Experiment No:-8

Color Combination

**CONTENTS**

|  |  |
| --- | --- |
| **SR.NO** | **Particulars** |
| 1. | Abstract |
| 2. | System Specifications |
| 3. | Introduction to openGL |
| 4. | Implementation |
| 5. | Interaction |
| 6. | Source Code |
| 7. | Output |
| 8. | Conclusion |
| 9. | Bibliography |

# Abstract

* Main aim of this Mini Project is to illustrate the concepts of color combination and usage of OpenGL library.
* Our project demonstrates what happens when the primary colors i.e Red, Green and Blue are combined.
* All the permutations and combinations of colors are taken care of.
* We have used input devices like mouse and key board to interact with Program.

# Introduction to openGL

As a software interface for graphics hardware, OpenGL&#39; s main purpose is to render two- and three-dimensional objects into a frame buffer.

These objects are described as sequences of vertices or pixels.

OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer.

## OpenGL Fundamentals

This section explains some of the concepts inherent in OpenGL. Primitives and Commands

OpenGL draws primitives—points, line segments, or polygons—subject to several selectable modes.

You can control modes independently of each other; that is, setting one mode doesn&#39;t affect whether other modes are set .Primitives are specified, modes are set, and other OpenGL operations are described by issuing commands in the form of function calls.

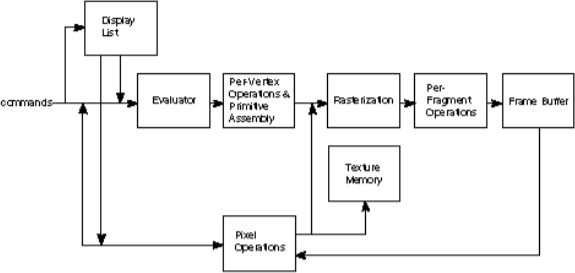
Primitives are defined by a group of one or more vertices. A vertex defines a point, an endpoint of a line, or a corner of a polygon where two edges meet. Data is associated with a vertex, and each vertex and its associated data are processed independently, in order, and in the same way. The type of clipping depends on which primitive the group of vertices represents.

Commands are always processed in the order in which they are received, although there may be an indeterminate delay before a command takes effect. This means that each primitive is drawn completely before any subsequent command takes effect. It also means that state-querying commands return data that&#39;s consistent with complete execution of all previously issued OpenGL commands.

## Basic OpenGL Operation

The figure shown below gives an abstract, high-level block diagram of how OpenGL processes data. In the diagram, commands enter from the left and proceed through what can be thought of as a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during the various processing stages.

**Figure . OpenGL Block Diagram**



As shown by the first block in the diagram, rather than having all commands proceed immediately through the pipeline, you can choose to accumulate some of them in a display list for processing at a later time.

Rasterization produces a series of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon.

Each fragment so produced is fed into the last stage,

per-fragment operations, which performs the final operations on the data before it&#39;s stored as pixels in the frame buffer. These operations include conditional updates to the frame buffer based on incoming and previously stored z-value s (for z-buffering) and blending of incoming pixel colors with stored colors, as well as masking and other logical operations on pixel values.

All elements of OpenGL state, including the contents of the texture memory and even of the frame buffer, can be obtained by an OpenGL application.

# Interaction with program

* This program includes interaction through keyboard.
* S :- Start the Project
* R :- Toggle Red Light
* G :- Toggle Green Light
* B :- Toggle Blue Light
* Q :- Quit

# Source Code

/\*An Interactive Program to create 3d objects\*/

#include <windows.h> #include<string.h> #include<stdarg.h> #include<stdio.h> #include <glut.h> static double x=0.0; static float red1=0;

static float green1=0; static float blue1=0; static float help1=1;

void stroke\_output(GLfloat x, GLfloat y, char \*format,...)

{

va\_list args;

char buffer[200], \*p; va\_start(args, format); vsprintf(buffer, format, args); va\_end(args); glPushMatrix();

glTranslatef(-2.5, y, 0);

glScaled(0.003, 0.005, 0.005);

for (p = buffer; \*p; p++) glutStrokeCharacter(GLUT\_STROKE\_ROMAN, \*p);

glPopMatrix();

}

static float green1=0; static float blue1=0; static float help1=1;

void stroke\_output(GLfloat x, GLfloat y, char \*format,...)

{

va\_list args;

char buffer[200], \*p; va\_start(args, format); vsprintf(buffer, format, args); va\_end(args); glPushMatrix();

glTranslatef(-2.5, y, 0);

glScaled(0.003, 0.005, 0.005);

for (p = buffer; \*p; p++) glutStrokeCharacter(GLUT\_STROKE\_ROMAN, \*p); glPopMatrix();

}

static float green1=0; static float blue1=0; static float help1=1;

void stroke\_output(GLfloat x, GLfloat y, char \*format,...)

{

va\_list args;

char buffer[200], \*p; va\_start(args, format); vsprintf(buffer, format, args); va\_end(args); glPushMatrix();

glTranslatef(-2.5, y, 0);

glScaled(0.003, 0.005, 0.005);

for (p = buffer; \*p; p++)

glutStrokeCharacter(GLUT\_STROKE\_ROMAN, \*p); glPopMatrix();

}

static float green1=0; static float blue1=0; static float help1=1;

void stroke\_output(GLfloat x, GLfloat y, char \*format,...)

{

va\_list args;

char buffer[200], \*p;

va\_start(args, format); vsprintf(buffer, format, args);

va\_end(args);

glPushMatrix(); glTranslatef(-2.5, y, 0);

glScaled(0.003, 0.005, 0.005);

for (p = buffer; \*p; p++) glutStrokeCharacter(GLUT\_STROKE\_ROMAN, \*p); glPopMatrix();

}

void doInit()

{

/\* Background and foreground color \*/ glClearColor(0.5,0.5,0.5,0.0);

glColor3f(.0,1.0,1.0); glViewport(0,0,640,480);

/\* Select the projection matrix and reset it then setup our view perspective \*/ glMatrixMode(GL\_PROJECTION); glLoadIdentity();

gluPerspective(30.0f,(GLfloat)640/(GLfloat)480,0.1f,200.0f);

/\* Select the modelview matrix, which we alter with rotatef() \*/ glMatrixMode(GL\_MODELVIEW);

glLoadIdentity(); glClearDepth(2.0f); glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_COLOR\_MATERIAL); glDepthFunc(GL\_LEQUAL);

}

void torch(){

glPushMatrix(); glutSolidSphere(0.8,50,50); glPopMatrix();

glPushMatrix(); glRotatef(90,0,1,0); glScaled(0.5,0.5,3); glColor3f(0,1,1); glutSolidTorus(0.4,1.5,50,50); glPopMatrix();

glPushMatrix(); glTranslatef(-1.5,0,0); glRotatef(90,0,1,0); glScaled(0.7,0.7,1.5);

glutSolidTorus(0.4,1.5,50,50); glPopMatrix();

}

void help(){

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glPushMatrix(); glScaled(0.7,0.7,0.7);

stroke\_output(-2,3,&quot;H &gt; Toggle Help&quot;);

stroke\_output(-2,2,&quot;R -----&gt; Toggle Red Light&quot;); stroke\_output(-2,1,&quot;G -----&gt; Toggle Green Light&quot;); stroke\_output(-2,0,&quot;B -----&gt; Toggle Blue Light&quot;);

glPopMatrix();

glFlush(); glutSwapBuffers();

}

void draw(){

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glLoadIdentity(); glTranslatef(0.0f,0.0f,-13.0f);

glPushMatrix(); glTranslatef(0,0,-15); glScaled(1,1,0.1);

glColor3f(0.3,0.3,0); glutSolidCube(9); glPopMatrix();

glPushMatrix(); glColor3f(1,0,1); glutSolidCone(4.5,15,40,40); glPopMatrix();

// Color Sphere

glPushMatrix(); glTranslatef(0,0,-14); glScaled(1,1,0.1);

glColor3f(red1,green1,blue1);

glutSolidSphere(2,30,30); glPopMatrix();

// red Torch glPushMatrix(); glTranslatef(-1.5,-2,2);

glRotatef(-20,0,0,1); glPushMatrix(); glRotatef(90,0,1,0); glScaled(0.3,0.3,0.3); glColor3f(1,0,0); torch(); glPopMatrix();

glPopMatrix(); if(red1)

{

glPushMatrix(); glRotatef(50,1,0,1); glRotatef(-55,0,1,1); glColor3f(1,0,0); glutWireCone(1.0,3,10,10); glPopMatrix();

}

// Green Torch glPushMatrix(); glTranslatef(0,-2,2); glRotatef(-20,0,0,1); glPushMatrix(); glRotatef(90,0,1,0); glScaled(0.3,0.3,0.3); glColor3f(0,1,0); torch(); glPopMatrix(); glPopMatrix();

if(green1)

{

glPushMatrix(); glTranslatef(0,-0.5,-3); glRotatef(10,1,0,1); glColor3f(0,1,0); glutWireCone(1.0,7,10,10); glPopMatrix();

}

// Blue Torch glPushMatrix();

glTranslatef(1.5,-2,2);

glRotatef(-20,0,0,1); glPushMatrix(); glRotatef(90,0,1,0); glScaled(0.3,0.3,0.3); glColor3f(0,0,1); torch(); glPopMatrix(); glPopMatrix();

if(blue1)

{

glPushMatrix(); glRotatef(90,1,0,1); glColor3f(0,0,1); glutWireCone(1.0,3,10,10); glPopMatrix();

}

glFlush(); glutSwapBuffers();

}

void doDisplay()

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glLoadIdentity(); glTranslatef(0.0f,0.0f,-13.0f);

// Write your Own Code Here if(help1)

help();

else draw();

GLfloat mat\_ambient[ ]={0.0f,1.0f,2.0f,1.0f}; GLfloat mat\_diffuse[ ]={0.0f,1.5f,.5f,1.0f}; GLfloat mat\_specular[ ]={5.0f,1.0f,1.0f,1.0f}; GLfloat mat\_shininess[ ]={100.0f};

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[ ]={3.7f,0.7f,0.7f,1.0f}; Orange GLfloat light\_position[ ]={2.0f,5.0f,3.0f,1.0f};\*/

/\*light source properties\*/

GLfloat lightIntensity[ ]={1.7f,1.7f,1.7f,1.0f}; GLfloat light\_position[ ]={2.0f,0.0f,0.0f,0.0f};

glLightfv(GL\_LIGHT0,GL\_POSITION,light\_position); GLfloat light\_position2[ ]={0.0f,0.0f,8.0f,0.0f};

glLightfv(GL\_LIGHT0,GL\_POSITION,light\_position2); GLfloat light\_position3[]={6.0f,0.0f,5.0f,0.0f};

glLightfv(GL\_LIGHT0,GL\_POSITION,light\_position3); glLightfv(GL\_LIGHT0,GL\_DIFFUSE,lightIntensity);

glFlush(); glutSwapBuffers();

}

void mykey(unsigned char key,int x,int y)

{

if(key==’q’||key==’Q’)

{

exit(0);

}

if(key==’h’||key==’H’)

{

help1=!help1; glutPostRedisplay();

}

if(key==’s’ || key==’S’)

{

glutIdleFunc(draw);

}

if(key=’r’||key==’R)

{

red1=!red1 ; glutPostRedisplay();

}

if(key==’g’||key==’G’)

{

green1=!green1; glutPostRedisplay();

}

if(key==’b’||key==’B’)

{

blue1=!blue1; glutPostRedisplay();

}

}

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

{

glutInit(&amp;argc, argv); glutInitDisplayMode(GLUT\_DOUBLE|GLUT\_RGB); glutInitWindowSize(640,480); glutInitWindowPosition(0,0);

glutCreateWindow(&quot;Basic Structures Orientation&quot;); glutDisplayFunc(doDisplay);

glEnable(GL\_LIGHTING); glEnable(GL\_LIGHT0); glShadeModel(GL\_SMOOTH); glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_NORMALIZE);

glutKeyboardFunc(mykey); doInit();

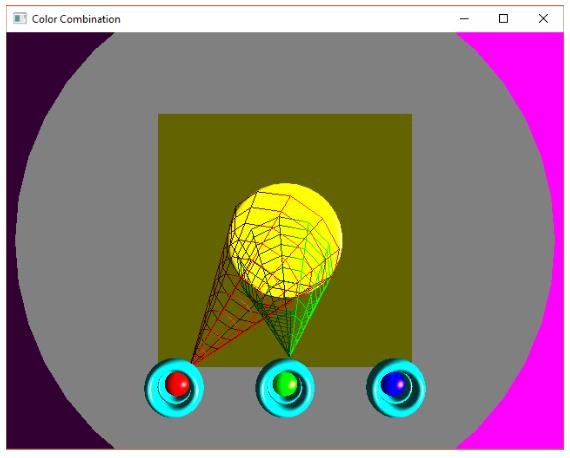
glutMainLoop(); return 0;

}

# OUTPUT OF THE PROGRAM









**Conclusion**

The project “Color Combination” clearly demonstrates the use of OpenGL functions and what happens when primary colors are mixed.

Finally we conclude that this program clearly illustrate the color combination using openGL and has been completed successfully and is ready to be demonstrate.

**Submitted by:- MANASI.RENUSE**