LABORATOR 5

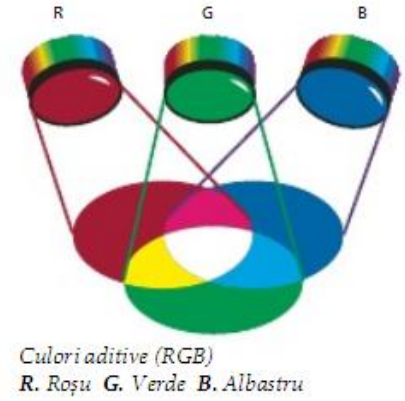
Specificarea culorii în OpenGL

În acest laborator se vor studia modelele de culoare in OpenGL

Se numeкte culoare percepŞia de cŁtre [ochi] a uneia sau a mai multor frecvenŞe (sau lungimi de undŁ) de luminŁ. În cazul oamenilor aceastŁ percepŞie provine din abilitatea ochiului de a distinge câteva (de obicei trei) analize filtrate diferite ale aceleia кi imagini. PercepŞia culorii este influenŞatŁ de biologie (de ex. unii oameni se nasc vŁzând culorile diferit, alŞi nu le percep deloc, vezi daltonism), de evoluŞia aceluia кi observator sau кi de culorile aflate în imediata apropiere a celei percepute (aceasta fiind explicaŞia multor iluzii optice).

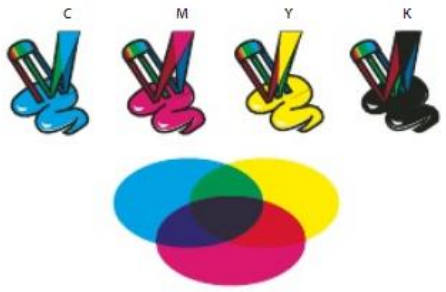
Culori primare aditive si subversive

Culorile primare aditive sunt cele trei culori ale luminii (roкu, verde кi albastru) care produc toate culorile din spectrul vizibil dacŁ sunt luate împreunŁ în diverse combinaŞii. DacŁ adŁugaŞi pŁrŞi egale de roкu, albastru кi verde, veŞi obŞine alb. AbsenŞa completŁ a luminii roкi, albastre кi negre, va genera negru. Monitoarele calculatoarelor sunt dispozitivecare utilizeazŁ culorile primare aditive pentru a crea culoarea.



Culori aditive (RGB)
***R.** Roşu **G.** Verde **B.** Albastru*

Figura 1



Culori substractive (CMYK)
***C.** Cyan **M.** Magenta **Y.** Galben **K.** Negru*

Figura 2

Culorile primare substractive(figura 2) sunt pigmenŞi care creeazŁ un spectru de culori în diverse combinaŞii. Spre deosebire demonitoare, imprimantele utilizeazŁ culori primare substractive (pigmenŞi cyan, magenta, galben кi negru) pentru aproduce culori prin amestec substractiv. Se utilizeazŁ termenul ñsubstractivò deoarece culorile primare sunt pure pânŁcând sunt amestecate, iar amestecul are ca rezultat culori care sunt versiuni mai puŞin pure ale culorilor primare. De exemplu, portocaliul este creat prin amestecul substractiv dintre magenta кi galben.

Reprezentarea culorilor în sistemele grafice si OpenGL.

Imaginea care se obtine pe display este compusa dintr-un anumit numar de pixeli, dat de rezolutia display-ului. Fiecare pixel are o pozitie pe ecran, data de adresa lui în ferestra de

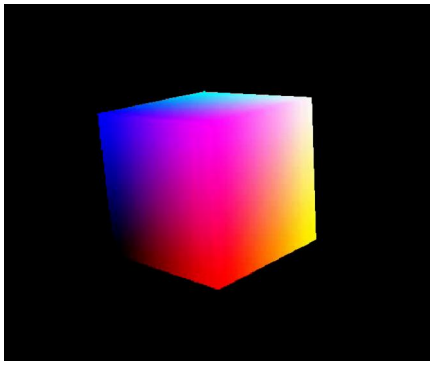
afisare, si o culoare care poate fi reprezentata în mai multe modele: modelul RGB, modelul HSV, modelul HLS, si altele. Dintre aceste modele, în grafica se foloseste cel mai frecvent modelul RGB.

În modelul RGB, culoarea este reprezentata printr-un triplet de culori primare, rosu (red),verde (green), albastru (blue). Utilizarea preponderenta a modelului RGB în grafica se datoreaza în primul rând faptului ca monitoarele color folosesc acest model de producere a culorii. Orice culoare în modelul RGB se exprima printr-un triplet (r,g,b) si îi corespunde un punct în spatiul RGB al carui vector C este:

C = rR + gG + bG

unde R, G, B sunt versorii axelor rosu (red), verde (green), albastru (blue). În acest model culoarea negru este reprezentata prin tripletul (0,0,0), iar culoarea alb este reprezentata prin tripletul (1,1,1).

În sistemele grafice se mai foloseste o varianta a modelului RGB, modelul RGBA, unde cea de-a patra componenta (α) indica transparenta suprafetei. Valoarea 1 a acestei componente (α = 1) înseamna suprafata opaca, iar valoarea minima (α = 0) înseamna suprafata complet transparenta.



Daca transparenta unei suprafete este diferita de zero, atunci culoarea care se atribuie pixelilor acestei suprafete se modifica în functie de culoarea existenta în bufferul de imagine.

În biblioteca OpenGL sunt definite doua modele de culori: modelul de culori RGBA si modelul de culori indexate. În modelul RGBA sunt memorate componentele de culoare R, G, B si transparenta A pentru fiecare primitiva geometrica sau pixel al imaginii. În modelul de culori indexate, culoarea primitivelor geometrice sau a pixelilor este reprezentata printr-un index într-o tabela de culori (color map), care are memorate pentru fiecare intrare (index) componentele corespunzatoare R,G,B,A ale culorii. În modul de culori indexate nu se pot efectua unele dintre prelucrarile grafice importante (cum sunt umbrirea, anti-aliasing, ceata).

Modelul de culori indexate este folosit în principal în aplicatii de proiectare grafica (CAD), în care este necesar un numar mic de culori si nu se folosesc umbrirea, ceata, etc. În aplicatiile de realitate virtuala nu se poate folosi modelul de culori indexate si de aceea în continuare nu vor mai fi prezentate comenzile sau optiunile care se refera la acest model si toate descrierile considera numai modelul RGBA.

Culoarea care se atribuie unui pixel dintr-o primitiva geometrica depinde de mai multe conditii, putând fi o culoare constanta a primitivei, o culoare calculata prin interpolare între culorile vârfurilor primitivei, sau o culoare calculata în functie de iluminare, anti-aliasing si texturare.

Presupunând pentru moment culoarea constanta a unei primitive, aceasta se obtine prin setarea unei variabile de stare a bibliotecii, variabila de culoare curenta (GL_CURRENT_COLOR). Culoarea curenta se seteaza folosind una din functiile glColor#(), care are mai multe variante, în functie de tipul si numarul argumentelor. De exemplu, doua din prototipurile acestei functii defin ite în fisierul gl.h sunt:

```
void glColor3f(GLfloat r, GLfloat g, GLfloat b);
```

```
void glColor4d(GLdouble r, GLdouble g, GLdouble b, GLdouble a);
```

Culoarea se poate specifica prin trei sau patru valori, care corespund componentelor rosu (r), verde (g), albastru (b), respectiv transparenta (α) ca a patra componenta pentru functiile cu 4
argumente.
Maniera cea mai obisnuita de schimbare a culorii este folosirea definitiei RGB:

```
glColor3f(GLfloat red, GLfloat green, GLfloat blue)
```

unde red, green, si blue sunt valori reale cuprinse între 0 si 1. Valoarea 0.0 corespunde unei contributii minime a culorii de baza folosite iar 1.0 unei contributii maxime.
În modul RGBA pentru selectarea culorii curente se folosesc funcŞile glColor#()

FuncŞile **glColor3() specifică un triplet (R, G, B)** vezi Tabelul 1. Valorile R, G, B si A variazŁ în domeniul [0,1] . Valoarea implicitŁ a opacitŁŞi este 1.0. Când sunt specificate valori din afara domeniului [0,1] , acestea se mapeazŁ liniar în acest interval.

Tabelu1 1

Negru glColor3f(1.0, 0.0, 0.0)
Rosu glColor3f(0.0, 1.0, 0.0)
Verde glColor3f(0.0, 0.0, 1.0)
Albastru glColor3f(1.0, 1.0, 0.0)
Galben glColor3f(0.0, 1.0, 1.0)
Cian glColor3f(1.0, 0.0, 1.0)
Magenta glColor3f(1.0, 1.0, 1.0)
Alb glColor3f(0.0, 0.0, 0.0)

InstrucŞiunea `glClear (...)` stabilexte buferul care va fi xters prin culoare specificatŁ în glClearColor, ea are prototipul:

```
void glClear(GLbitfield mask);
```

unde mask poate lua valorile:

- GL_COLOR_BUFFER_BIT;
- GL_DEPTH_BUFFER_BIT;
- GL_ACCUM_BUFFER_BIT;
- GL_STENCIL_BUFFER_BIT.

Pentru primele aplicaţi o vom folosi doar cu masca **GL_COLOR_BUFFER_BIT.**

Exerciţiu rezolvat.
ăn exemplul de mai jos s-a creat o linie de culoare verde.

```
#include <GL/glut.h>// ==> gl.h and glu.h

void init(void)
{
	glClearColor(1.0, 1.0, 1.0, 0.0);

	glMatrixMode(GL_PROJECTION);
	gluOrtho2D(0.0, 200.0, 0.0, 150.0);
}

void lineSegmentGreen(void)
{
	glClear(GL_COLOR_BUFFER_BIT);

	glColor3f(0.0, 1.0, 0.0);
	glBegin(GL_LINES);
	glVertex2i(0, 0);
	glVertex2i(150, 112);
	glEnd();

	glFlush();
}

int main(int argc, char ** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
	glutInitWindowPosition(50, 100);
	glutInitWindowSize(400, 300);
	glutCreateWindow("Exemplu de linie colorata");

	init();  // vom scrie aceasta

	glutDisplayFunc(lineSegmentGreen); // si aceasta
	glutMainLoop();
}
```

Exerciţiu 5-1.
ăn programul de mai jos sunt prezentate colorat , primitive grafice colorate.

```
/*
 * GL02Primitive colorate.cpp: Vertex, Primitive si Culoare
 * Deseneaza figuri 2D colorate: patrat, triunghi si polygon.
 */
#include <windows.h>  // pentru MS Windows
#include <GL/glut.h>  // GLUT, include glu.h si gl.h
```

```
/* Initializarea graficii OpenGL  */
void initGL() {
   // Seteaza "stergerea ecranului" sau culoare de fundal
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Negru si opac
}

/* Apeleaza evenimentul de colorare al ferestrei. */
void display() {
   glClear(GL_COLOR_BUFFER_BIT);   // Sterge culoarea buffer-ului

   // Defineste figurile constituie din contur inchis intre secventele glBegin si
glEnd
   glBegin(GL_QUADS);              // Fiecare set de  4 vertexuri ale patratului
      glColor3f(1.0f, 0.0f, 0.0f); // Rosu
      glVertex2f(-0.8f, 0.1f);     // Defineste vertxurile in sens contrar acelor de
ceasornic(CCW)
      glVertex2f(-0.2f, 0.1f);
      glVertex2f(-0.2f, 0.7f);
      glVertex2f(-0.8f, 0.7f);

      glColor3f(0.0f, 1.0f, 0.0f); // Verde
      glVertex2f(-0.7f, -0.6f);
      glVertex2f(-0.1f, -0.6f);
      glVertex2f(-0.1f,  0.0f);
      glVertex2f(-0.7f,  0.0f);

      glColor3f(0.2f, 0.2f, 0.2f); // gri inchis
      glVertex2f(-0.9f, -0.7f);
      glColor3f(1.0f, 1.0f, 1.0f); // Alb
      glVertex2f(-0.5f, -0.7f);
      glColor3f(0.2f, 0.2f, 0.2f); // gri inchis
      glVertex2f(-0.5f, -0.3f);
      glColor3f(1.0f, 1.0f, 1.0f); // alb
      glVertex2f(-0.9f, -0.3f);
   glEnd();

   glBegin(GL_TRIANGLES);          // Fiecare set de vertexuri ale triunghiului
      glColor3f(0.0f, 0.0f, 1.0f); // Albastru
      glVertex2f(0.1f, -0.6f);
      glVertex2f(0.7f, -0.6f);
      glVertex2f(0.4f, -0.1f);

      glColor3f(1.0f, 0.0f, 0.0f); // Rosu
      glVertex2f(0.3f, -0.4f);
      glColor3f(0.0f, 1.0f, 0.0f); // Verde
      glVertex2f(0.9f, -0.4f);
      glColor3f(0.0f, 0.0f, 1.0f); // Albastru
      glVertex2f(0.6f, -0.9f);
   glEnd();

   glBegin(GL_POLYGON);            // Aceste varfuri formeaza un poligon inchis
      glColor3f(1.0f, 1.0f, 0.0f); // Galben
      glVertex2f(0.4f, 0.2f);
      glVertex2f(0.6f, 0.2f);
      glVertex2f(0.7f, 0.4f);
      glVertex2f(0.6f, 0.6f);
      glVertex2f(0.4f, 0.6f);
      glVertex2f(0.3f, 0.4f);
   glEnd();

   glFlush();  // Incepe randarea
```

```
}

/* Functia principala: GLUT ruleaza ca o consola()  */
int main(int argc, char** argv) {
   glutInit(&argc, argv);          // Initializare GLUT
   glutCreateWindow("Vertex, Primitive & Culoare");  // Creare fereastra cu titlu
   glutInitWindowSize(320, 320);   // Seteaza fereastra initiala
   glutInitWindowPosition(50, 50); // Pozitioneaza fereastra initiala
   glutDisplayFunc(display);       // Inregistreaza functia callback de revopsire a
ferestrei
   initGL();                       // Initializarea
   glutMainLoop();                 // Introdu bucla de prelucrare a evenimentelor
   return 0;
}
```

Exerciţiu 5-2.

ăn programul urmŁtor sunt dispuse colorat , pe linii si pe coloane un set de primitive grafice colorate.

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


#define PIXEL_CENTER(x) ((long)(x) + 0.5)

#define GAP 10
#define ROWS 3
#define COLS 4

#define OPENGL_WIDTH 48
#define OPENGL_HEIGHT 13


GLenum rgb, doubleBuffer;
GLint windW = 600, windH = 300;

GLenum mode1, mode2;
GLint boxW, boxH;
GLubyte OpenGL_bits[] = {
       0x00, 0x03, 0x00, 0x00, 0x00, 0x00,
       0x7f, 0xfb, 0xff, 0xff, 0xff, 0x01,
       0x7f, 0xfb, 0xff, 0xff, 0xff, 0x01,
       0x00, 0x03, 0x00, 0x00, 0x00, 0x00,
       0x3e, 0x8f, 0xb7, 0xf9, 0xfc, 0x01,
       0x63, 0xdb, 0xb0, 0x8d, 0x0d, 0x00,
       0x63, 0xdb, 0xb7, 0x8d, 0x0d, 0x00,
       0x63, 0xdb, 0xb6, 0x8d, 0x0d, 0x00,
       0x63, 0x8f, 0xf3, 0xcc, 0x0d, 0x00,
       0x63, 0x00, 0x00, 0x0c, 0x4c, 0x0a,
       0x63, 0x00, 0x00, 0x0c, 0x4c, 0x0e,
       0x63, 0x00, 0x00, 0x8c, 0xed, 0x0e,
       0x3e, 0x00, 0x00, 0xf8, 0x0c, 0x00,
};
```

```
static void Init(void)
{

	mode1 = GL_TRUE;
	mode2 = GL_TRUE;
}

static void Reshape(int width, int height)
{

	windW = width;
	windH = height;
}

static void RotateColorMask(void)
{
	static GLint rotation = 0;

	rotation = (rotation + 1) & 0x3;
	switch (rotation) {
	case 0:
		glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
		glIndexMask(0xff);
		break;
	case 1:
		glColorMask(GL_FALSE, GL_TRUE, GL_TRUE, GL_TRUE);
		glIndexMask(0xFE);
		break;
	case 2:
		glColorMask(GL_TRUE, GL_FALSE, GL_TRUE, GL_TRUE);
		glIndexMask(0xFD);
		break;
	case 3:
		glColorMask(GL_TRUE, GL_TRUE, GL_FALSE, GL_TRUE);
		glIndexMask(0xFB);
		break;
	}
}

static void SetColor(int index)
{

	if (rgb) {
		switch (index) {
		case 0:
			glColor3f(0.0, 0.0, 0.0);
			break;
		case 1:
			glColor3f(1.0, 0.0, 0.0);
			break;
		case 2:
			glColor3f(0.0, 1.0, 0.0);
			break;
		case 3:
			glColor3f(1.0, 1.0, 0.0);
			break;
		case 4:
			glColor3f(0.0, 0.0, 1.0);
			break;
		case 5:
			glColor3f(1.0, 0.0, 1.0);
			break;
```

```
                case 6:
                        glColor3f(0.0, 1.0, 1.0);
                        break;
                case 7:
                        glColor3f(1.0, 1.0, 1.0);
                        break;
                }
        }
        else {
                glIndexi(index);
        }
}

static void Key(unsigned char key, int x, int y)
{

        switch (key) {
        case '1':
                mode1 = !mode1;
                glutPostRedisplay();
                break;
        case '2':
                mode2 = !mode2;
                glutPostRedisplay();
                break;
        case '3':
                RotateColorMask();
                glutPostRedisplay();
                break;
        case 27:
                exit(0);
        }
}

static void Viewport(GLint row, GLint column)
{
        GLint x, y;

        boxW = (windW - (COLS + 1) * GAP) / COLS;
        boxH = (windH - (ROWS + 1) * GAP) / ROWS;

        x = GAP + column * (boxW + GAP);
        y = GAP + row * (boxH + GAP);

        glViewport(x, y, boxW, boxH);

        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(-boxW / 2, boxW / 2, -boxH / 2, boxH / 2, 0.0, 1.0);
        glMatrixMode(GL_MODELVIEW);

        glEnable(GL_SCISSOR_TEST);
        glScissor(x, y, boxW, boxH);
}

static void Point(void)
{
        GLint i;

        glBegin(GL_POINTS);
        for (i = 1; i < 8; i++) {
                GLint j = i * 2;
```

```
                SetColor(i);
                glVertex2i(-j, -j);
                glVertex2i(-j, 0);
                glVertex2i(-j, j);
                glVertex2i(0, j);
                glVertex2i(j, j);
                glVertex2i(j, 0);
                glVertex2i(j, -j);
                glVertex2i(0, -j);
        }
        glEnd();
}

static void Lines(void)
{
        GLint i;

        glPushMatrix();

        glTranslatef(-12, 0, 0);
        for (i = 1; i < 8; i++) {
                SetColor(i);
                glBegin(GL_LINES);
                glVertex2i(-boxW / 4, -boxH / 4);
                glVertex2i(boxW / 4, boxH / 4);
                glEnd();
                glTranslatef(4, 0, 0);
        }

        glPopMatrix();

        (rgb) ? glColor3f(1.0, 1.0, 1.0) : glIndexi(7);
        glBegin(GL_LINES);
        glVertex2i(0, 0);
        glEnd();
}

static void LineStrip(void)
{

        glBegin(GL_LINE_STRIP);
        SetColor(1);
        glVertex2f(PIXEL_CENTER(-boxW / 4), PIXEL_CENTER(-boxH / 4));
        SetColor(2);
        glVertex2f(PIXEL_CENTER(-boxW / 4), PIXEL_CENTER(boxH / 4));
        SetColor(3);
        glVertex2f(PIXEL_CENTER(boxW / 4), PIXEL_CENTER(boxH / 4));
        SetColor(4);
        glVertex2f(PIXEL_CENTER(boxW / 4), PIXEL_CENTER(-boxH / 4));
        glEnd();

        (rgb) ? glColor3f(1.0, 1.0, 1.0) : glIndexi(7);
        glBegin(GL_LINE_STRIP);
        glVertex2i(0, 0);
        glEnd();
}

static void LineLoop(void)
{

        glBegin(GL_LINE_LOOP);
        SetColor(1);
```

```
        glVertex2f(PIXEL_CENTER(-boxW / 4), PIXEL_CENTER(-boxH / 4));
        SetColor(2);
        glVertex2f(PIXEL_CENTER(-boxW / 4), PIXEL_CENTER(boxH / 4));
        SetColor(3);
        glVertex2f(PIXEL_CENTER(boxW / 4), PIXEL_CENTER(boxH / 4));
        SetColor(4);
        glVertex2f(PIXEL_CENTER(boxW / 4), PIXEL_CENTER(-boxH / 4));
        glEnd();

        glEnable(GL_LOGIC_OP);
        glLogicOp(GL_XOR);

        glEnable(GL_BLEND);
        glBlendFunc(GL_ONE, GL_ONE);

        SetColor(5);
        glBegin(GL_LINE_LOOP);
        glVertex2f(PIXEL_CENTER(-boxW / 8), PIXEL_CENTER(-boxH / 8));
        glVertex2f(PIXEL_CENTER(-boxW / 8), PIXEL_CENTER(boxH / 8));
        glEnd();
        glBegin(GL_LINE_LOOP);
        glVertex2f(PIXEL_CENTER(-boxW / 8), PIXEL_CENTER(boxH / 8 + 5));
        glVertex2f(PIXEL_CENTER(boxW / 8), PIXEL_CENTER(boxH / 8 + 5));
        glEnd();
        glDisable(GL_LOGIC_OP);
        glDisable(GL_BLEND);

        SetColor(6);
        glBegin(GL_POINTS);
        glVertex2i(0, 0);
        glEnd();

        (rgb) ? glColor3f(1.0, 1.0, 1.0) : glIndexi(7);
        glBegin(GL_LINE_LOOP);
        glVertex2i(0, 0);
        glEnd();
}

static void Bitmap(void)
{

        glBegin(GL_LINES);
        SetColor(1);
        glVertex2i(-boxW / 2, 0);
        glVertex2i(boxW / 2, 0);
        glVertex2i(0, -boxH / 2);
        glVertex2i(0, boxH / 2);
        SetColor(2);
        glVertex2i(0, -3);
        glVertex2i(0, -3 + OPENGL_HEIGHT);
        SetColor(3);
        glVertex2i(0, -3);
        glVertex2i(OPENGL_WIDTH, -3);
        glEnd();

        SetColor(4);

        glPixelStorei(GL_UNPACK_LSB_FIRST, GL_TRUE);
        glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

        glRasterPos2i(0, 0);
        glBitmap(OPENGL_WIDTH, OPENGL_HEIGHT, 0, 3, 0.0, 0.0, OpenGL_bits);
```

```
}

static void Triangles(void)
{

	glBegin(GL_TRIANGLES);
	SetColor(1);
	glVertex2i(-boxW / 4, -boxH / 4);
	SetColor(2);
	glVertex2i(-boxW / 8, -boxH / 16);
	SetColor(3);
	glVertex2i(boxW / 8, -boxH / 16);

	SetColor(4);
	glVertex2i(-boxW / 4, boxH / 4);
	SetColor(5);
	glVertex2i(-boxW / 8, boxH / 16);
	SetColor(6);
	glVertex2i(boxW / 8, boxH / 16);
	glEnd();

	(rgb) ? glColor3f(1.0, 1.0, 1.0) : glIndexi(7);
	glBegin(GL_TRIANGLES);
	glVertex2i(0, 0);
	glVertex2i(-100, 100);
	glEnd();
}

static void TriangleStrip(void)
{

	glBegin(GL_TRIANGLE_STRIP);
	SetColor(1);
	glVertex2i(-boxW / 4, -boxH / 4);
	SetColor(2);
	glVertex2i(-boxW / 4, boxH / 4);
	SetColor(3);
	glVertex2i(0, -boxH / 4);
	SetColor(4);
	glVertex2i(0, boxH / 4);
	SetColor(5);
	glVertex2i(boxW / 4, -boxH / 4);
	SetColor(6);
	glVertex2i(boxW / 4, boxH / 4);
	glEnd();

	(rgb) ? glColor3f(1.0, 1.0, 1.0) : glIndexi(7);
	glBegin(GL_TRIANGLE_STRIP);
	glVertex2i(0, 0);
	glVertex2i(-100, 100);
	glEnd();
}

static void TriangleFan(void)
{
	GLint vx[8][2];
	GLint x0, y0, x1, y1, x2, y2, x3, y3;
	GLint i;

	y0 = -boxH / 4;
	y1 = y0 + boxH / 2 / 3;
	y2 = y1 + boxH / 2 / 3;
```

```
	y3 = boxH / 4;
	x0 = -boxW / 4;
	x1 = x0 + boxW / 2 / 3;
	x2 = x1 + boxW / 2 / 3;
	x3 = boxW / 4;

	vx[0][0] = x0; vx[0][1] = y1;
	vx[1][0] = x0; vx[1][1] = y2;
	vx[2][0] = x1; vx[2][1] = y3;
	vx[3][0] = x2; vx[3][1] = y3;
	vx[4][0] = x3; vx[4][1] = y2;
	vx[5][0] = x3; vx[5][1] = y1;
	vx[6][0] = x2; vx[6][1] = y0;
	vx[7][0] = x1; vx[7][1] = y0;

	glBegin(GL_TRIANGLE_FAN);
	SetColor(7);
	glVertex2i(0, 0);
	for (i = 0; i < 8; i++) {
		SetColor(7 - i);
		glVertex2iv(vx[i]);
	}
	glEnd();

	(rgb) ? glColor3f(1.0, 1.0, 1.0) : glIndexi(7);
	glBegin(GL_TRIANGLE_FAN);
	glVertex2i(0, 0);
	glVertex2i(-100, 100);
	glEnd();
}

static void Rect(void)
{

	(rgb) ? glColor3f(1.0, 0.0, 1.0) : glIndexi(5);
	glRecti(-boxW / 4, -boxH / 4, boxW / 4, boxH / 4);
}

static void Polygons(void)
{
	GLint vx[8][2];
	GLint x0, y0, x1, y1, x2, y2, x3, y3;
	GLint i;

	y0 = -boxH / 4;
	y1 = y0 + boxH / 2 / 3;
	y2 = y1 + boxH / 2 / 3;
	y3 = boxH / 4;
	x0 = -boxW / 4;
	x1 = x0 + boxW / 2 / 3;
	x2 = x1 + boxW / 2 / 3;
	x3 = boxW / 4;

	vx[0][0] = x0; vx[0][1] = y1;
	vx[1][0] = x0; vx[1][1] = y2;
	vx[2][0] = x1; vx[2][1] = y3;
	vx[3][0] = x2; vx[3][1] = y3;
	vx[4][0] = x3; vx[4][1] = y2;
	vx[5][0] = x3; vx[5][1] = y1;
	vx[6][0] = x2; vx[6][1] = y0;
	vx[7][0] = x1; vx[7][1] = y0;
```

```
	glBegin(GL_POLYGON);
	for (i = 0; i < 8; i++) {
		SetColor(7 - i);
		glVertex2iv(vx[i]);
	}
	glEnd();

	(rgb) ? glColor3f(1.0, 1.0, 1.0) : glIndexi(7);
	glBegin(GL_POLYGON);
	glVertex2i(0, 0);
	glVertex2i(100, 100);
	glEnd();
}

static void Quads(void)
{

	glBegin(GL_QUADS);
	SetColor(1);
	glVertex2i(-boxW / 4, -boxH / 4);
	SetColor(2);
	glVertex2i(-boxW / 8, -boxH / 16);
	SetColor(3);
	glVertex2i(boxW / 8, -boxH / 16);
	SetColor(4);
	glVertex2i(boxW / 4, -boxH / 4);

	SetColor(5);
	glVertex2i(-boxW / 4, boxH / 4);
	SetColor(6);
	glVertex2i(-boxW / 8, boxH / 16);
	SetColor(7);
	glVertex2i(boxW / 8, boxH / 16);
	SetColor(0);
	glVertex2i(boxW / 4, boxH / 4);
	glEnd();

	(rgb) ? glColor3f(1.0, 1.0, 1.0) : glIndexi(7);
	glBegin(GL_QUADS);
	glVertex2i(0, 0);
	glVertex2i(100, 100);
	glVertex2i(-100, 100);
	glEnd();
}

static void QuadStrip(void)
{

	glBegin(GL_QUAD_STRIP);
	SetColor(1);
	glVertex2i(-boxW / 4, -boxH / 4);
	SetColor(2);
	glVertex2i(-boxW / 4, boxH / 4);
	SetColor(3);
	glVertex2i(0, -boxH / 4);
	SetColor(4);
	glVertex2i(0, boxH / 4);
	SetColor(5);
	glVertex2i(boxW / 4, -boxH / 4);
	SetColor(6);
	glVertex2i(boxW / 4, boxH / 4);
	glEnd();
```

```
        (rgb) ? glColor3f(1.0, 1.0, 1.0) : glIndexi(7);
        glBegin(GL_QUAD_STRIP);
        glVertex2i(0, 0);
        glVertex2i(100, 100);
        glVertex2i(-100, 100);
        glEnd();
}

static void Draw(void)
{

        glViewport(0, 0, windW, windH);
        glDisable(GL_SCISSOR_TEST);

        glPushAttrib(GL_COLOR_BUFFER_BIT);

        glColorMask(1, 1, 1, 1);
        glIndexMask(~0);

        glClearColor(0.0, 0.0, 0.0, 0.0);
        glClear(GL_COLOR_BUFFER_BIT);

        glPopAttrib();

        if (mode1) {
                glShadeModel(GL_SMOOTH);
        }
        else {
                glShadeModel(GL_FLAT);
        }

        if (mode2) {
                glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
        }
        else {
                glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
        }

        Viewport(0, 0); Point();
        Viewport(0, 1); Lines();
        Viewport(0, 2); LineStrip();
        Viewport(0, 3); LineLoop();

        Viewport(1, 0); Bitmap();

        Viewport(1, 1); TriangleFan();
        Viewport(1, 2); Triangles();
        Viewport(1, 3); TriangleStrip();

        Viewport(2, 0); Rect();
        Viewport(2, 1); Polygons();
        Viewport(2, 2); Quads();
        Viewport(2, 3); QuadStrip();

        if (doubleBuffer) {
                glutSwapBuffers();
        }
        else {
                glFlush();
        }
}
```

```
static void Args(int argc, char **argv)
{
	GLint i;

	rgb = GL_TRUE;
	doubleBuffer = GL_FALSE;

	for (i = 1; i < argc; i++) {
		if (strcmp(argv[i], "-ci") == 0) {
			rgb = GL_FALSE;
		}
		else if (strcmp(argv[i], "-rgb") == 0) {
			rgb = GL_TRUE;
		}
		else if (strcmp(argv[i], "-sb") == 0) {
			doubleBuffer = GL_FALSE;
		}
		else if (strcmp(argv[i], "-db") == 0) {
			doubleBuffer = GL_TRUE;
		}
	}
}

int main(int argc, char **argv)
{
	GLenum type;

	glutInit(&argc, argv);
	Args(argc, argv);

	type = (rgb) ? GLUT_RGB : GLUT_INDEX;
	type |= (doubleBuffer) ? GLUT_DOUBLE : GLUT_SINGLE;
	glutInitDisplayMode(type);
	glutInitWindowSize(windW, windH);
	glutCreateWindow("Primitive Test");

	Init();

	glutReshapeFunc(Reshape);
	glutKeyboardFunc(Key);
	glutDisplayFunc(Draw);
	glutMainLoop();
}
```

Exerciţiu 5-3.

ModificaŞi programul din laboratorul 3(**Exercitiu propus)** pentru a" adŁuga puŞinŁ culoare" la program.Vom începe cu codul sursŁ din laboratorul 3,unde vom elimina unele dintre comentariile fŁcute.

În primul rând, vom adŁuga **glEnable (GL_COLOR_MATERIAL)** la sfârkitul secvenŞei initRendering.

Programul va suferi un set de moficari ptr. configurarea scenei conform secvenŞelor de mai jos.

```
glColor3f(0.5f, 0.0f, 0.8f);
   glBegin(GL_QUADS);
```

```
//Trapezoid
glVertex3f(-0.7f, -0.5f, 0.0f);
glVertex3f(0.7f, -0.5f, 0.0f);
glVertex3f(0.4f, 0.5f, 0.0f);
glVertex3f(-0.4f, 0.5f, 0.0f);

glEnd();

glPopMatrix();
glPushMatrix();
glTranslatef(1.0f, 1.0f, 0.0f);
glRotatef(_angle, 0.0f, 1.0f, 0.0f);
glScalef(0.7f, 0.7f, 0.7f);

glBegin(GL_TRIANGLES);
glColor3f(0.0f, 0.75f, 0.0f);

//Pentagon
glVertex3f(-0.5f, -0.5f, 0.0f);
glVertex3f(0.5f, -0.5f, 0.0f);
glVertex3f(-0.5f, 0.0f, 0.0f);

glVertex3f(-0.5f, 0.0f, 0.0f);
glVertex3f(0.5f, -0.5f, 0.0f);
glVertex3f(0.5f, 0.0f, 0.0f);

glVertex3f(-0.5f, 0.0f, 0.0f);
glVertex3f(0.5f, 0.0f, 0.0f);
glVertex3f(0.0f, 0.5f, 0.0f);

glEnd();

glPopMatrix();
glPushMatrix();
glTranslatef(-1.0f, 1.0f, 0.0f);
glRotatef(_angle, 1.0f, 2.0f, 3.0f);

glColor3f(0.0f, 0.65f, 0.65f);
glBegin(GL_TRIANGLES);

//Triangle
glVertex3f(0.5f, -0.5f, 0.0f);
glVertex3f(0.0f, 0.5f, 0.0f);
glVertex3f(-0.5f, -0.5f, 0.0f);

glEnd();
```

Dupa ce vom efectua aceste modificŁri, programul nostru va indica urmŁtoarea fereastrŁ.(Figura 4).

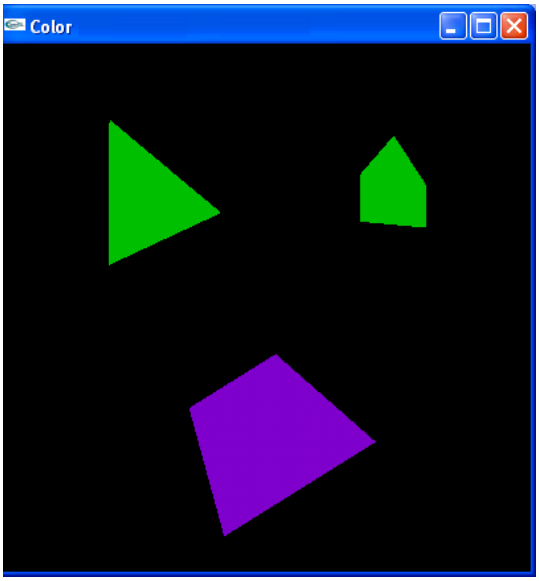


Figura 4

ReŞneŞ cŁ spre deosebire de funcŞile de transformare, putem sŁ introducem **glColor3f** între blocurile **glBegin-glEnd.**

Exerciţiu- 5-4.Amestecarea culorilor

Testul de combinare (blending) este specific modului RGBA de specificare a culorilor scenei de vizualizat. In acest mode de afisare fiecarui fragment i se asociaza o pondere de amestec a colorii sale.

Vom amesteca culorile ki vom face urmŁtoarele modificŁri la codul sursŁ:

```
glBegin(GL_QUADS);

   //Trapezoid
   glColor3f(0.5f, 0.0f, 0.8f);
   glVertex3f(-0.7f, -0.5f, 0.0f);
   glColor3f(0.0f, 0.9f, 0.0f);
   glVertex3f(0.7f, -0.5f, 0.0f);
   glColor3f(1.0f, 0.0f, 0.0f);
   glVertex3f(0.4f, 0.5f, 0.0f);
   glColor3f(0.0f, 0.65f, 0.65f);
   glVertex3f(-0.4f, 0.5f, 0.0f);

   glEnd();

   glPopMatrix();
   glPushMatrix();
   glTranslatef(1.0f, 1.0f, 0.0f);
   glRotatef(_angle, 0.0f, 1.0f, 0.0f);
   glScalef(0.7f, 0.7f, 0.7f);
```

```
glBegin(GL_TRIANGLES);
glColor3f(0.0f, 0.75f, 0.0f);

//Pentagon
glVertex3f(-0.5f, -0.5f, 0.0f);
glVertex3f(0.5f, -0.5f, 0.0f);
glVertex3f(-0.5f, 0.0f, 0.0f);

glVertex3f(-0.5f, 0.0f, 0.0f);
glVertex3f(0.5f, -0.5f, 0.0f);
glVertex3f(0.5f, 0.0f, 0.0f);

glVertex3f(-0.5f, 0.0f, 0.0f);
glVertex3f(0.5f, 0.0f, 0.0f);
glVertex3f(0.0f, 0.5f, 0.0f);

glEnd();

glPopMatrix();
glPushMatrix();
glTranslatef(-1.0f, 1.0f, 0.0f);
glRotatef(_angle, 1.0f, 2.0f, 3.0f);

glBegin(GL_TRIANGLES);

//Triangle
glColor3f(1.0f, 0.7f, 0.0f);
glVertex3f(0.5f, -0.5f, 0.0f);
glColor3f(1.0f, 1.0f, 1.0f);
glVertex3f(0.0f, 0.5f, 0.0f);
glColor3f(0.0f, 0.0f, 1.0f);
glVertex3f(-0.5f, -0.5f, 0.0f);

glEnd();
```

Dupa ce vom efectua aceste modificŁri, programul nostru va indica urmŁtoarea fereastrŁ.(Figura 5).

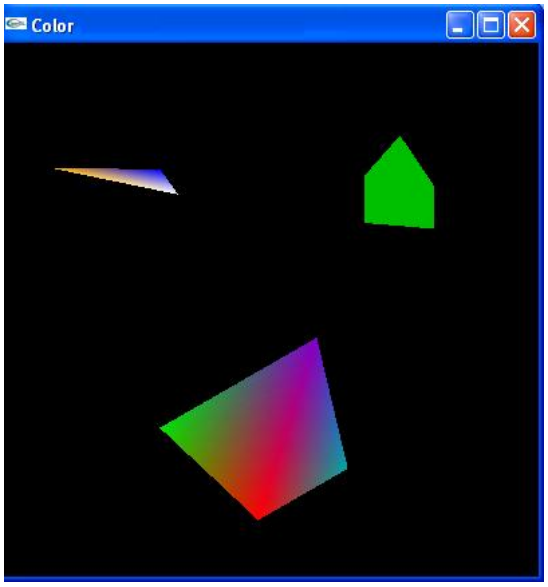


Figura 5

Putem folosi o culoare diferită pentru fiecare nod, si OpenGL le va integra fără probleme în mod automat între culorile de noduri diferite.

Exerciţiu 5-5. Setarea culorii de fundal

Să va schimba culoarea de fundal, de la negru la albastru deschis.

Pentru a face acest lucru, trebuie doar să adăugaţi un apel la **glClearColor(0.7f, 0.9f, 1.0f, 1.0f)** la sfârşitul secventei initRendering..Primii trei parametrii indica culoarea RGB de fundal.Fereastra va fi afişată sub următoarea forma.(Figura 6).

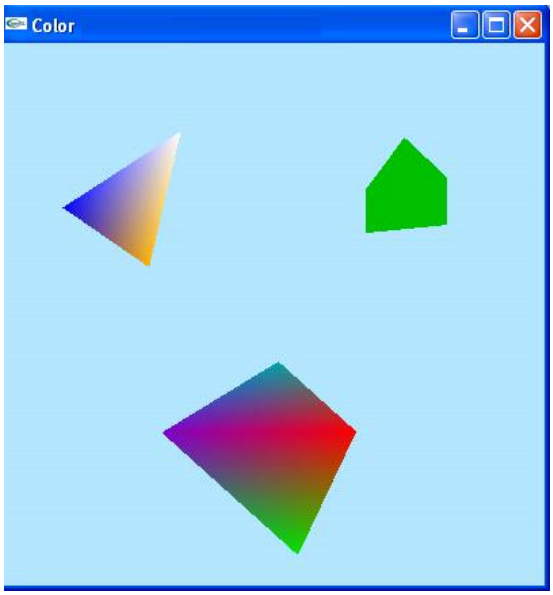


Figura 6

Codul sursă rezultat este prezentat mai jos.

```
#include <iostream>
```

```
#include <stdlib.h>

#ifdef __APPLE__
#include <OpenGL/OpenGL.h>
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

void handleKeypress(unsigned char key, int x, int y) {
	switch (key) {
		case 27:
			exit(0);
	}
}


void initRendering() {
	glEnable(GL_DEPTH_TEST);
	glEnable(GL_COLOR_MATERIAL); //Enable color
	glClearColor(0.7f, 0.9f, 1.0f, 1.0f);
}

void handleResize(int w, int h) {
	glViewport(0, 0, w, h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(45.0, (double)w / (double)h, 1.0, 200.0);
}

float _angle = 30.0f;
float _cameraAngle = 0.0f;

void drawScene() {
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	glRotatef(-_cameraAngle, 0.0f, 1.0f, 0.0f);
	glTranslatef(0.0f, 0.0f, -5.0f);

	glPushMatrix();
	glTranslatef(0.0f, -1.0f, 0.0f);
	glRotatef(_angle, 0.0f, 0.0f, 1.0f);

	glBegin(GL_QUADS);

	//Trapezoid
	glColor3f(0.5f, 0.0f, 0.8f);
	glVertex3f(-0.7f, -0.5f, 0.0f);
```

```
glColor3f(0.0f, 0.9f, 0.0f);
glVertex3f(0.7f, -0.5f, 0.0f);
glColor3f(1.0f, 0.0f, 0.0f);
glVertex3f(0.4f, 0.5f, 0.0f);
glColor3f(0.0f, 0.65f, 0.65f);
glVertex3f(-0.4f, 0.5f, 0.0f);

glEnd();

glPopMatrix();
glPushMatrix();
glTranslatef(1.0f, 1.0f, 0.0f);
glRotatef(_angle, 0.0f, 1.0f, 0.0f);
glScalef(0.7f, 0.7f, 0.7f);

glBegin(GL_TRIANGLES);
glColor3f(0.0f, 0.75f, 0.0f);

//Pentagon
glVertex3f(-0.5f, -0.5f, 0.0f);
glVertex3f(0.5f, -0.5f, 0.0f);
glVertex3f(-0.5f, 0.0f, 0.0f);

glVertex3f(-0.5f, 0.0f, 0.0f);
glVertex3f(0.5f, -0.5f, 0.0f);
glVertex3f(0.5f, 0.0f, 0.0f);

glVertex3f(-0.5f, 0.0f, 0.0f);
glVertex3f(0.5f, 0.0f, 0.0f);
glVertex3f(0.0f, 0.5f, 0.0f);

glEnd();

glPopMatrix();
glPushMatrix();
glTranslatef(-1.0f, 1.0f, 0.0f);
glRotatef(_angle, 1.0f, 2.0f, 3.0f);

glBegin(GL_TRIANGLES);

//Triangle
glColor3f(1.0f, 0.7f, 0.0f);
glVertex3f(0.5f, -0.5f, 0.0f);
glColor3f(1.0f, 1.0f, 1.0f);
glVertex3f(0.0f, 0.5f, 0.0f);
glColor3f(0.0f, 0.0f, 1.0f);
glVertex3f(-0.5f, -0.5f, 0.0f);

glEnd();
```

```
	glPopMatrix();

	glutSwapBuffers();
}

void update(int value) {
	_angle += 2.0f;
	if (_angle > 360) {
		_angle -= 360;
	}

	glutPostRedisplay();
	glutTimerFunc(25, update, 0);
}

int main(int argc, char** argv) {
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
	glutInitWindowSize(400, 400);

	glutCreateWindow("Culoare");
	initRendering();

	glutDisplayFunc(drawScene);
	glutKeyboardFunc(handleKeypress);
	glutReshapeFunc(handleResize);

	glutTimerFunc(25, update, 0);
	glutMainLoop();
	return 0;
}
```