CHAPTER 1
INTRODUCTION

Every project has a goal or a certain idea to convey. It is needless to make one, not demonstrating the essense of the basic principles we've learned, which later would become the building blocks of what we would call technological revolution. Everything starts small. Hence, here, I wanna present a vague idea of how an object is built, block by block from pixels.

This project mainly demonstrates affine transformations of 3D objects such as translation, rotation and scaling. We are using keyboard and mouse to interact with the program. The various features of the project are:

- Projection of objects at different depths
- Demonstration of light source
- Hidden surface removal
- Keyboard and mouse interface

It has been developed using OpenGL package installed on linux. The report also includes details of the system requirements, overview, implementation, analysis, design and testing of the above mentioned project.

1.1 Computer Graphics

Graphics provides one of the most natural means of communicating within a computer, since our highly developed 2D and 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly and effectively. Interactive computer graphics is the most important means of producing pictures since the invention of photography and television. It has the added advantage that, with the computer, we can make pictures not only of concrete real world objects but also of abstract, synthetic objects, such as mathematical surfaces and of data that have no inherent geometry, such as survey results.

Computer graphics started with the display of data on hardcopy plotters and cathode ray tube screens soon after the introduction of computers themselves. It has grown to include the creation, storage, and manipulation of models and images of objects.

These models come from a diverse and expanding set of fields, and include physical, mathematical, engineering, architectural, and even conceptual structures, natural phenomena, and so on. Computer graphics today is largely interactive. The user controls the contents, structure, and appearance of the objects and of their displayed images by using input devices, such as keyboard, mouse, or touch-screen. Due to close relationships between the input devices and the display, the handling of such devices is included in the study of computer graphics. The advantages of the interactive graphics are many in number. Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D and 3D patter-recognition abilities allow us to perceive and process data rapidly and efficiently. In many design, implementation, and construction processes today, the information pictures can give is virtually indispensable. Scientific visualization became an important field in the 1980s when the scientists and engineers realized that they could not interpret the prodigious quantities of data produced in supercomputer runs without summarizing the data and highlighting trends and phenomena in various kinds of graphical representations.

1.2 OpenGL Interface

OpenGL is an application program interface (API) offering various functions to implement primitives, models and images. This offers functions to create and manipulate render lighting, coloring, viewing the models. OpenGL offers different coordinate system and frames. OpenGL offers translation, rotation and scaling of objects.

Most of our applications will be designed to access OpenGL directly through functions in three libraries. They are:

- Main GL: Library has names that begin with the letter GL and are stored in a library usually referred to as GL.
- OpenGL Utility Library (GLU): This library uses only GL functions but contains code for creating common objects and simplifying viewing.
- OpenGL Utility Toolkit (GLUT): This provides the minimum functionality that should be accepted in any modern windowing system.

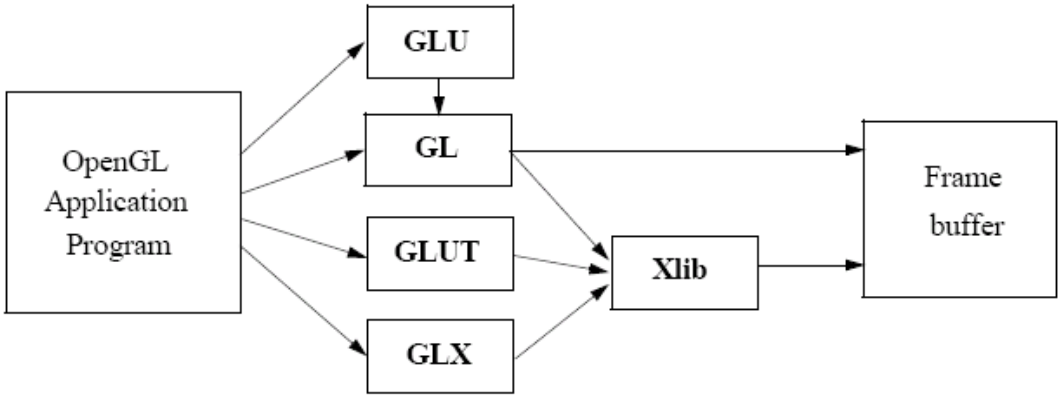


Fig1.1: Library organization

Some features of OpenGL include the following:

- Geometric and raster primitives RGBA or color index mode
- Display list or immediate mode
- Viewing and modeling transformations
- Lighting and Shading
- Hidden surface removal (Depth Buffer)
- Texture Mapping

1.3 OpenGL Overview

OpenGL(Open Graphics Library) is the interface between a graphic program and graphics hardware. *It is streamlined*. In other words, it provides low-level functionality. For example, all objects are built from points, lines and convex polygons. Higher level objects like cubes are implemented as six four-sided polygons.

OpenGL supports features like 3-dimensions, lighting, anti-aliasing, shadows, textures, depth effects, etc.

- It is system-independent. It does not assume anything about hardware or operating system and is only concerned with efficiently rendering mathematically described scenes. As a result, it does not provide any windowing capabilities.
- It is a state machine. At any moment during the execution of a program there is a current model transformation
- It is a rendering pipeline**.** The rendering pipeline consists of the following steps:

- Defines objects mathematically.
- Arranges objects in space relative to a viewpoint.
- Calculates the color of the objects.
- Rasterizes the objects.

1.4 Objective

- The aim of the project is to demonstrate the effect of hidden surface removal and give an insight into the formation of objects from pixels.
- It should be easy to understand, user interactive interface.
- Reation of primitives,that is, polygons etc.
- Providing human interaction through Mouse and Keyboard

CHAPTER 2
DESIGN

2.1 Overview

The project demonstrates formation of objects using the smallest unit of display,called pixels. This is brought about by using various tools or features embedded in openGL like hidden surface removal,bilinear interpolation. It shows the effects of lighting on the object and the basic mouse and keyboard interfaces. It uses openGL functions such as translation,rotation and scaling.

2.2 User Interface

2.2.1 Keyboard Controls

X	Moves the camera in negative x direction
x	Moves the camera in positive x direction
Y	Moves the camera in negative y direction
y	Moves the camera in positive y direction
Z	Moves the camera in negative z direction
z	Moves the camera in positive z direction

2.2.2 Mouse Interaction

Menu is displayed by clicking the Right Mouse Button (RMB). Left click selects the required option. The various options are discussed below:

- Cube: Displays only the large cube.
- Div 1: Displays the front half of cube divided into 4 equal parts.
- Div 2: Displays the front half of cube divided into 16 equal parts.
- Div 3: Displays the front half of cube divided into 64 equal parts.
- Div 2: Displays the front half of cube divided into 128 equal parts.
- Lighting: Enables light source.
- Light Off: Disables light source.

CHAPTER 3
IMPLEMENTATION

3.1 Header files

- <GL/glut.h> : This header file provides an interface with OpenGL application programs that are installed on the system.
- <stdlib.h> : This header file contains various functions for performing general functions.

3.2 OpenGL functions

glutPostRedisplay( );

- Marks the current window as needing to be redisplayed.
- **Usage:**
  void glutPostRedisplay( )

glutKeyboardFunc(keyboard func name);

- This OpenGL will teach you how to add keyboard interaction to your OpenGL application through the use of glut and glut's keyboard calls.
  Now glut has many different keyboard calls.
- But anyway, all you have to edit after that is the 'main' function.

glutInitWindowPosition( ) and glutInitWindowSize( ) **:**

- Set the *initial window position* and *size* respectively.
- **Usage:**
  void glutInitWindowSize(int width, int height);
  void glutInitWindowPosition(int x, int y);

glutCreateWindow( ):

- Creates a top-level window.
- **Usage:**
  glutCreateWindow(char *name);

glutDisplayFunc():

- Sets the display callback for the *current window*.
- **Usage:**

  void glutDisplayFunc(void (*func)(void));
- **Parameter:**

  func : The new display callback function

glutPostRedisplay( );

- Marks the current window as needing to be redisplayed.
- **Usage:**

  void glutPostRedisplay( )

glutSwapBuffers( );

- swaps the buffers of the current window if double buffered.
- **Usage:**

  void glutSwapBuffers( )

glTranslatef( );

- Displaces points by a fixed distance in a given direction.
- **Usage:**

  void glTranslate[fd](TYPE x, TYPE y, TYPE z)
- **Parameter:**

  Displacement values.

glRotatef();

- Rotate the corresponding object by a given values.
- **Usage:**

  void glRotate[fd](TYPE angle, TYPE dx, TYPE dy, TYPE dz)
- **Parameter:**

  Angle in degrees about the axis(dx,dy,dz);
- **Description:**

  Alters the current matrix by a rotation of angle in degrees.

glClearColor( );

- Set the background color.
- **Usage:**
- void glClearColor(GLclampf *red* , GLclampf *green* , GLclampf *blue* , GLclampf *alpha*);
- **Parameter:**
- Red, green, blue, and alpha values.

glEnable( ) and glDisable( );

- Enable or disable server-side GL capabilities
- **Usage:**

  void glEnable(GLenum cap);

  void glDisable(GLenum cap);
- **Parameter:**

  A symbolic constant indicating a GL capability.

glLightfv( ) ;

- Set light source parameters
- **Usage:**

  void glLightfv(GLenum light, GLenum pname, const GLfloat *params)
- **Parameter:**

  light - Specifies a light

  pname - Specifies a light source parameter for *light*.

glMaterialfv( ) ;

- Specify material parameters for the lighting model
- **Usage:**

  void glMaterialtfv(GLenum face, GLenum pname, const GLfloat *params)
- **Parameter:**

  face - Specifies which face or faces are being updated.

  pname - Specifies the single-valued material parameter of the face

params - Specifies a pointer to the value or values that *pname* will be set to.

glMatrixMode( ) ;

- Specify which matrix is the current matrix
- **Usage:**

  void glMatrixMode(GLenum mode);
- **Parameter:**

  mode - Specifies which matrix stack is the target for subsequent matrix operations.

glPushMatrix( ) and glPopMatrix( ) ;

- Push and pop the current matrix stack
- **Usage:**

  void glPushMatrix( );

  void glPopMatrix( );

glShadeModel ( ) ;

- Select flat or smooth shading
- **Usage:**

  void glShadeModel(GLenum mode);
- **Parameter:**

  Mode - Specifies a symbolic value representing a shading technique.

The following are the 3D objects that are created using the API's present in the utility toolkit :

glutSolidCube():

Render a solid cube.

void glutSolidCube(GLdouble *size*);

3.3 Source 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[8][3]={
{0.0,0.0,0.0},  //front bottom-left
{1.0,0.0,0.0},  //front bottom-right
{1.0,1.0,0.0},  //front top-right
{0.0,1.0,0.0},  //front top-left
{0.0,0.0,1.0},  //back bottom-left
{1.0,0.0,1.0},  //back bottom-right
{1.0,1.0,1.0},  //back top-right
{0.0,1.0,1.0}   //back top-left
};
GLfloat divi=1.0,scale=1.0,v=0.5,nd=2, ik=0.0,m=0.0,h,c=0.0;
GLint il=10,projection=0,flag[5],i=0,elseflag=0,lighting=0;
//projection is the no. of times m is pressed
//flag[] is used to select iteration nos.
//nd is used to select the no. of divisions
/*elseflag is used to restrict the original cube to be
displayed only once followed by its divided constituents when using 'm'*/

GLfloat amb[]={0.7,0.7,0.7,1.0};
GLfloat diff[]={1.0,1.0,1.0,1.0};
GLfloat spec[]={1.0,1.0,1.0,1.0};
GLfloat shininess[]={50.0};
GLfloat li[]={1.0,1.0,1.0,1.0};
GLfloat lp[]={-15.0,15.0,3.0,0.0};
GLfloat lp1[]={45.0,15.0,3.0,0.0};

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(void)
{
    polygon(0,3,2,1);
    polygon(2,3,7,6);
    polygon(4,5,6,7);
    polygon(0,4,7,3);
    polygon(0,1,5,4);
    polygon(1,2,6,5);
}



void di(GLfloat x,GLfloat y,GLfloat z,GLfloat scale,GLfloat divi,int il,GLfloat v,int col,GLfloat ik,GLint
i)
{
/* This function is a replica of divide() but is used to call
original cube in each iteration only once*/
GLfloat m;

  if(projection==1)
    m=scale/2.0;
else
    m=scale/2.0+v+ik;
   glPushMatrix();// topleft
 if(il==10)
     glColor3fv(colors[3]);
else
    glColor3fv(colors[col]);
    glTranslatef(x-m,y+m,-z);
    glutSolidCube(divi);
    glPopMatrix();

    glPushMatrix();// topright
    if(il==10)
     glColor3fv(colors[2]);
    else
    glColor3fv(colors[col]);
    glTranslatef(x+m,y+m,-z);
    glutSolidCube(divi);
    glPopMatrix();

    glPushMatrix();// bottomleft
    if(il==10)
```

```
    glColor3fv(colors[0]);
   else
   glColor3fv(colors[col]);
   glTranslatef(x-m,y-m,-z);
   glutSolidCube(divi);
   glPopMatrix();

   glPushMatrix();// bottomright
   if(il==10)
    glColor3fv(colors[1]);
   else
   glColor3fv(colors[col]);
   glTranslatef(x+m,y-m,-z);
   glutSolidCube(divi);
   glPopMatrix();

}


void divide(GLfloat x,GLfloat y,GLfloat z,GLfloat scale,GLfloat divi,int il,GLfloat v,int col,GLfloat
ik,GLint i)
{
GLfloat m;//should declare m again!
GLfloat c;
//x,y are centre of the cube to which we translate it to
//scale is used to give the lenght of cube
//divi is the size of the cube
//col is the index needed to be given to colors[]
//i is the iteration no.
/*v is the amount of spacing/projection onto the next
divided cube-set*/
//il is used to recongnise the no. of iterations to be given
//ik=-v initially and incrementally goes towards 0

h=v;

 if(projection==0)
 {
  m=scale/2.0+v+ik;

   glPushMatrix();// topleft
   if(il==10)
    glColor3fv(colors[3]);
   else
   glColor3fv(colors[col]);
   glTranslatef(x-m,y+m,-z);
   glutSolidCube(divi);
   glPopMatrix();

   glPushMatrix();// topright
```

```
   if(il==10)
    glColor3fv(colors[2]);
   else
   glColor3fv(colors[col]);
   glTranslatef(x+m,y+m,-z);
   glutSolidCube(divi);
   glPopMatrix();

   glPushMatrix();// bottomleft
   if(il==10)
    glColor3fv(colors[0]);
   else
   glColor3fv(colors[col]);
   glTranslatef(x-m,y-m,-z);
   glutSolidCube(divi);
   glPopMatrix();

   glPushMatrix();// bottomright
   if(il==10)
    glColor3fv(colors[1]);
   else
   glColor3fv(colors[col]);
   glTranslatef(x+m,y-m,-z);
   glutSolidCube(divi);
   glPopMatrix();

c=m;
if(il>=nd && il<=20)
{

 if(il==10)
 {
   divide(x-m,y+m,z+1.0,scale/2,divi/2,il/2,v/2,3,0.0,i+1); //tl
   divide(x+m,y+m,z+1.0,scale/2,divi/2,il/2,v/2,2,0.0,i+1); //tr
   divide(x-m,y-m,z+1.0,scale/2,divi/2,il/2,v/2,0,0.0,i+1); //bl
   divide(x+m,y-m,z+1.0,scale/2,divi/2,il/2,v/2,1,0.0,i+1); //br
 }

 else
 {
   divide(x-m,y+m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,0.0,i+1); //tl
   divide(x+m,y+m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,0.0,i+1); //tr
   divide(x-m,y-m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,0.0,i+1); //bl
   divide(x+m,y-m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,0.0,i+1); //br
 }
}
}

else if(projection>=0)
 {
    if(projection==1)
```

```
    m=scale/2.0;
     else m=scale/2.0+v+ik;

     if(flag[0]==1)
     {
    glPushMatrix();// topleft
    if(il==10)
     glColor3fv(colors[3]);
    else
    glColor3fv(colors[col]);
    glTranslatef(x-m,y+m,-z);
    glutSolidCube(divi);
    glPopMatrix();

    glPushMatrix();// topright
    if(il==10)
     glColor3fv(colors[2]);
    else
    glColor3fv(colors[col]);
    glTranslatef(x+m,y+m,-z);
    glutSolidCube(divi);
    glPopMatrix();

    glPushMatrix();// bottomleft
    if(il==10)
     glColor3fv(colors[0]);
    else
    glColor3fv(colors[col]);
    glTranslatef(x-m,y-m,-z);
    glutSolidCube(divi);
    glPopMatrix();

    glPushMatrix();// bottomright
    if(il==10)
     glColor3fv(colors[1]);
    else
    glColor3fv(colors[col]);
    glTranslatef(x+m,y-m,-z);
    glutSolidCube(divi);
    glPopMatrix();
     }

if(flag[1]==1)
{

  if(il==10)
    {
    di(x-m,y+m,z+1.0,scale/2,divi/2,il/2,v/2,3,ik,i+1); //tl
    di(x+m,y+m,z+1.0,scale/2,divi/2,il/2,v/2,2,ik,i+1); //tr
    di(x-m,y-m,z+1.0,scale/2,divi/2,il/2,v/2,0,ik,i+1); //bl
    di(x+m,y-m,z+1.0,scale/2,divi/2,il/2,v/2,1,ik,i+1); //br
```

```
   }

else
 {
   di(x-m,y+m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //tl
   di(x+m,y+m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //tr
   di(x-m,y-m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //bl
   di(x+m,y-m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //br
 }
}

if(flag[2]==1)
{
  if(i==0)
  {
    if(il==10)
    {
   divide(x-m,y+m,z+1.0,scale/2,divi/2,il/2,v/2,3,ik,i+1); //tl
   divide(x+m,y+m,z+1.0,scale/2,divi/2,il/2,v/2,2,ik,i+1); //tr
   divide(x-m,y-m,z+1.0,scale/2,divi/2,il/2,v/2,0,ik,i+1); //bl
   divide(x+m,y-m,z+1.0,scale/2,divi/2,il/2,v/2,1,ik,i+1); //br
    }

else
{
   divide(x-m,y+m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //tl
   divide(x+m,y+m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //tr
   divide(x-m,y-m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //bl
   divide(x+m,y-m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //br
}
}

else if(i==1)
{
  if(il==10)
 {
   di(x-m,y+m,z+1.0,scale/2,divi/2,il/2,v/2,3,ik,i+1); //tl
   di(x+m,y+m,z+1.0,scale/2,divi/2,il/2,v/2,2,ik,i+1); //tr
   di(x-m,y-m,z+1.0,scale/2,divi/2,il/2,v/2,0,ik,i+1); //bl
   di(x+m,y-m,z+1.0,scale/2,divi/2,il/2,v/2,1,ik,i+1); //br
 }

 else
 {
   di(x-m,y+m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //tl
   di(x+m,y+m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //tr
   di(x-m,y-m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //bl
   di(x+m,y-m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //br
 }
}
}
```

```
if(flag[3]==1)
{
  if(i==0)
  {
   if(il==10)
    {
   divide(x-m,y+m,z+1.0,scale/2,divi/2,il/2,v/2,3,ik,i+1); //tl
   divide(x+m,y+m,z+1.0,scale/2,divi/2,il/2,v/2,2,ik,i+1); //tr
   divide(x-m,y-m,z+1.0,scale/2,divi/2,il/2,v/2,0,ik,i+1); //bl
   divide(x+m,y-m,z+1.0,scale/2,divi/2,il/2,v/2,1,ik,i+1); //br
    }

else
{
   divide(x-m,y+m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //tl
   divide(x+m,y+m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //tr
   divide(x-m,y-m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //bl
   divide(x+m,y-m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //br
}

}
else if(i==1)
{
if(il==10)
{
   divide(x-m,y+m,z+1.0,scale/2,divi/2,il/2,v/2,3,ik,i+1); //tl
   divide(x+m,y+m,z+1.0,scale/2,divi/2,il/2,v/2,2,ik,i+1); //tr
   divide(x-m,y-m,z+1.0,scale/2,divi/2,il/2,v/2,0,ik,i+1); //bl
   divide(x+m,y-m,z+1.0,scale/2,divi/2,il/2,v/2,1,ik,i+1); //br
}

else
{
   divide(x-m,y+m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //tl
   divide(x+m,y+m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //tr
   divide(x-m,y-m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //bl
   divide(x+m,y-m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //br
}
}

}
if(i==2)
{
 if(il==10)
  {
   di(x-m,y+m,z+1.0,scale/2,divi/2,il/2,v/2,3,ik,i+1); //tl
   di(x+m,y+m,z+1.0,scale/2,divi/2,il/2,v/2,2,ik,i+1); //tr
   di(x-m,y-m,z+1.0,scale/2,divi/2,il/2,v/2,0,ik,i+1); //bl
   di(x+m,y-m,z+1.0,scale/2,divi/2,il/2,v/2,1,ik,i+1); //br
  }
```

```
else
{
   di(x-m,y+m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //tl
   di(x+m,y+m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //tr
   di(x-m,y-m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //bl
   di(x+m,y-m,z+1.0,scale/2,divi/2,il/2,v/1.8,col,ik,i+1); //br
}
}
}
}

static GLfloat theta[]={0.0,0.0,0.0};
static GLint axis=2;

void display()
{
   glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
   glClearColor(1.0,1.0,1.0,1.0);
   glMaterialfv(GL_FRONT,GL_AMBIENT,amb);
   glMaterialfv(GL_FRONT,GL_DIFFUSE,diff);
   glMaterialfv(GL_FRONT,GL_SPECULAR,spec);
   glMaterialfv(GL_FRONT,GL_SHININESS,shininess);
   glColorMaterial(GL_FRONT,GL_DIFFUSE);
   glLightfv(GL_LIGHT0,GL_POSITION,lp);
   glLightfv(GL_LIGHT0,GL_DIFFUSE,li);

   glLoadIdentity();

     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);

     glRotatef(theta[3],-1.0,0.0,0.0);
     glRotatef(theta[4],0.0,-1.0,0.0);
     glRotatef(theta[5],0.0,0.0,-1.0);

   colorcube();
   if(nd>0 && projection==0)
   divide(0.0,0.0,1.75,scale,1.0,il,v,0,0.0,0);


   else if(projection>0)
   {
   if(flag[0]==1)
   {
   nd=25;
   divide(0.0,0.0,1.75,scale,1.0,il,v,0,ik,0);
   }
   if(flag[1]==1)
   {
```

```
    if(elseflag)
    {
    di(0.0,0.0,1.75,scale,1.0,il,v,0,ik,0);
    elseflag=0;
    }
    nd=10;
    divide(0.0,0.0,1.75,scale,1.0,il,v,0,ik,0);
    }
    if(flag[2]==1)
    {
    nd=5;
    if(elseflag)
    {
    nd=10;
    divide(0.0,0.0,1.75,scale,1.0,il,v,0,ik,0);
    elseflag=0;}
    divide(0.0,0.0,1.75,scale,1.0,il,v,0,ik,0);}

    if(flag[3]==1)
    {
    nd=2;
    if(elseflag==1)
    {
    nd=5;
    divide(0.0,0.0,1.75,scale,1.0,il,v,0,ik,0);
    elseflag=0;
    }
    divide(0.0,0.0,1.75,scale,1.0,il,v,0,ik,0);}

    if(flag[4]==1)
    {
    nd=2;
    projection=0;
    divide(0.0,0.0,1.75,scale,1.0,il,v,0,ik,0);
    }
    }
    glFlush();
    glutSwapBuffers();
}


void spincube()
{
    theta[axis]+=1.0;
    if(theta[axis]>360.0) theta[axis]-=360.0;
    display();
    glutPostRedisplay();
}

void myr(int w,int h)
{
```

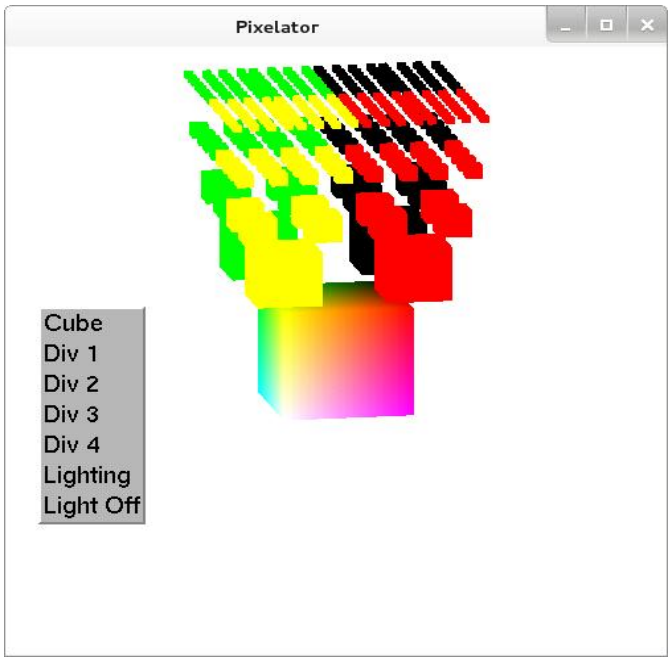
```
   glViewport(0,0,w,h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if(w<=h)
   //glOrtho(-2.0,2.0,-2.0*(GLfloat)h/(GLfloat)w,2.0*(GLfloat)h/(GLfloat)w,-10.0,10.0);
   //glOrtho(-2.0*(GLfloat)w/(GLfloat)h,2.0*(GLfloat)w/(GLfloat)h,-2.0,2.0,-10.0,10.0);
   glOrtho(-5.0,5.0,-5.5*(GLfloat)h/(GLfloat)w,5.5*(GLfloat)h/(GLfloat)w,-10.0,10.0);
   else
   glOrtho(-5.0*(GLfloat)w/(GLfloat)h,5.0*(GLfloat)w/(GLfloat)h,-5.5,5.5,-10.0,10.0);

   glMatrixMode(GL_MODELVIEW);

}

void keys(unsigned char key,int x,int y)
{
   if(key=='x') axis=0;
   if(key=='y') axis=1;
   if(key=='z') axis=2;
   if(key=='X') axis=3;
   if(key=='Y') axis=4;
   if(key=='Z') axis=5;

   display();

   if(key=='m') {
   projection+=1;
   if(projection>0 && projection<=4)
   flag[0]=1;
   else flag[0]=0;
   if(projection>=5 && projection<=7)
    flag[1]=1;
   else flag[1]=0;
   if(projection>=8 && projection<=10)
   flag[2]=1;

   else flag[2]=0;
   if(projection>=11)
   flag[3]=1;
   else flag[3]=0;
   if(projection==13)
   flag[4]=1;
   else flag[4]=0;
   if(projection==5||projection==8||projection==11)
   {elseflag=1;
   ik=-c-0.2;
   //ik=-c;
   if(projection==11||projection==8)
   ik=-c-0.05;
   }
   else elseflag=0;
```

```
    display();
    ik+=h/5.0;
    }

    if(key=='n')
    { projection-=1;

    display();
    ik-=h/5.0;
    }

}

void lighti()
{
     if(lighting==1)
    {
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glShadeModel(GL_SMOOTH);
    glEnable(GL_NORMALIZE);
    glEnable(GL_COLOR_MATERIAL);
    }
     else
      glDisable(GL_LIGHTING);
}

void mymenu(int id)
{

  switch(id)
  {
    case 0: nd=-1;
    glutIdleFunc(spincube);
    break;
    case 1: nd=25;
    glutIdleFunc(spincube);
    break;
    case 2: nd=10;
    glutIdleFunc(spincube);
    break;
    case 3: nd=5;
    glutIdleFunc(spincube);
    break;
    case 4: nd=2;
    glutIdleFunc(spincube);
    break;
    case 5: lighting=1;
    lighti();
    break;
```

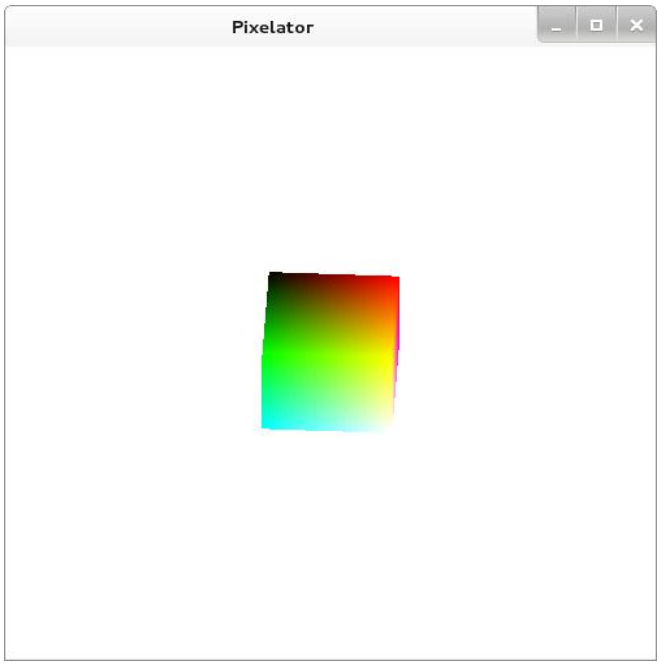
```
    case 6: lighting=0;
    lighti();
    break;
  }
}

int main(int argc,char **argv)
{
      ik=ik-v;
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
    glutInitWindowSize(500,500);
    glutCreateWindow("Pixelator");
    glutReshapeFunc(myr);
    glutDisplayFunc(display);

    glutCreateMenu(mymenu);
    glutAddMenuEntry("Cube",0);
    glutAddMenuEntry("Div 1",1);
    glutAddMenuEntry("Div 2",2);
    glutAddMenuEntry("Div 3",3);
    glutAddMenuEntry("Div 4",4);
    glutAddMenuEntry("Lighting",5);
    glutAddMenuEntry("Light Off",6);
    glutAttachMenu(GLUT_RIGHT_BUTTON);

    glutIdleFunc(spincube);
    glutKeyboardFunc(keys);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}
```

CHAPTER 4

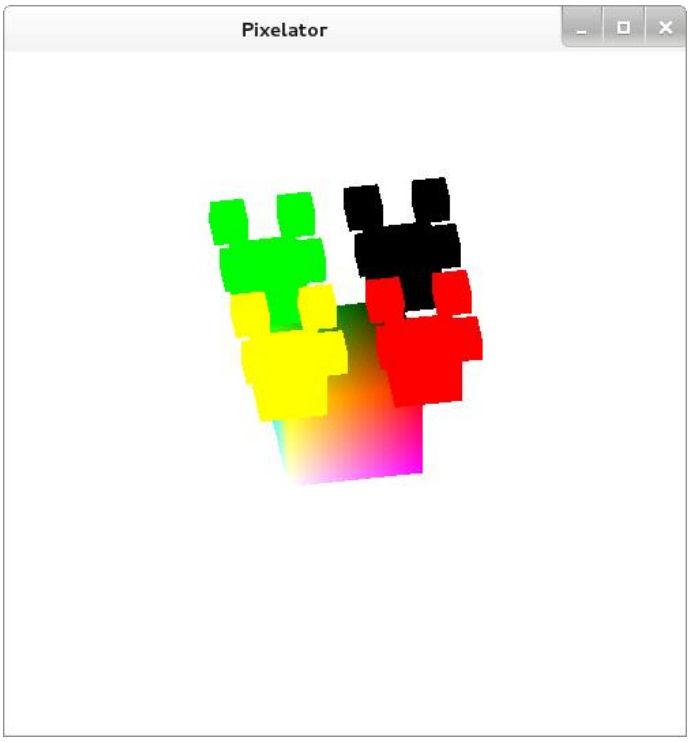
SCREENSHOTS



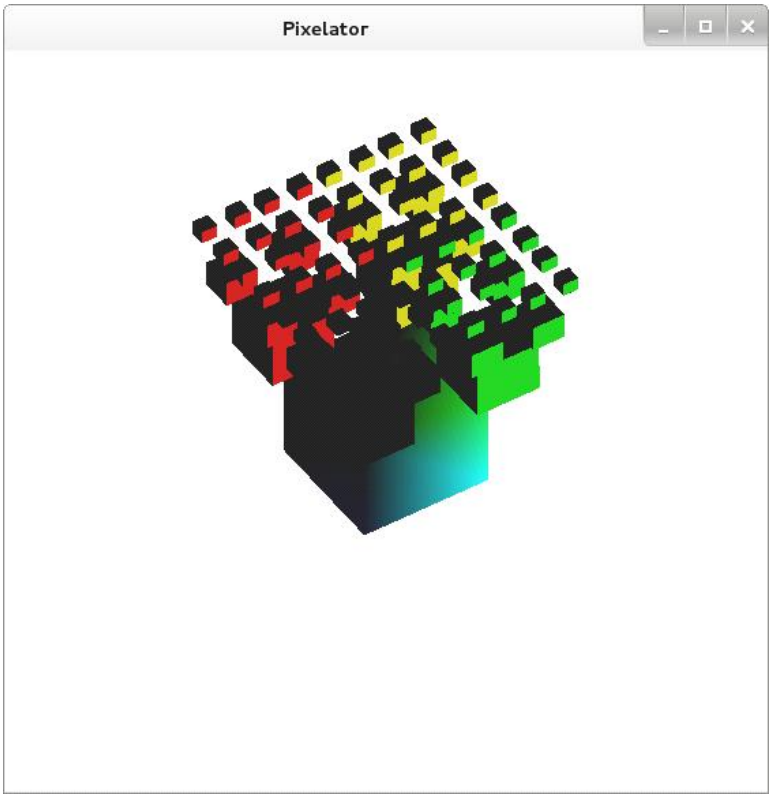
Snapshot 4.1: Entire display of cubes with menu



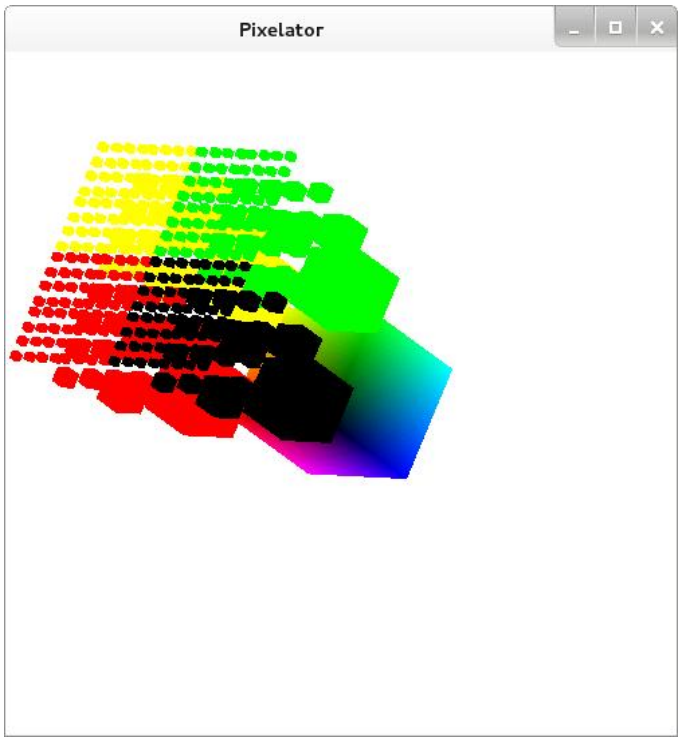
Snapshot 4.2: The cube



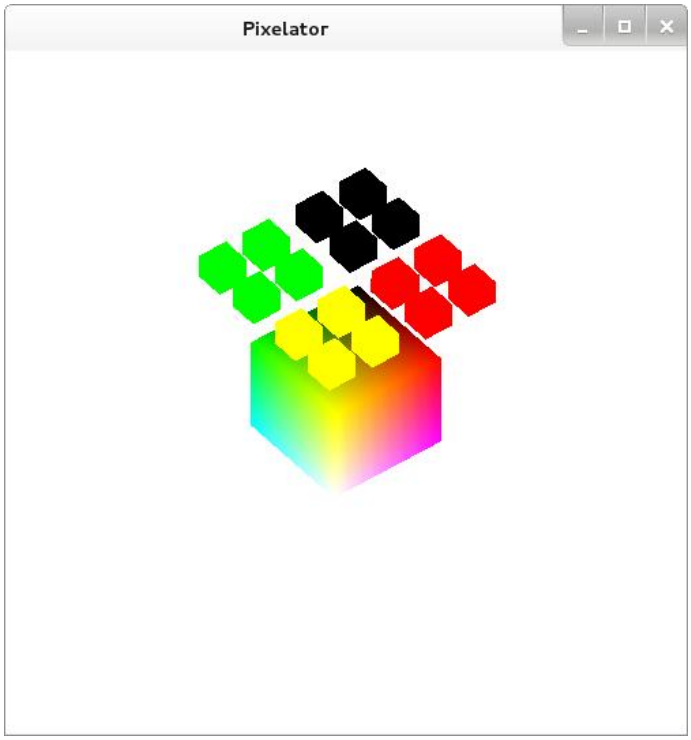
Snapshot 4.3: Different divisions of cube selected from menu

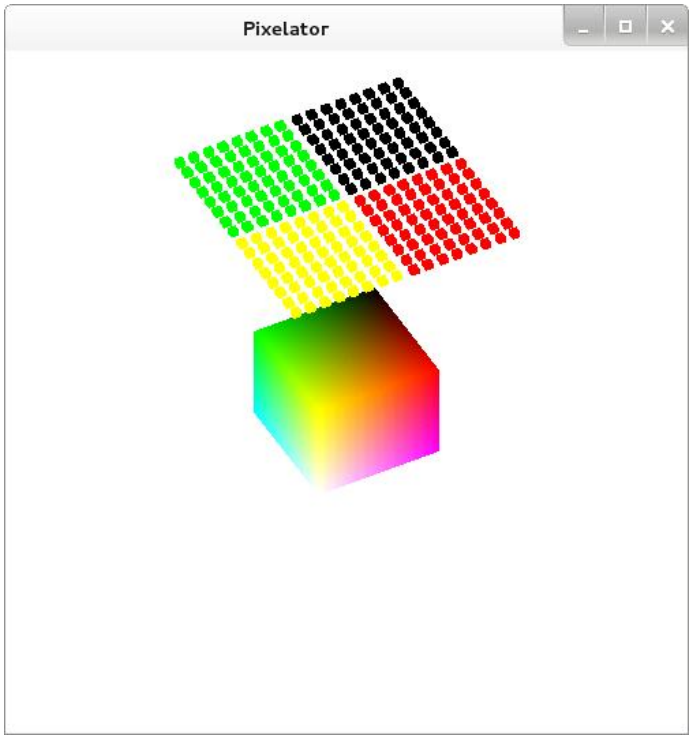


Snapshot 4.4: Lighting enabled

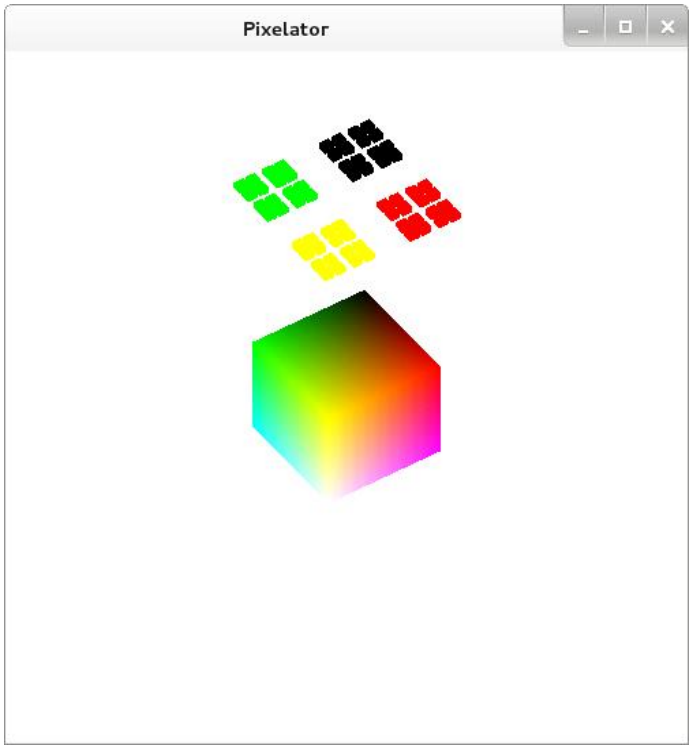


Snapshot 4.5: Rotation about different axis





Snapshot 4.6: Using 'm' to show the division



Snapshot 4.5: Using 'n' to show the cubes integrating into one

CHAPTER 5

CONCLUSION

5.1 Overview

An attempt has been made to develop an OpenGL graphics package which meets all the necessary requirements that were set out. The aim in developing this project was to demonstrate the formation of objects using the smallest unit of display,called pixels and to depict the effect of lighting.

This program was designed using Open GL application software by applying the skills learnt in class, and in doing so, to understand the algorithms and techniques. It is extremely user friendly. The interfaces are keyboard and mouse driven and hence even a user with minimal knowledge of these devices can run the program. I finally conclude that this graphics package satisfies all requirements and provides good entertainment.

5.2 Future Enhancements

These are the features that are planned to be supported in the future

- Support for multiple layer modelling
- Support for color blending
- Support for enhanced 3d transformations
- Support for transparency of layers

CHAPTER 6

BIBLIOGRAPHY

- **Edward Angel**, "Interactive Computer Graphics",5th edition, Pearson Education,2005.
- **Donald D Hearn** and **M. Pauline Baker,** " Computer Graphics with OpenGL", 3rd Edition "OpenGL Programming Guide", Addision-Wesley Publishing Company.