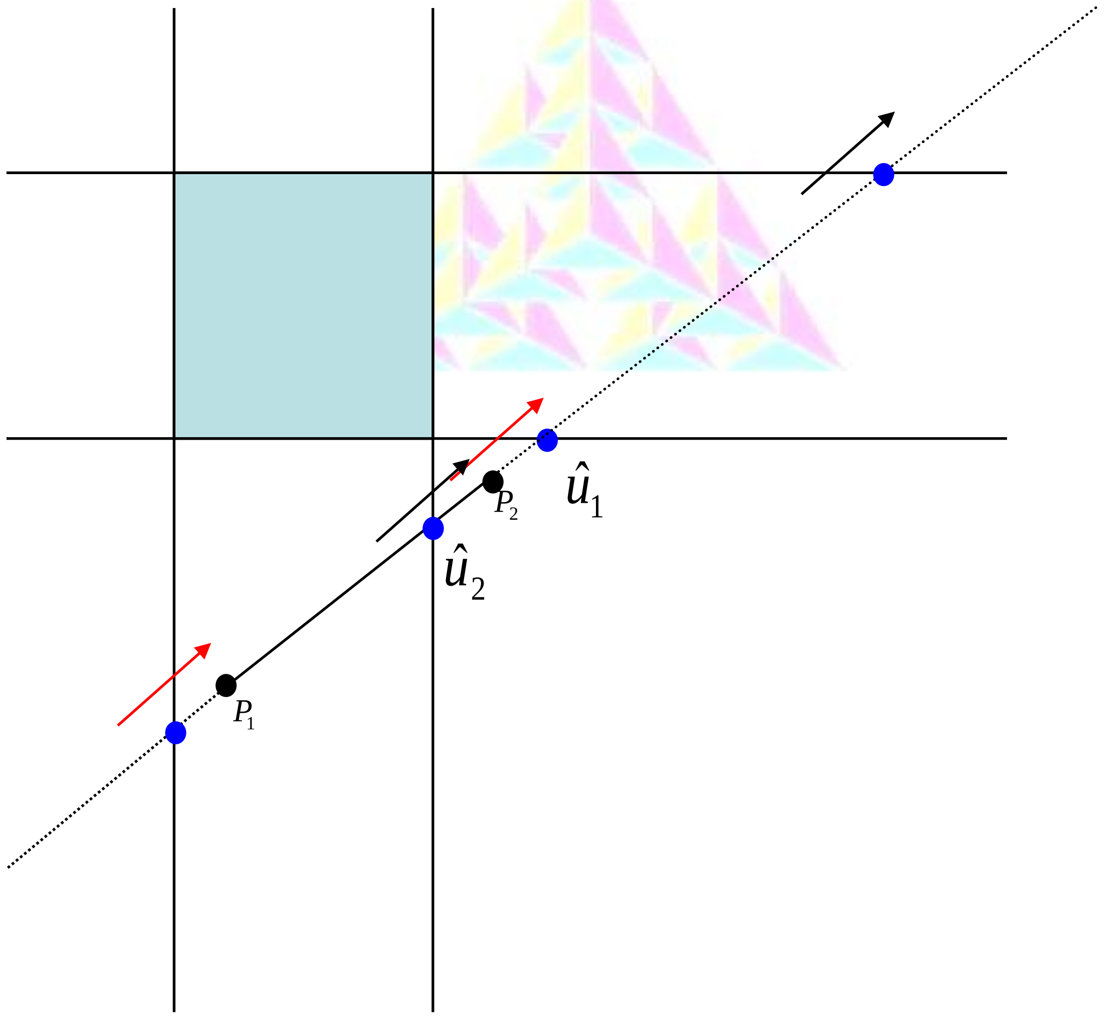
$\overline{P_1P_2}$ is contained in the window !!

- Horizontal or vertical lines:
  - Any line that is parallel to a boundary has pk = 0 for the value of k corresponding to the boundary.
  - If, for that value of k, qk < 0, then the line is completely outside and can be eliminated.
  - If qk >= 0, the line is inside the boundary.
- Lines entering or leaving the boundary:
  - When pk<0, the infinite extension of the line proceeds from the outside to the inside of the infinite extension of this particular clipping boundary
  - When pk>0, the line proceeds from the inside to the outside

– The value of u for the intersection with boundary k is
  u=qk/pk

- For each line, u1 and u2 define the part of the line that lies within the clip rectangle.
  - u1 correspond to edges for which the line proceeds from the outside to the inside. u1 is the largest of 0 and the "entering" u values.
  - u2 corresponds to edges for which the line proceeds from the inside to the outside. u2 is the minimum of 1 and the "leaving" u values.
- If u1 > u2, the line lies completely outside of the clipping area.
- Otherwise the segment from u1 to u2 lies inside the clipping window.

if $\hat{u}_1 > \hat{u}_2$, rejected.



In the program implementation that follows, *u* is referred as *t* . *u*1 and *u*2 referred as *te* and *tl*. denom refers pk and num refers the uk terms given in the explanation.

Program Code:

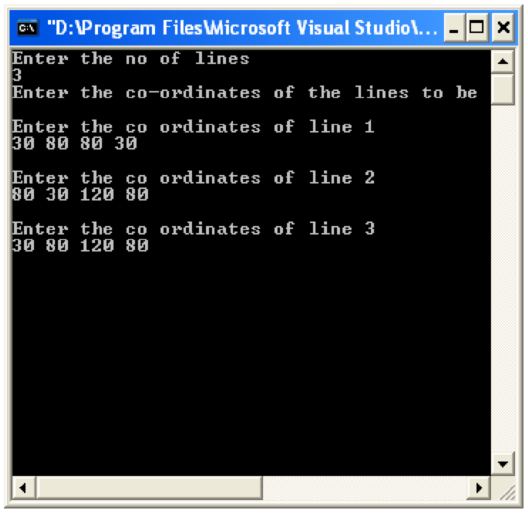
```
#include<GL/glut.h>
#include<stdio.h>
double xmin=50,xmax=100,ymin=50,ymax=100;
double xvmin=200,xvmax=300,yvmin=200,yvmax=300;
float xl[10],yl[10];
int n;
bool Clipt(float denom, float num, float *te,float *tl)
{
  double t;
  if(denom>0)
  {
        t=num/denom;
        if(t>*tl)
              return false;
        else if(t>*te)
              *te=t;
  }
  else if(denom<0)
  {
        t=num/denom;
        if(t<*te)
              return false;
        else if(t<*tl)
              *tl=t;

  }
  else if(denom==0.0) //line parallel to edge
  {
  if(num>0.0)
  {return false;}
  }
  return true;
}
bool ClipPoint(float x0,float y0)
{
  if((x0<=xmax||x0>=xmin)&&(y0<=ymax&&y0>=ymin))
  return true;
  else
        return false;
}
```

```
void LiangBarskyLineClipper(float x0,float y0,float x1,float y1)
{

  float dx=x1-x0;
  float dy=y1-y0;
  float te=0.0;
  float tl=1.0;
  bool accept=false;
  if(dx==0&&dy==0&&ClipPoint(x0,y0))
  {//degenerate line and clipped as a point
  glPointSize(2.0);
  glBegin(GL_POINTS);
  glVertex2d(x0,y0);
  glEnd();
  }
  else{
          if(Clipt(dx,xmin-x0,&te,&tl))//left edge
                  if(Clipt(-dx,x0-xmax,&te,&tl))//right edge
                          if(Clipt(dy,ymin-y0,&te,&tl))//bottom edge
                                  if(Clipt(-dy,y0-ymax,&te,&tl))//top edge
                                  {
                                   if(tl<1.0)
                                          {
                                                  x1=x0+tl*dx;
                                                  y1=y0+tl*dy;
                                          }
                                          if(te>0.0)
                                          {
                                                  x0=x0+te*dx;
                                                  y0=y0+te*dy;
                                          }
          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();
```

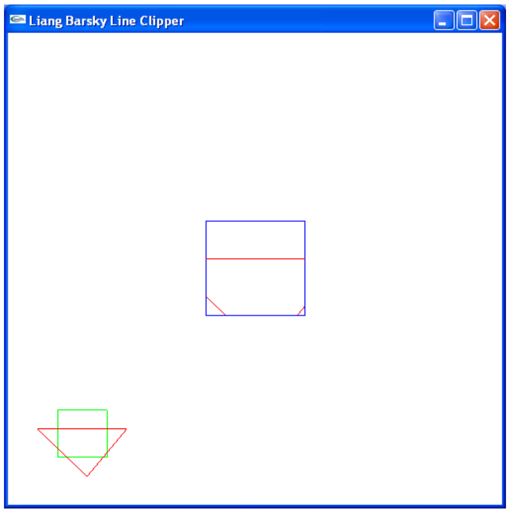
```
            glColor3f(1.0,0.0,0.0);
            glBegin(GL_LINES);
            glVertex2f(vx0,vy0);
            glVertex2f(vx1,vy1);
            glEnd();
            }
  }

}

void display()
{
  int i;
  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(i=0;i<n*2;i=i+2)
  {
  glVertex2f(xl[i],yl[i]);
  glVertex2f(xl[i+1],yl[i+1]);
  }
  glEnd();
  for( i=0;i<n*2;i=i+2)
  {

            LiangBarskyLineClipper(xl[i],yl[i],xl[i+1],yl[i+1]);
  }
  glFlush();
}
void myinit()
{
  glClearColor(1.0,1.0,1.0,1.0);
  gluOrtho2D(0.0,499.0,0.0,499.0);
}
```

```
int 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",&xl[c],&yl[c]);c++;
            }
  }

  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
  glutInitWindowPosition(0,0);
  glutInitWindowSize(500,500);
  glutCreateWindow("Liang Barsky Line Clipper");
  myinit();
  glutDisplayFunc(display);
  glutMainLoop();
}
```

Output:



```
Enter the no of lines
3
Enter the co-ordinates of the lines to be

Enter the co ordinates of line 1
30 80 80 30

Enter the co ordinates of line 2
80 30 120 80

Enter the co ordinates of line 3
30 80 120 80
```



<u>LAB 3:</u> Program to draw a color cube and spin it using OpenGL transformation matrices.

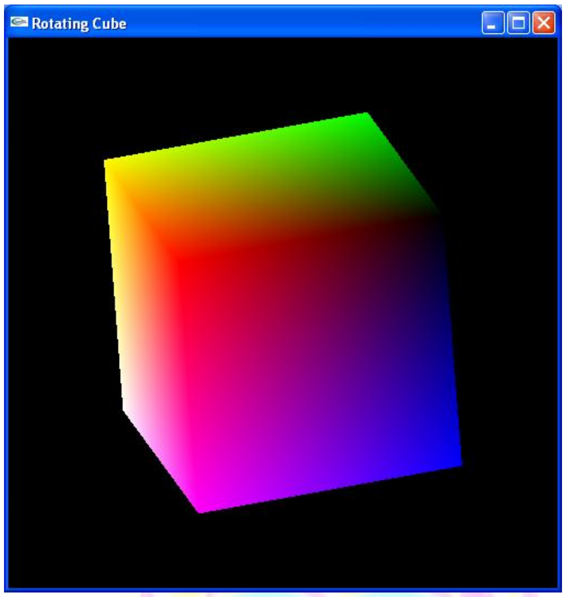
Program Code:

```
#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();
  }
```

Output:



<u>LAB 4:</u> Program to create a house like figure and rotate it about a given fixed point using OpenGL functions.

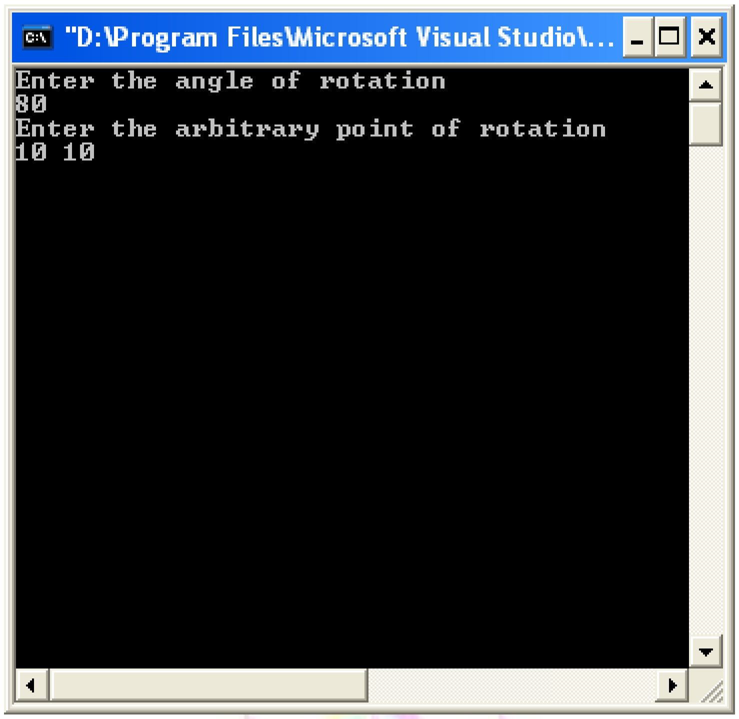
Program Code:

```
#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();
}
```

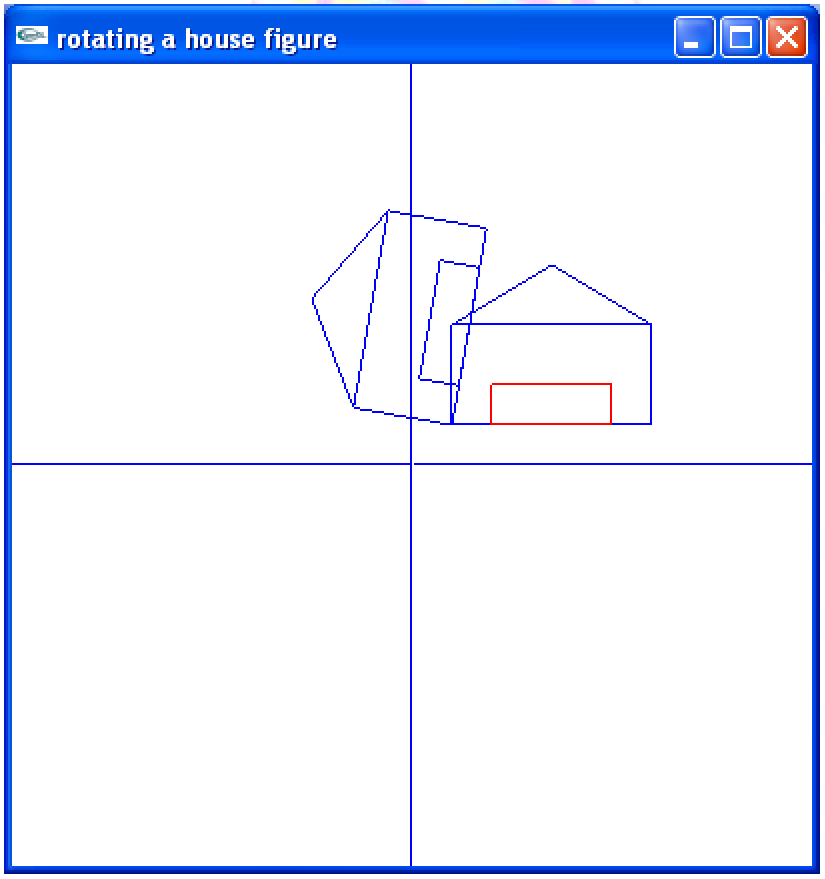
```
void drawhouse()
{
  glColor3f(0.0,0.0,1.0);
  glBegin(GL_LINE_LOOP);
  glVertex2f(house[0][0],house[1][0]);
  glVertex2f(house[0][1],house[1][1]);
  glVertex2f(house[0][3],house[1][3]);
  glVertex2f(house[0][4],house[1][4]);
  glEnd();
  glColor3f(1.0,0.0,0.0);
  glBegin(GL_LINE_LOOP);
  glVertex2f(house[0][5],house[1][5]);
  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
```



LAB 5: Program to implement the Cohen-Sutherland line-clipping algorithm. Make provision to specify the input line, window for clipping and viewport for displaying the clipped image.

Algorithm at work:

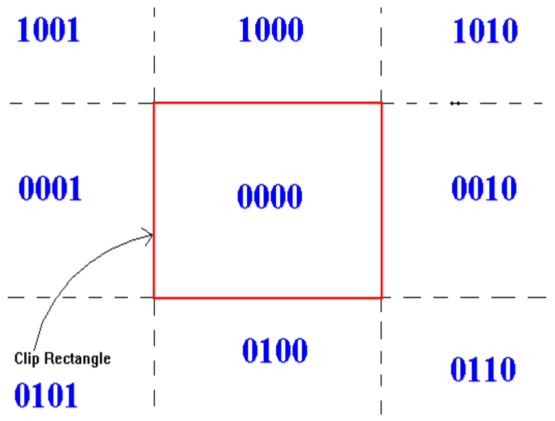
Line-Clipping
In computer graphics, **line clipping** is the process of removing lines or portions of lines outside of an area of interest. Typically, any line or part thereof which is outside of the viewing area is removed.

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.



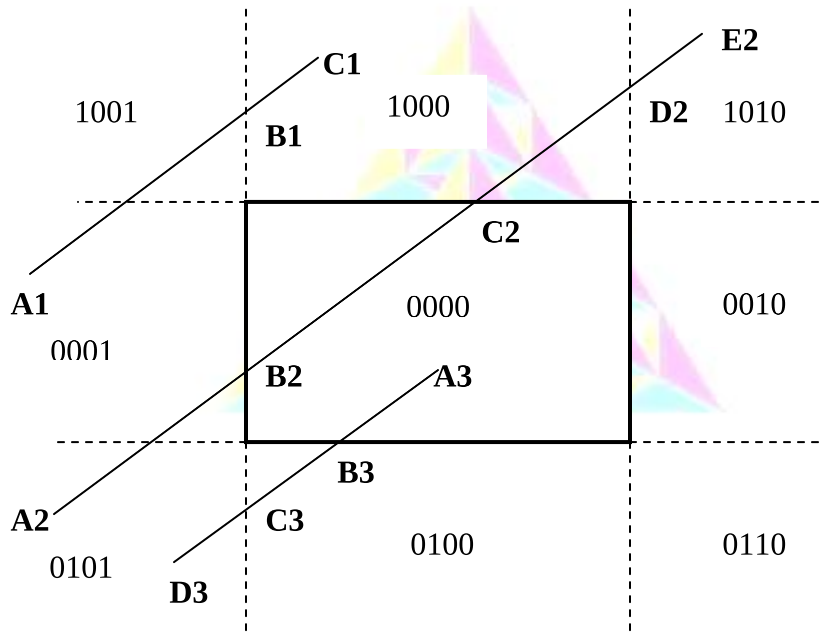
Alternate description of the algorithm:

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 $C_0 \wedge C_{end} \neq 0$ then P_0P_{end} is trivially rejected
3. If $C_0 \vee C_{end} = 0$ then P_0P_{end} is trivially accepted
4. Otherwise subdivide and go to step 1 with new segment

C_0 = Bit code of **P_0**
C_{end} = Bit code of **P_{end}**



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**	

Program Code:

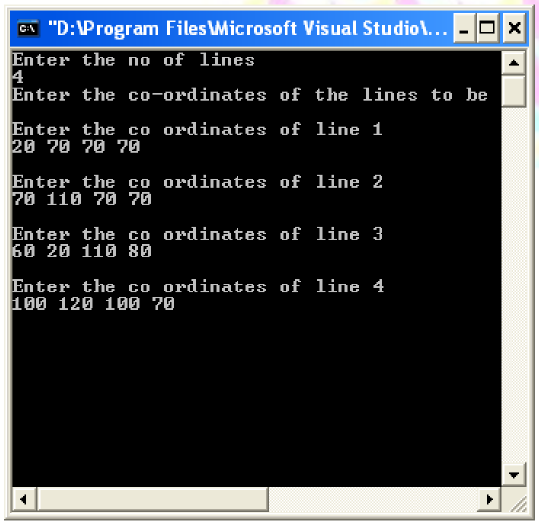
```
#include<GL/glut.h>
#include<stdio.h>
double xmin=50,xmax=100,ymin=50,ymax=100;
double xvmin=200,xvmax=300,yvmin=200,yvmax=300;
float xc[10],yc[10];
int n;
typedef int outcode;
const int TOP=8;
const int BOTTOM=4;
const int RIGHT=2;
const int LEFT=1;
/*computing/ assigning region codes to end points 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();
```

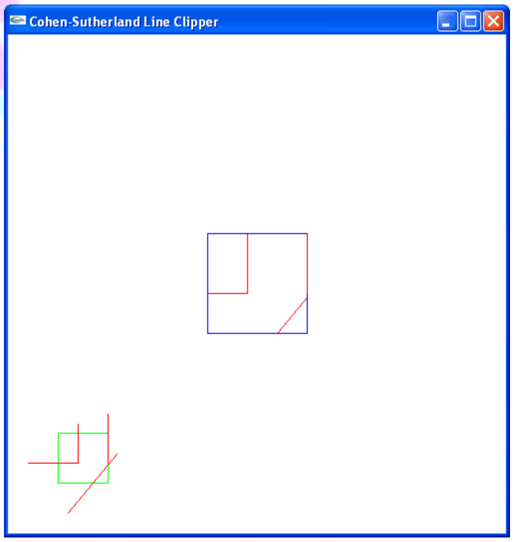
```
            glColor3f(1.0,0.0,0.0);
            glBegin(GL_LINES);
            glVertex2f(vx0,vy0);
            glVertex2f(vx1,vy1);
            glEnd();
   }
}
void display()
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(0.0,1.0,0.0);
   glBegin(GL_LINE_LOOP);
   glVertex2f(xmin,ymin);
   glVertex2f(xmax,ymin);
   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();
}
```

Output:



```
"D:\Program Files\Microsoft Visual Studio\...
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
```



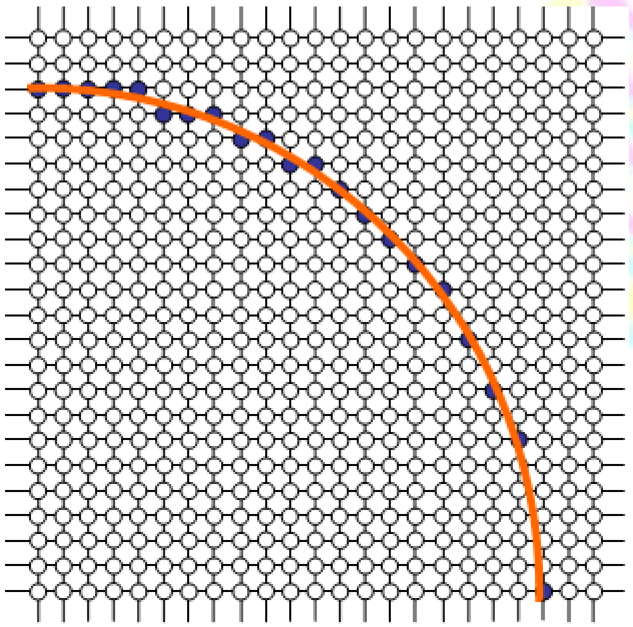
LAB 6: Program to create a cylinder and parallelopipd by extruding circle and quadilaterals, allow the user to specify the cicle and quadrilateral respectively.

Algorithm at work:
A Simple Circle Drawing Algorithm
The equation for a circle is:

$$x^2 + y^2 = r^2$$

where *r* is the radius of the circle
So, we can write a simple circle drawing algorithm by solving the equation for *y* at unit *x* intervals using:

$$y = \pm\sqrt{r^2 - x^2}$$



$$y_0 = \sqrt{20^2 - 0^2} \approx 20$$

$$y_1 = \sqrt{20^2 - 1^2} \approx 20$$

$$y_2 = \sqrt{20^2 - 2^2} \approx 20$$

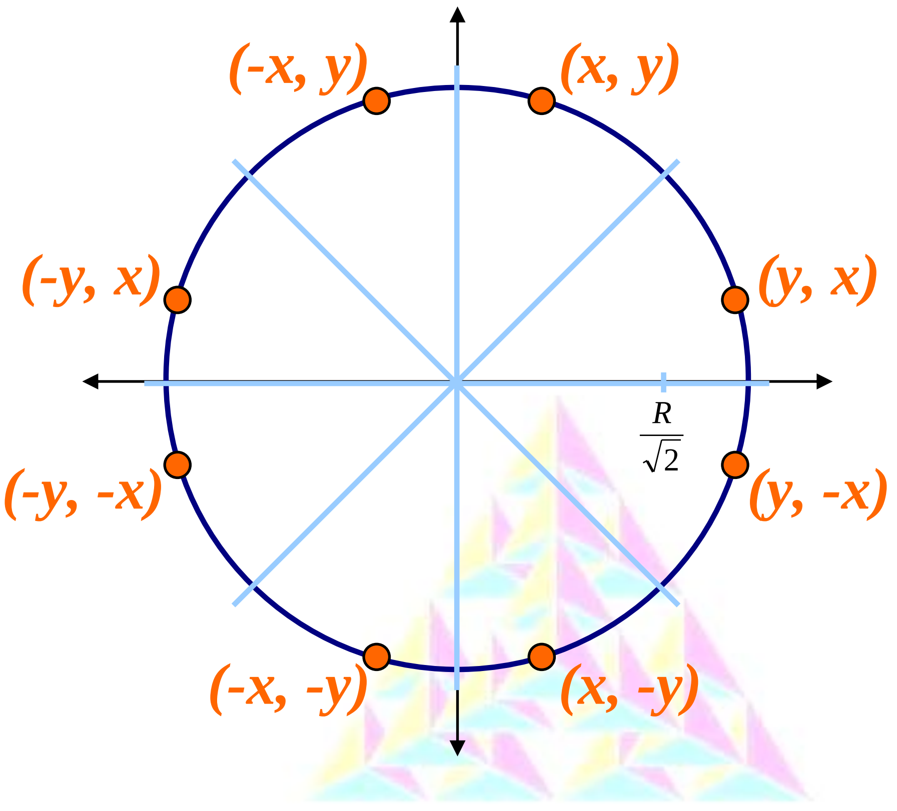
$$y_{19} = \sqrt{20^2 - 19^2} \approx 6$$

$$y_{20} = \sqrt{20^2 - 20^2} \approx 0$$

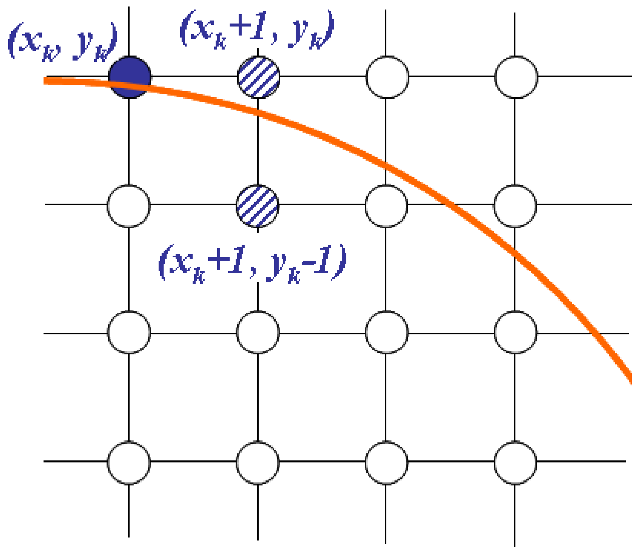
However, unsurprisingly this is not a brilliant solution!
Firstly, the resulting circle has large gaps where the slope approaches the vertical
Secondly, the calculations are not very efficient

- The square (multiply) operations
- The square root operation – try really hard to avoid these!

We need a more efficient, more accurate solution
The first thing we can notice to make our circle drawing algorithm more efficient is that circles centred at (*0, 0*) have *eight-way symmetry*



Similarly to the case with lines, there is an incremental algorithm for drawing circles – the *mid-point circle algorithm*
In the mid-point circle algorithm we use eight-way symmetry so only ever calculate the points for the top right eighth of a circle, and then use symmetry to get the rest of the points
Assume that we have just plotted point *(xk, yk)*
The next point is a choice between *(xk+1, yk)* and *(xk+1, yk-1)*
We would like to choose the point that is nearest to the actual circle
So how do we make this choice?



Let’s re-jig the equation of the circle slightly to give us:

$$f_{circ}(x,y) = x^2 + y^2 - r^2$$

The equation evaluates as follows:

$$f_{circ}(x,y) \begin{cases} < 0, & \text{if } (x,y) \text{ is inside the circle boundary} \\ = 0, & \text{if } (x,y) \text{ is on the circle boundary} \\ > 0, & \text{if } (x,y) \text{ is outside the circle boundary} \end{cases}$$

By evaluating this function at the midpoint between the candidate pixels we can make our decision
Assuming we have just plotted the pixel at (*xk*,*yk*) so we need to choose between (*xk*+1,*yk*) and (*xk*+1,*yk*-1)
Our decision variable can be defined as:

$$\begin{aligned} p_k &= f_{circ}(x_k + 1, y_k - \tfrac{1}{2}) \\ &= (x_k + 1)^2 + (y_k - \tfrac{1}{2})^2 - r^2 \end{aligned}$$

If *pk* < 0 the midpoint is inside the circle and and the pixel at *yk* is closer to the circle
Otherwise the midpoint is outside and *yk*-1 is closer
To ensure things are as efficient as possible we can do all of our calculations incrementally
First consider:

$$\begin{aligned} p_{k+1} &= f_{circ}\left(x_{k+1} + 1, y_{k+1} - \tfrac{1}{2}\right) \\ &= [(x_k + 1) + 1]^2 + \left(y_{k+1} - \tfrac{1}{2}\right)^2 - r^2 \end{aligned}$$

or:

$$p_{k+1} = p_k + 2(x_k + 1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$$

where *yk+1* is either *yk* or *yk*-1 depending on the sign of *pk*
The first decision variable is given as:

$$\begin{aligned} p_0 &= f_{circ}(1, r - \tfrac{1}{2}) \\ &= 1 + (r - \tfrac{1}{2})^2 - r^2 \\ &= \tfrac{5}{4} - r \end{aligned}$$

Then if *pk* < 0 then the next decision variable is given as:

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

If *pk* > 0 then the decision variable is:

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_k + 1$$

MID-POINT CIRCLE ALGORITHM

1. Input radius *r* and circle centre *(xc, yc)*, then set the coordinates for the first point on the circumference of a circle centred on the origin as:

   $$(x_0, y_0) = (0, r)$$

2. Calculate the initial value of the decision parameter as:

   $$p_0 = 5/4 - r$$

3. Starting with *k = 0* at each position *xk*, perform the following test. If *pk < 0*, the next point along the circle centred on *(0, 0)* is *(xk+1, yk)* and:

   $$p_{k+1} = p_k + 2x_{k+1} + 1$$

   Otherwise the next point along the circle is *(xk+1, yk-1)* and:

   $$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$$

4. Determine symmetry points in the other seven octants
5. Move each calculated pixel position *(x, y)* onto the circular path centred at *(xc, yc)* to plot the coordinate values:

   $$x = x + x_c \qquad y = y + y_c$$

6. Repeat steps 3 to 5 until *x >= y*

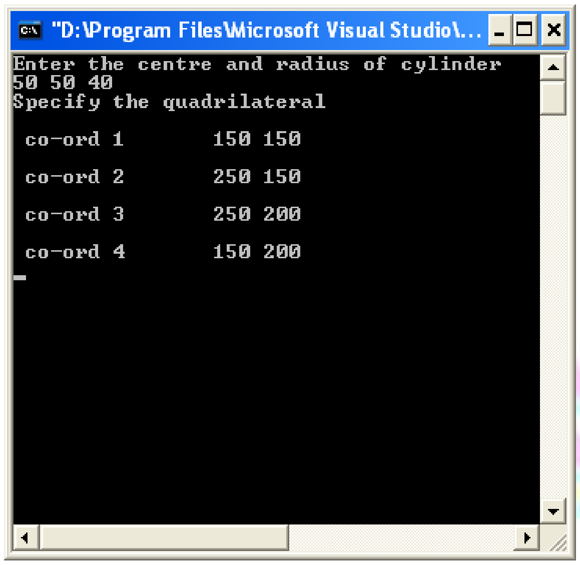
Program Code:

```
#include<GL/glut.h >
#include<stdio.h>
GLint xc,yc,r;
GLint x[5],y[5];
void draw_pixel(GLint cx,GLint cy)
{
 glColor3f(1.0,0.0,0.0);
 glBegin(GL_POINTS);
 glVertex2i(cx,cy);
glEnd();
}
void plot_pixels(GLint h,GLint k,GLint x,GLint y)
{
  draw_pixel(x+h,y+k);
  draw_pixel(-x+h,y+k);
  draw_pixel(x+h,-y+k);
  draw_pixel(-x+h,-y+k);
  draw_pixel(y+h,x+k);
  draw_pixel(-y+h,x+k);
  draw_pixel(y+h,-x+k);
  draw_pixel(-y+h,-x+k);
}
void circle_draw(GLint h,GLint k,GLint r)
{
  GLint d=1-r,x=0,y=r;
  while(y>x)
  {
          plot_pixels(h,k,x,y);
          if(d<0)d+=2*x+3;
          else
          {
                  d+=2*(x-y)+5;
                  --y;
          }
          ++x;
  }
  plot_pixels(h,k,x,y);
}
```

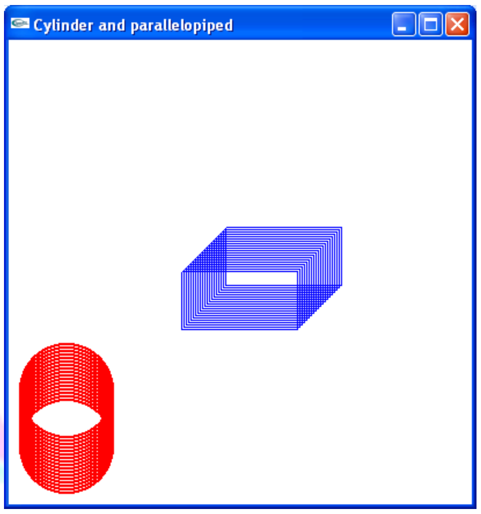
```
void cylinder_draw()
{
  GLint i,n=50;
  for(i=0;i<n;i+=3)
          circle_draw(xc,yc+i,r);
}
void parallelopiped()
{
  GLint i,j,n=40;
  glColor3f(0.0,0.0,1.0);
  glPointSize(2.0);
  for(i=0;i<n;i+=2)
  {
  glBegin(GL_LINE_LOOP);
  for(j=0;j<4;j++)
  glVertex2i(x[j]+i,y[j]+i);
  glEnd();
  }
}

void myinit()
{
  glClearColor(1.0,1.0,1.0,1.0);
  glMatrixMode(GL_PROJECTION);
  gluOrtho2D(0.0,400.0,0.0,400.0);
}
void display()
{
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0,0.0,0.0);
  glPointSize(2.0);
  cylinder_draw();
  parallelopiped();
  glFlush();
}
void main(int argc,char** argv)
{
  printf("Enter the centre and radius of cylinder\n");
  scanf("%d%d%d",&xc,&yc,&r);
  printf("Specify the quadrilateral\n");
  for(int i=0;i<4;i++)
  {printf("\n co-ord %d\t",i);
          scanf("%d%d",&x[i],&y[i]);
  }
```

```
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(400,400);
        glutInitWindowPosition(50,50);
        glutCreateWindow("Cylinder and parallelopiped");
        myinit();
        glutDisplayFunc(display);
        glutMainLoop();

}
```

Output:



```
Enter the centre and radius of cylinder
50 50 40
Specify the quadrilateral

 co-ord 1       150 150

 co-ord 2       250 150

 co-ord 3       250 200

 co-ord 4       150 200
```



LAB 7: Program, using OpenGL functions, 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 properties of the surfaces of the solid object used in the scene.

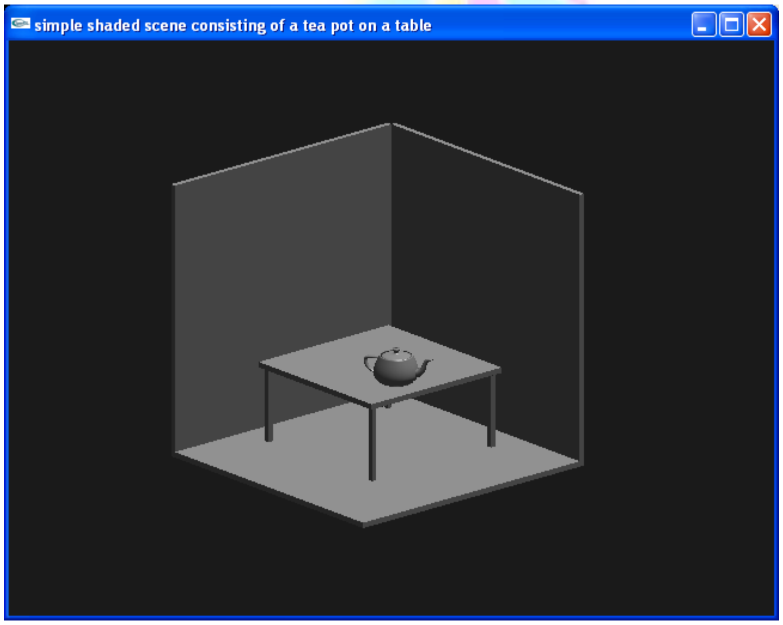
Program Code:

```
#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();
}
```

Output:



LAB 8: Program to draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing. Use OpenGL functions.

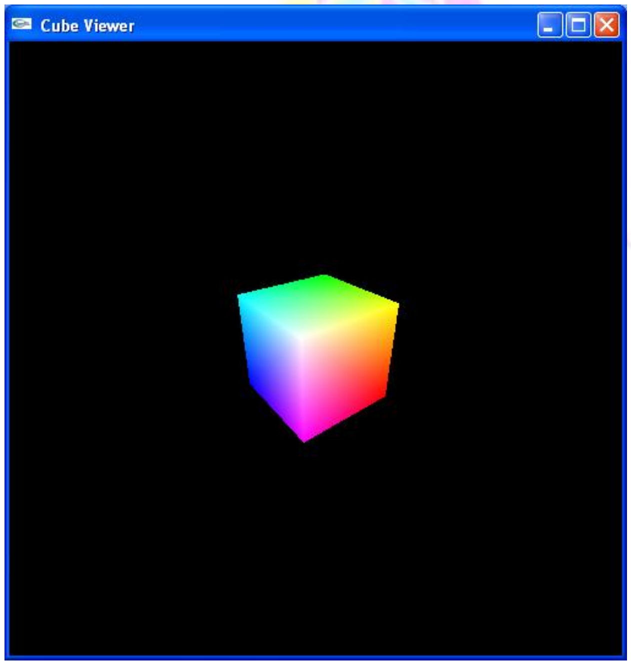
Program Code:

```
#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();
}
```

Output:

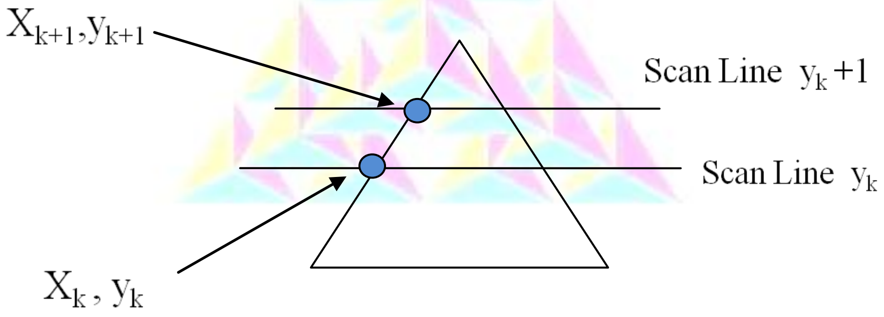


<u>LAB 9:</u> Program to fill any given polygon using scan-line area filling algorithm.(Use appropriate data structures.)

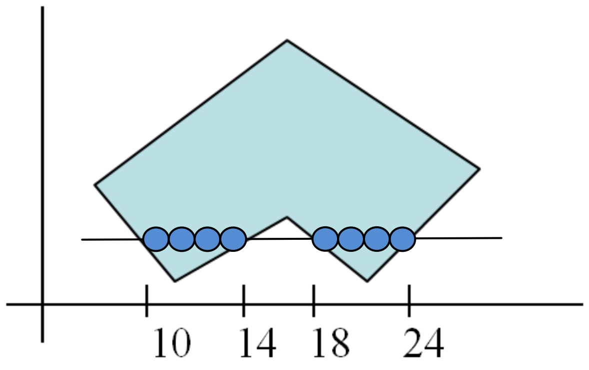
Algorithm at work:

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

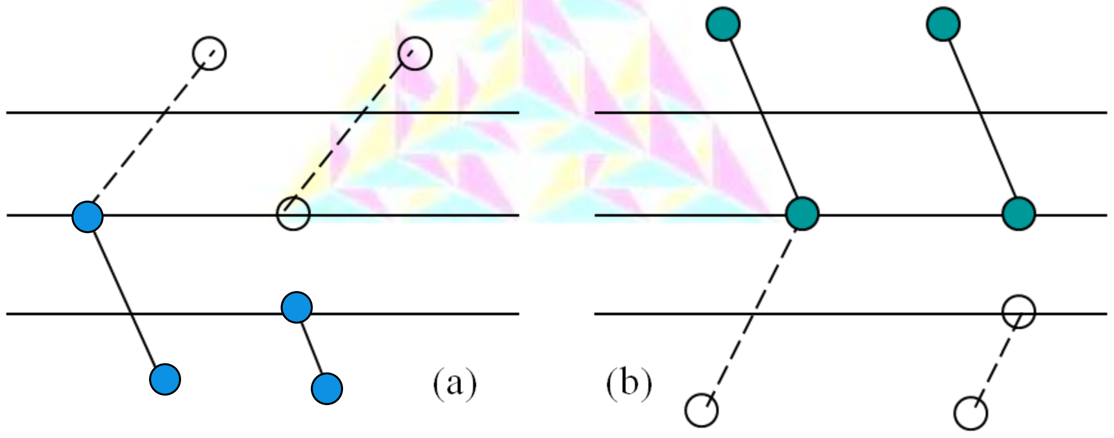


Scan Line Polygon Fill Algorithm:



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

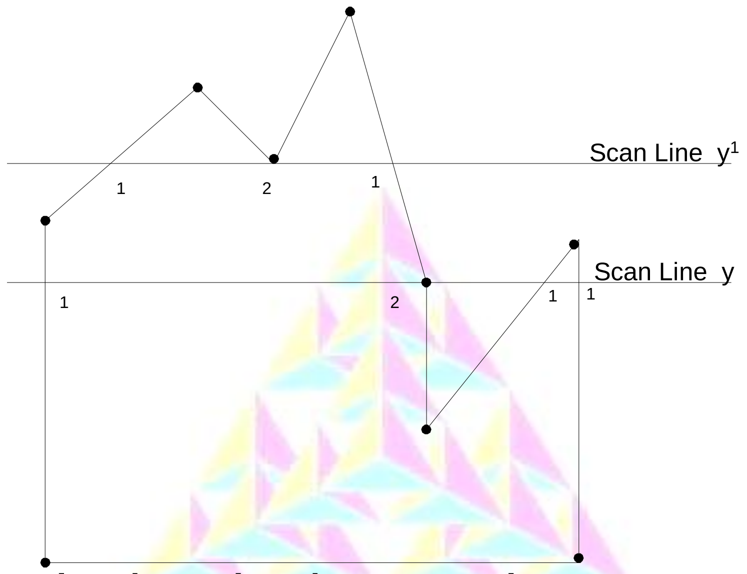


Adjusting endpoint values for a polygon, as we process edges in order around the polygon perimeter. The edge currently being processed is indicated as a solid like. In (a), the y coordinate of the upper endpoint of the current edge id decreased by 1. In (b), the y coordinate of the upper end point of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

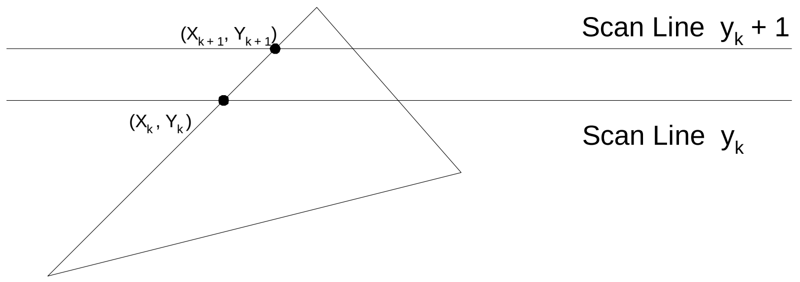
The scan conversion algorithm works as follows

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.

Program Code:

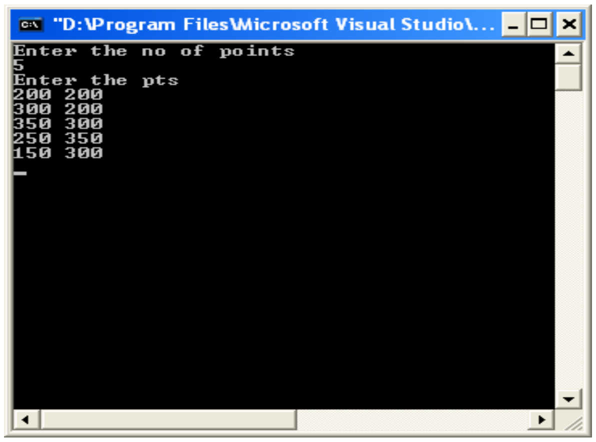
```
#include<stdio.h>
#include<GL/glut.h>
#include<stdlib.h>
#define WINDOW_HEIGHT 500
/*The edge data structure*/
typedef struct tEdge {
int yUpper;
float xIntersect, dxPerScan;
struct tEdge * next;
} Edge;
typedef struct tdcPt {
int x;
int y;
} dcPt;
int cnt;
dcPt pts[60];
/* Inserts edge into list 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;
}
/* Store lower-y coordinate and inverse slope for each edge. Adjust
and store upper-y coordinate for edges that are the lower member
of a monotically increasing or decreasing pair of edges */
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);
}
```

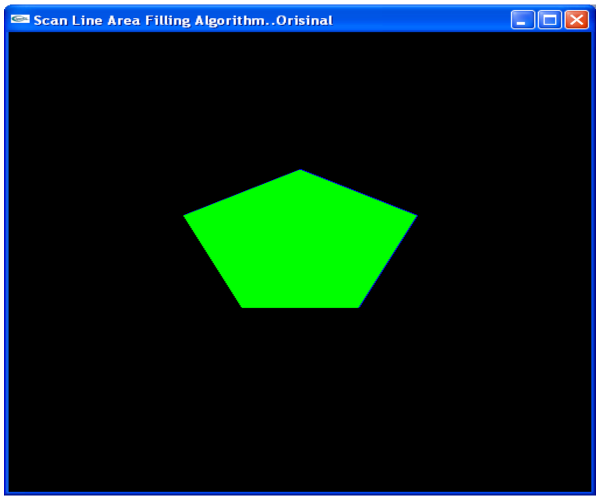
```
else
{
p->xIntersect = p->xIntersect + p->dxPerScan;/*x=x+1/m*/
q = p;
p = p->next;
}
}

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();
}
```

Output:



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



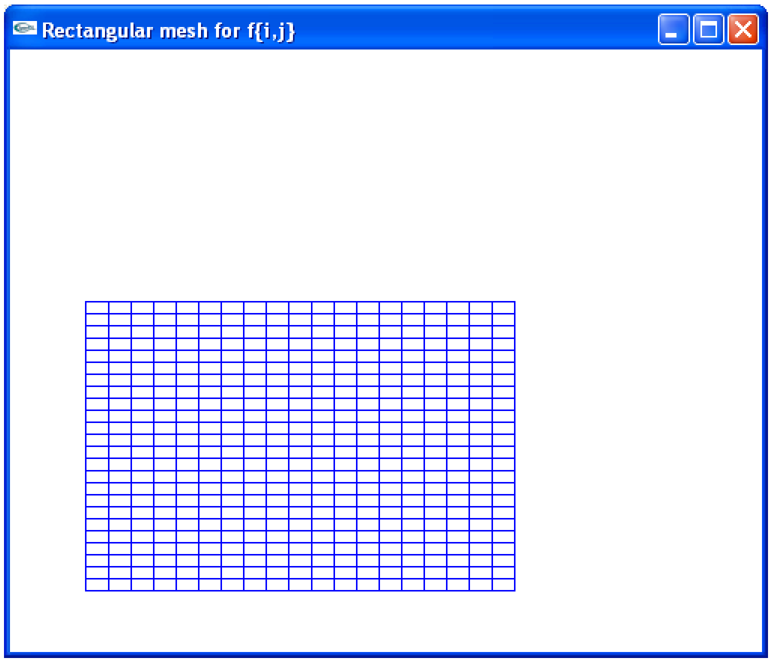
LAB 10: Program to display a set of values { fij } as a rectangular mesh.

Program Code:

```
#include<GL/glut.h>
#include<stdlib.h>
#define maxx 20
#define maxy 25
#define dx 15
#define dy 10
GLfloat x[maxx]={0.0};
GLfloat y[maxy]={0.0};
GLfloat x0=50,y0=50;
GLint i,j;
void display()
{
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(0.0,0.0,1.0);
  for(i=0;i<maxx;i++)
                  x[i]=x0+i*dx;
  for(j=0;j<maxy;j++)
                  y[j]=y0+j*dy;
  for(i=0;i<maxx-1;i++)//drawing the mesh
          for(j=0;j<maxy-1;j++)
          {
                  glBegin(GL_LINE_LOOP);
                  glVertex2f(x[i],y[j]);
                  glVertex2f(x[i+1],y[j]);
                  glVertex2f(x[i+1],y[j+1]);
                  glVertex2f(x[i],y[j+1]);
                  glEnd();
          }
  glFlush();
}

void myinit()
{
  glClearColor(1.0,1.0,1.0,1.0);
  glPointSize(5.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(0,499.0,0,499.0);
  glMatrixMode(GL_MODELVIEW);
}
```

```
void main(int argc,char **argv)
{
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
  glutInitWindowPosition(0,0);
  glutInitWindowSize(500,400);
  glutCreateWindow("Rectangular mesh for f{i,j}");
  myinit();
  glutDisplayFunc(display);
  glutMainLoop();
}
```

Output:


Rectangular mesh for f{i,j}