Concepte geometrice în grafica pe calculator

Mihai-Sorin Stupariu

Sem. I, 2016-2017

Cuprins

Capitolul 1

Generalităţi

1.1 Exemple de programe care utilizează OpenGL

1.1.1 Versiune folosind OpenGL "vechi"

```
// Codul sursa este adaptat dupa OpenGLBook.com
#include <windows.h> // biblioteci care urmeaza sa fie incluse
#include <GL/freeglut.h> // nu trebuie uitat freeglut.h
void Initialize(void)
   {
   glClearColor(0.0f, 0.0f, 0.0f, 0.0f); // culoarea de fond a ecranului
   }
void RenderFunction(void)
   {
   glClear(GL_COLOR_BUFFER_BIT);
   glPointSize(20.0);
   glBegin(GL_POINTS);
   // primul varf
   glColor4f( 1.0f, 0.0f, 0.0f, 1.0f);
   glVertex4f(-0.8f, -0.8f, 0.0f, 1.0f);
   // al doilea varf
   glColor4f(0.0f, 1.0f, 0.0f, 1.0f);
   glVertex4f(0.0f, 0.8f, 0.0f, 1.0f);
   // al treilea varf
   glColor4f(0.0f, 0.0f, 1.0f, 1.0f);
   glVertex4f(0.8f, -0.8f, 0.0f, 1.0f);
   glEnd( );
   glFlush ( );
   }
int main(int argc, char* argv[]) {
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
   glutInitWindowPosition (100,100); // pozitia initiala a ferestrei
   glutInitWindowSize(1000,700); //dimensiunile ferestrei
   glutCreateWindow("Primul triunghi - OpenGL <<nou>>"); // titlul ferestrei
   Initialize( );
   glutDisplayFunc(RenderFunction);
   glutMainLoop();
   }
```

Elemente importante:

- Directive preprocesare

- Main
  - Iniţializări GLUT
  - Generare fereastră
  - Apelare procedură iniţializare
  - Apelare procedură desen
  - Apelare MainLoop

1.1.2 Versiune folosind OpenGL "nou"

```
// Codul sursa este adaptat dupa OpenGLBook.com
#include <windows.h> // biblioteci care urmeaza sa fie incluse
#include <stdlib.h> // necesare pentru citirea shader-elor
#include <stdio.h>
#include <GL/glew.h> // glew apare inainte de freeglut
#include <GL/freeglut.h> // nu trebuie uitat freeglut.h
// // // // // // // // // //

GLuint
   VaoId,
   VboId,
   ColorBufferId,
   VertexShaderId,
   FragmentShaderId,
   ProgramId;
// // // // // // // // // //

// Shader-ul de varfuri / Vertex shader (este privit ca un sir de caractere)
const GLchar* VertexShader =
   {
   "#version 400\n"
   "layout(location=0) in vec4 in_Position;\n"
   "layout(location=1) in vec4 in_Color;\n"
   "out vec4 ex_Color;\n"
   "void main(void)\n"
   "\n"\
   " gl_Position = in_Position;\n"
   " ex_Color = in_Color;\n"
   "\n"
   };
// Shader-ul de fragment / Fragment shader (este privit ca un sir de caractere)
const GLchar* FragmentShader =
   {
   "#version 400\n"
   "in vec4 ex_Color;\n"
   "out vec4 out_Color;\n"
   "void main(void)\n"
   "\n"
   " out_Color = ex_Color;\n"
   "\n"
   };
void CreateVBO(void)
   {
// varfurile
   GLfloat Vertices[] = {
   -0.8f, -0.8f, 0.0f, 1.0f,
   0.0f, 0.8f, 0.0f, 1.0f,
```

```
   0.8f, -0.8f, 0.0f, 1.0f };
// culorile, ca atribute ale varfurilor
   GLfloat Colors[] = {
   1.0f, 0.0f, 0.0f, 1.0f,
   0.0f, 1.0f, 0.0f, 1.0f,
   0.0f, 0.0f, 1.0f, 1.0f };
   }
   // se creeaza un buffer nou
   glGenBuffers(1, &VboId);
   // este setat ca buffer curent
   glBindBuffer(GL_ARRAY_BUFFER, VboId);
   // punctele sunt "copiate" in bufferul curent
   glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL_STATIC_DRAW);
   // se creeaza / se leaga un VAO (Vertex Array Object) - util cand se utilizeaza
mai multe VBO
   glGenVertexArrays(1, &VaoId);
   glBindVertexArray(VaoId);
   // se activeaza lucrul cu atribute; atributul 0 = pozitie
   glEnableVertexAttribArray(0);
   //
   glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, 0);
   // un nou buffer, pentru culoare
   glGenBuffers(1, &ColorBufferId);
   glBindBuffer(GL_ARRAY_BUFFER, ColorBufferId);
   glBufferData(GL_ARRAY_BUFFER, sizeof(Colors), Colors, GL_STATIC_DRAW);
   // atributul 1 = culoare
   glEnableVertexAttribArray(1);
   glVertexAttribPointer(1, 4, GL_FLOAT, GL_FALSE, 0, 0);
   }

void DestroyVBO(void)
   {
   glDisableVertexAttribArray(1);
   glDisableVertexAttribArray(0);
   glBindBuffer(GL_ARRAY_BUFFER, 0);
   glDeleteBuffers(1, &ColorBufferId);
   glDeleteBuffers(1, &VboId);
   glBindVertexArray(0);
   glDeleteVertexArrays(1, &VaoId);
   }

void CreateShaders(void)
   {
   VertexShaderId = glCreateShader(GL_VERTEX_SHADER);
   glShaderSource(VertexShaderId, 1, &VertexShader, NULL);
   glCompileShader(VertexShaderId);

   FragmentShaderId = glCreateShader(GL_FRAGMEN_SHADER);
   glShaderSource(FragmentShaderId, 1, &FragmentShader, NULL);
   glCompileShader(FragmentShaderId);

   ProgramId = glCreateProgram();
   glAttachShader(ProgramId, VertexShaderId);
   glAttachShader(ProgramId, FragmentShaderId);
   glLinkProgram(ProgramId);
   glUseProgram(ProgramId);
   }

void DestroyShaders(void)
   {
   glUseProgram(0);
```

```
    glDetachShader(ProgramId, VertexShaderId);
    glDetachShader(ProgramId, FragmentShaderId);
    glDeleteShader(FragmentShaderId);
    glDeleteShader(VertexShaderId);
    glDeleteProgram(ProgramId);
    }

void Initialize(void)
    {
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);// culoarea de fond a ecranului
    }

void RenderFunction(void)
    {
    glClear(GL_COLOR_BUFFER_BIT);
    CreateVBO();
    CreateShaders();
    glPointSize(20.0);
    glDrawArrays(GL_POINTS, 0, 3);
    glFlush ( );
    }

void Cleanup(void)
    {
    DestroyShaders();
    DestroyVBO();
    }

int main(int argc, char* argv[])
    {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowPosition (100,100); // pozitia initiala a ferestrei
    glutInitWindowSize(1000,700); //dimensiunile ferestrei
    glutCreateWindow("Primul triunghi - OpenGL <<nou>>"); // titlul ferestrei
    glewInit(); // nu uitati de initializare glew; trebuie initializat inainte
de a a initializa desenarea
    Initialize( );
    glutDisplayFunc(RenderFunction);
    glutCloseFunc(Cleanup);
    glutMainLoop();
    }
```

Elemente importante

- Directive preprocesare
- Main
  - Iniţializări GLUT
  - Generare fereastră
  - Apelare procedură iniţializare
  - Apelare procedură desen
    - Creare VBO / VAO
    - Creare / Apelare shader-e
  - Ştergere Shader-e, VBO / VAO
  - Apelare MainLoop

1.2 Despre OpenGL

1.2.1 Elemente generale

- OpenGL este o interfaţă de programare (Application Programming Interface API)
- Pentru a funcţiona are nevoie de o serie de biblioteci; versiunea suportată depinde de placa grafică a calculatorului
- OpenGL **nu** este un limbaj de programare (există GLSL)
- Arhitectura de tip *client-server* (CPU-GPU)

1.2.2 "Vechi" *versus* "nou" - scurt istoric

- 1992: versiunea 1.0 a OpenGL (derivat din IRIS GL), lansat de SGI; "open standard"
- 1992: OpenGL Architecture Review Board (elaborarea specificaţiilor)
- 1997: versiunea 1.1
- 2002/2003: versiunile 1.4, 1.5; GLSL (GL Shading Language) apare ca o extensie a funcţionalităţii de bază
- 2004: OpenGL 2.0; GLSL 1.10.59 inclus în "core"
- 2008: OpenGL 3.0: conceptul de "funcţionalităţi depreciate", legate de modul de redare imediat; mecanismul de depreciere a fost implementat / extins în 2009, când au fost lansate versiunile 3.1 şi 3.2
- 2010: OpenGL 3.3 (hardware care suportă Direct3D 10); OpenGL 4.0 (hardware care suportă Direct3D 11); numerotarea versiunilor de GLSL este "sincronizată" (GLSL v. 3.30.6, respectiv v. 4.00.9)
- 2014 OpenGL 4.5 (respectiv GLSL 4.50)

1.2.3 Biblioteci utilizate de OpenGL şi funcţii asociate

- bibliotecă fundamentală (core library): este independentă de platforma pe care se lucrează; funcţiile corespunzătoare au prefixul `gl` (de exemplu: (i) `glClearColor ( ); glFlush ( );` – comune; (ii) `glVertex ( ); glColor ( ); glBegin ( )` – depreciate; (iii) `glGenVertexArrays ( ); glDrawArrays ( )` – OpenGL nou.
- GLU (OpenGL Utility): conţine mai ales proceduri / funcţii legate de proiecţie, precum şi funcţii pentru conice şi cuadrice; funcţiile asociate au prefixul `glu`
- GLUT (OpenGL Utility Toolkit) / freeglut: bibliotecă *dependentă de sistem*, utilizată pentru a realiza fereastra de vizualizare; poate interacţiona cu sisteme de operare bazate pe ferestre de vizualizare; funcţiile specifice au prefixul `glut`

1.3 Resurse

Resurse generale:

- Site-ul oficial OpenGL
- Cărţi: [9], [7], [6]
- Cărţi / tutoriale online: D. Eck, Introduction to Computer Graphics; A. Gerdelan, OpenGL 4 tutorials; http://learnopengl.com/; etc.
- Resurse curs (tutorial de instalare, coduri sursă pentru laborator)

Capitolul 2

Primitive grafice. Atribute ale primitivelor grafice

2.1 Fundamente teoretice

2.1.1 Intersecţii de segmente

În plan (context 2D)

- Segmentele $[AB]$ şi $[CD]$ se intersectează $\Leftrightarrow$ A şi B sunt de o parte şi de alta a dreptei CD **şi** C şi D sunt de o parte şi de alta a dreptei AB.
- Două puncte M şi N sunt de o parte şi de alta a dreptei d de ecuaţie $f(x, y) = \alpha x + \beta y + \gamma = 0 \Leftrightarrow f(M) \cdot f(N) < 0$.

În spaţiu (context 3D)

- **Varianta 1** Reducere la cazul 2D. Se găseşte o izometrie / transformare afină astfel ca figura să fie situată în planul $z = 0$.
- **Varianta 2** Se foloseşte reprezentarea segmentelor cu ajutorul combinaţiilor afine. Segmentele $[AB]$ şi $[CD]$ se intersectează $\Leftrightarrow$

$$\exists s_0, t_0 \in [0, 1] \quad \text{a.î.} \quad (1 - t_0)A + t_0 B = (1 - s_0)C + s_0 D.$$

2.1.2 Poligoane

Reguli referitoare la vârfurile $P_1, P_2, \ldots, P_N$ (diferite două câte două) care determină un poligon, pentru ca poligonul să poată fi desenat:

1. Punctele trebuie să fie coplanare. **De verificat:** condiţia de coplanaritate

$$\text{rang} \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ x_{P_1} & x_{P_2} & x_{P_3} & \ldots & x_{P_N} \\ y_{P_1} & y_{P_2} & y_{P_3} & \ldots & y_{P_N} \\ z_{P_1} & z_{P_2} & z_{P_3} & \ldots & z_{P_N} \end{pmatrix} = 3 \tag{2.1}$$

sau faptul că

$$\dim_{\mathbb{R}} \langle \overrightarrow{P_1P_2}, \overrightarrow{P_1P_3}, \ldots, \overrightarrow{P_1P_N} \rangle = 2. \tag{2.2}$$

O condiţie alternativă este coliniaritatea vectorilor $\overrightarrow{P_1P_2} \times \overrightarrow{P_2P_3}$, $\overrightarrow{P_2P_3} \times \overrightarrow{P_3P_4}$, $\ldots$, $\overrightarrow{P_{N-1}P_N} \times \overrightarrow{P_NP_1}$, $\overrightarrow{P_NP_1} \times \overrightarrow{P_1P_2}$

2. Vârfurile trebuie indicate în ordinea corectă, astfel încât linia pologonală să nu aibă autointersecţii. **De verificat:** intersecţii de segmente (cf. secţiunea 2.1.1).

3. Poligonul trebuie să fie convex. **De verificat:** convexitatea (folosind produse vectoriale).

2.1.3 Orientarea poligoanelor. Faţa/spatele unui poligon convex

Să considerăm un poligon convex $(A_1, A_2, \ldots, A_n)$ (sensul de parcurgere este important!). Alegem trei vârfuri consecutive, de exemplu A_1, A_2, A_3, având coordonatele $A_1 = (x_1, y_1, z_1)$, $A_2 = (x_2, y_2, z_2)$, respectiv $A_3 = (x_3, y_3, z_3)$. Ecuaţia planului determinat de cele trei puncte are forma

$$Ax + By + Cz + D = 0,$$

unde coeficienţii A, B, C şi D sunt daţi de formulele

$$A = \begin{vmatrix} y_1 & z_1 & 1 \\ y_2 & z_2 & 1 \\ y_3 & z_3 & 1 \end{vmatrix}, \qquad B = -\begin{vmatrix} x_1 & z_1 & 1 \\ x_2 & z_2 & 1 \\ x_3 & z_3 & 1 \end{vmatrix} = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix},$$

$$C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}, \qquad D = -\begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix},$$

fiind deduşi din condiţia de coliniaritate

$$\begin{vmatrix} x & y & z & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{vmatrix} = 0.$$

Notăm $\pi(x, y, z) = Ax + By + Cz + D$. Noţiunile de **faţă/spate** a planului (şi, implicit, a poligonului convex fixat) sunt definite astfel:

- $P = (x, y, z)$ se află în faţa planului (poligonului) $\Leftrightarrow \pi(x, y, z) > 0$;
- $P = (x, y, z)$ se află în spatele planului (poligonului) $\Leftrightarrow \pi(x, y, z) < 0$.

Presupunem acum că $D = 0$, deci planul trece prin origine, iar ecuaţia sa este $\pi(x, y, z) = Ax + By + Cz = 0$. Considerând vectorul $N = (A, B, C)$ care direcţionează normala la plan, avem $\pi(A, B, C) > 0$, deci vectorul N indică partea din faţă a poligonului (planului). În general, un vector (x, y, z) este orientat înspre partea din faţă a planului dacă $\pi(x, y, z) > 0$, i.e. $\langle (x, y, z), N, \rangle > 0$, ceea ce înseamnă că proiecţia vectorului (x, y, z) pe N este la fel orientată ca şi N. Prin translaţie, aceste rezultate pot fi extinse pentru un plan arbitrar. Mai mult, presupunând că parcurgem poligonul $(A_1, A_2, \ldots, A_n)$ în sens trigonometric şi că rotim un burghiu drept în sensul indicat de această parcurgere, acesta se va deplasa în sensul indicat de vectorul N, deci înspre faţa poligonului (vezi figura 2.1.3).

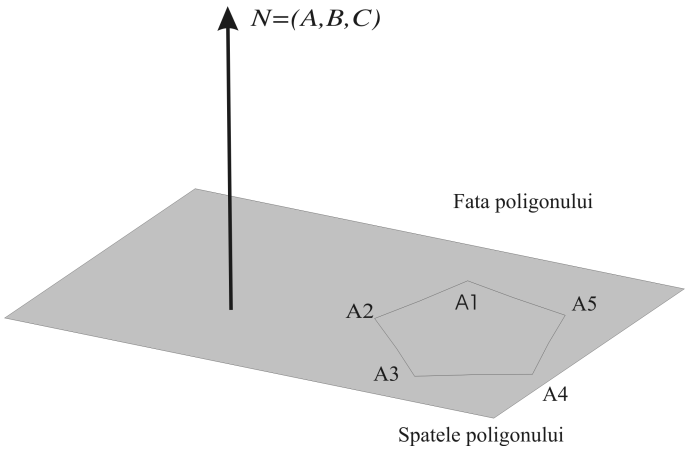

Figura 2.1: Orientarea unui poligon

Exemplul 2.1 Să considerăm poligonul (A_1, A_2, A_3, A_4), unde

$$A_1 = (1, 1, 1),\ A_2 = (-1, 1, 1),\ A_3 = (-1, -1, 1),\ A_4 = (1, -1, 1).$$

În acest caz coeficienţii A, B, C, D sunt $A = 0,\ B = 0,\ C = 4,\ D = -4$, deci vectorul normal este $N = (0, 0, 4)$ (are aceeaşi direcţie şi acelaşi sens cu axa Oz), iar ecuaţia planului suport al acestui poligon este $4z - 4 = 0$. În particular, punctul $(0, 0, 2)$ este situat în faţa poligonului, iar originea $(0, 0, 0)$ în spatele acestuia.

2.2 Algoritmi de rasterizare pentru segmente

Fie $M_0 = (x_0, y_0), M_{End} = (x_{End}, y_{End})$ extremităţile unui segment ce urmează să fie reprezentat în format raster (cu $x_0, y_0, x_{End}, y_{End} \in \mathbb{N}$). Se presupune că dreapta M_0M_{End} are ecuaţia

$$y = mx + n. \tag{2.3}$$

Notaţii şi proprietăţi:

$$\Delta x = x_{End} - x_0, \quad \Delta y = y_{End} - y_0, \quad m = \frac{\Delta y}{\Delta x}.$$

Presupunem $\Delta x > \Delta y > 0$. Numărul de paşi care trebuie realizaţi în lungul axei orizontale Ox este mai mare decât numărul de paşi care trebuie realizaţi în lungul axei verticale Oy.

- **Varianta 1** Implementare directă a ecuaţiei (2.3)
- **Varianta 2** Algoritmul DDA
- **Varianta 2** Algoritmul lui Bresenham

2.3 Exerciţii

Exerciţiul 2.3.1 Fie $A = (3, 1), B = (5, 3), C = (10, 8), D = (1, -1)$. Stabiliţi dacă segmentele $[AB]$ şi $[CD]$ se intersectează şi, în caz afirmativ, determinaţi coordonatele punctului de intersecţie.

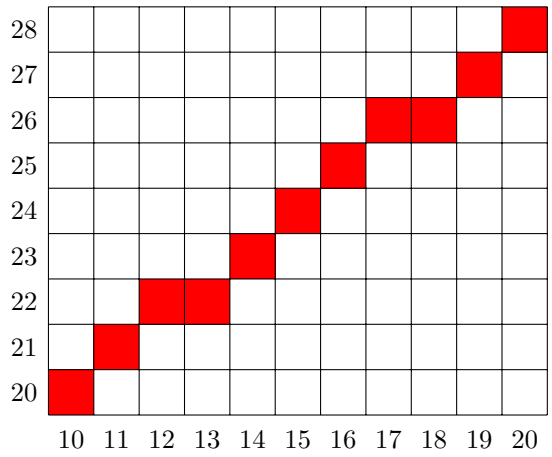

Figura 2.2: **Algoritmul DDA / algoritmul lui Bresenham.**
Pixelii selectaţi pentru a uni punctele $(10, 20)$ şi $(20, 28)$ sunt coloraţi cu roşu.

Exerciţiul 2.3.2 Fie $A = (1, 2, 3), B = (3, 2, 5), C = (8, 6, 2), D = (-1, 0, 3)$, Stabiliţi dacă segmentele $[AB]$ şi $[CD]$ se intersectează şi, în caz afirmativ, determinaţi coordonatele punctului de intersecţie.

Exerciţiul 2.3.3
a) Verificaţi echivalenţa dintre condiţiile (2.1) şi (2.2).
b) Care este interpretarea geometrică dacă egalitatea din condiţia (2.1) devine inegalitate?

Exerciţiul 2.3.4 Calculaţi produsul vectorial $v \times w$, pentru $v = (1, 2, 0)$ şi $w = (-3, 1, 0)$. Aceeaşi cerinţă pentru $v = (2, 4, 1)$ şi $w = (-1, -1, 1)$.

Exerciţiul 2.3.5 Stabiliţi natura virajelor din poligonul $P_1P_2P_3P_4P_5P_6$, pentru $P_1 = (1, 0)$, $P_2 = (4, 0)$, $P_3 = (7, 2)$, $P_4 = (5, 4)$, $P_5 = (8, 6)$, $P_6 = (0, 7)$.

Exerciţiul 2.3.6 Considerăm punctele $P_1 = (2, 3, 5), P_2 = (3, 4, 6), P_3 = (0, 3, 4)$, $P_4 = (3, 2, 5)$. Stabiliţi care este poziţia lui P_4 faţă de $\Delta P_1P_2P_3$.

Exerciţiul 2.3.7 Fie P_1, P_2, P_3 trei puncte necoliniare din $\mathbb{R}^3$. Considerăm ecuaţia planului $Ax+By+Cz+D = 0$ obţinută prin dezvoltarea determinantului din ecuaţia (2.1). Verificaţi că $\overrightarrow{P_1P_2} \times \overrightarrow{P_2P_3} = (A, B, C)$.

Exerciţiul 2.3.8 Fie punctele $A = (2, 3, 0), B = (5, 4, 0), C = (0, 8, 0)$. Stabiliţi ordinea în care trebuie indicate punctele astfel ca punctul $(-1, 2, 6)$ să fie în faţa planului triunghiului determinat de ele.

Exerciţiul 2.3.9 Considerăm punctele $A = (-1, 0, 1)$, $B = (1, 0, 1)$, $C = (1, 0, -1)$, $D = (-1, -0, -1)$. În ce ordine trebuie indicate punctele pentru ca faţa poligonului să fie orientată spre prelungirea axei Oy?

Exerciţiul 2.3.10 La reprezentarea unui segment folosind algoritmul DDA sunt selectaţi 11 pixeli (inclusiv cei corespunzând extremităţilor segmentului). Calculaţi panta dreptei suport a segmentului, ştiind că dintre pixelii selectaţi doar doi au aceeaşi ordonată. Daţi exemplu de puncte M_0 şi M_{End} care să verifice aceste condiţii şi scrieţi explicit algoritmul DDA pentru aceste puncte.

Exerciţiul 2.3.11 La reprezentarea unui segment folosind algoritmul Bresenham s-a obţinut parametrul de decizie iniţial $p_0 = 2$. Ştiind că panta dreptei suport a segmentului este 0.6, calculaţi valoarea lui p_1. Daţi exemplu de puncte M_0 şi M_{End} care să verifice aceste condiţii şi scrieţi explicit algoritmul lui Bresenham pentru aceste puncte (indicaţi şi valorile parametrului de decizie).

Exerciţiul 2.3.12

a) Cum se procedează în algoritmul DDA / algoritmul lui Bresenham dacă se doreşte parcurgerea segmentului de la dreapta la stânga?

b) Scrieţi algoritmul DDA / algoritmul lui Bresenham pentru cazul $0 < \Delta x < \Delta y$. Ce alte situaţii mai pot să apară?

Exerciţiul 2.3.13 Explicaţi cum poate fi adaptată metoda din algoritmul lui Bresenham pentru a reprezenta alte curbe (cerc, elipsă, etc.).

Exerciţiul 2.3.14 Ilustraţi diferenţa de robusteţe şi de eficienţă pentru cele trei variante de algoritmi din seţiunea 2.2.

Exerciţiul 2.3.15 Care este secvenţa de cod sursă care generează culoarea galben pentru o primitivă grafică?

Exerciţiul 2.3.16 Câte triunghiuri vor fi desenate pentru `GL_TRIANGLES` dacă sunt indicate 40 de vârfuri distincte?

Capitolul 3

Transformări

OpenGL utilizează (implicit sau explicit) 4 tipuri de transformări:

- **de modelare**, aplicate obiectelor care urmează să fie reprezentate; acest tip de transformări este discutat în detaliu în paragrafele din acest capitol;
- **de vizualizare**, care au rolul de a modifica poziţia observatorului faţă de cea implicită;
- **de proiecţie**, care precizează tipul proiecţiei utilizate, precum şi ceea ce urmează a fi decupat;
- **de viewport**, care indică date despre fereastra de vizualizare (vizor).

Pentru a realiza o uniformizare a tuturor acestor tipuri şi pentru a putea compune diferitele transformări care pot sa apară în realizarea unei scene, o transformare arbitrară este reprezentată în memoria internă a bibliotecilor OpenGL printr-o matrice 4×4. Fundamentele matematice ale acestei identificări sunt explicate în secţiunea 3.2.

3.1 Coordonate omogene

Scopul acestui paragraf este de a explica, pe scurt, construcţia spaţiului proiectiv real şi de a explica modul în care sunt introduse coordonatele omogene. Pentru mai multe detalii referitoare la acest subiect (şi la geometria proiectivă, în general) pot fi consultate lucrările [10], [12] sau [2]. Utilitatea coodonatelor omogene va deveni clară în secţiunea 3.2.

Pe mulţimea $\mathbb{R}^4 \setminus \{0\}$ introducem relaţia de echivalenţă $\sim$ dată prin $X \sim Y$ dacă şi numai dacă există $\lambda \neq 0$ astfel ca $Y = \lambda X$. Clasa de echivalenţă a unui element $X = (X_1, X_2, X_3, X_0)$ va fi notată cu $[X_1, X_2, X_3, X_0]$. Astfel, de exemplu, avem $[1, 2, 3, -5] = [2, 4, 6, -10]$. Mulţimea

$$\mathbb{P}^3\mathbb{R} = \{[X_1, X_2, X_3, X_0] | (X_1, X_2, X_3, X_0) \in \mathbb{R}^4 \setminus \{0\}$$

a tuturor claselor de echivalenţă este numită **spaţiu proiectiv real**. Pentru un element $P = [X_1, X_2, X_3, X_0] \in \mathbb{P}^3\mathbb{R}$, sunt definite şi unice până la înmulţirea cu un scalar nenul **coordonatele omogene** ale lui P, şi anume (X_1, X_2, X_3, X_0).

Există o aplicaţie de incluziune naturală $\iota : \mathbb{R}^3 \hookrightarrow \mathbb{P}^3\mathbb{R}$, dată prin formula $\iota(x_1, x_2, x_3) = [x_1, x_2, x_3, 1]$.

Pe de altă parte, fie $[\delta]$ clasa de echivalenţă a unei drepte δ modulo relaţia de paralelism (intuitiv, o astfel de clasă de echivalenţă poate fi privită ca punct de

la infinit al unei drepte, în sensul că două drepte paralele au în comun un punct la infinit, iar prin acest punct trece orice dreaptă paralelă cu ele). Fie (v_1, v_2, v_3) un vector director al lui δ (atunci orice vector de forma $(\lambda v_1, \lambda v_2, \lambda v_3)$, cu $\lambda \neq 0$ este, la rândul său, un vector director al lui δ). Îi putem asocia lui $[\delta]$ elementul $[v_1, v_2, v_3, 0]$ din spaţiul $\mathbb{P}^3\mathbb{R}$.

În concluzie, spaţiul proiectiv real $\mathbb{P}^3\mathbb{R}$ conţine două tipuri de elemente:

- **"puncte reale":**, adică elemente de forma $P = [X_1, X_2, X_3, X_0]$ cu $X_0 \neq 0$; P îcorespunde punctului $\left(\frac{X_1}{X_0}, \frac{X_2}{X_0}, \frac{X_3}{X_0}\right)$ din spaţiul geometric $\mathbb{R}^3$,
- **"puncte de la infinit":**, adică elemente de forma $Q = [X_1, X_2, X_3, 0]$. Mulţimea punctelor de la infinit formează un plan, numit **planul de la infinit**, având ecuaţia $X_0 = 0$.

Oricărui loc geometric L din $\mathbb{R}^3$ i se poate asocia un loc geometric $\bar{L}$ din $\mathbb{P}^3\mathbb{R}$, prin "completarea" cu puncte de la infinit. Mai mult, dacă L este descris prin ecuaţii implicite, i se poate asocia un loc geometric $\bar{L}$ din $\mathbb{P}^3\mathbb{R}$, obţinut prin **omogenizarea** ecuaţiilor lui L. Astfel, dacă L este dat prin ecuaţia

$$a_1x_1 + a_2x_2 + a_3x_3 + a_0 = 0,$$

omogenizarea ecuaţiei lui L se obţine astfel: se notează $x_i = \frac{X_i}{X_0}$ $(i = 1, 2, 3)$ şi se efectuează înlocuirile în ecuaţia lui L, obţinând, după eliminarea numitorului, ecuaţia omogenă a lui $\bar{L}$

$$a_1X_1 + a_2X_2 + a_3X_3 + a_0X_0 = 0.$$

Este de menţionat faptul că, în general, o ecuaţie omogenă de forma

$$a_1X_1 + a_2X_2 + a_3X_3 + a_0X_0 = 0 \qquad (\text{cu } \operatorname{rang}(a_1, a_2, a_3, a_0) = 1)$$

reprezintă **un plan** din spaţiul proiectiv real $\mathbb{P}^3\mathbb{R}$, iar **o dreaptă** este dată printr-un sistem de forma

$$\begin{cases} a_1X_1 + a_2X_2 + a_3X_3 + a_0X_0 = 0 \\ b_1X_1 + b_2X_2 + b_3X_3 + b_0X_0 = 0 \end{cases} \qquad \left(\text{cu } \operatorname{rang} \begin{pmatrix} a_1 & a_2 & a_3 & a_0 \\ b_1 & b_2 & b_3 & b_0 \end{pmatrix} = 2\right).$$

Exemplul 3.1 Să considerăm dreapta d din $\mathbb{R}^3$ având ecuaţiile implicite

$$\begin{cases} x_1 - x_2 - x_3 - 4 = 0 \\ 3x_1 - x_2 + x_3 + 2 = 0. \end{cases}$$

Trecând la ecuaţii parametrice, se deduce că un vector director al dreptei este $(1, 2, -1)$. Prin omogenizare, se obţine dreapta $\bar{d}$ din $\mathbb{P}^3\mathbb{R}$ având ecuaţiile

$$\begin{cases} X_1 - X_2 - X_3 - 4X_0 = 0 \\ 3X_1 - X_2 + X_3 + 2X_0 = 0. \end{cases}$$

Este de remarcat faptul că intersecţia dintre $\bar{d}$ şi dreapta de la infinit este dată de soluţiile sistemului

$$\begin{cases} X_1 - X_2 - X_3 - 4X_0 = 0 \\ 3X_1 - X_2 + X_3 + 2X_0 = 0 \\ X_0 = 0, \end{cases}$$

adică punctul de la infinit al dreptei d, şi anume $[1, 2, -1, 0]$.

3.2 Reprezentarea matriceală a transformărilor

O translaţie $\mathbf{T} : \mathbb{R}^3 \to \mathbb{R}^3$ de vector $\mathbf{t} = (t_1, t_2, t_3)$ poate fi reprezentată matriceal sub forma

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \mapsto \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix}.$$

Pe de altă parte, rotaţia de axă Ox_3 şi unghi θ este dată de

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \mapsto \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix},$$

iar scalarea de factori s_1, s_2, s_3 (de-a lungul celor trei axe) se reprezintă ca

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \mapsto \begin{pmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

Se observă o diferenţă calitativă evidentă între translaţie, pe de o parte, şi rotaţie şi scalare, pe de altă parte. O notaţie uniformă, care înglobează toate cele trei tipuri de transformări poate fi obţinută utilizând coordonate omogene Mai precis, fie $f : \mathbb{R}^3 \to \mathbb{R}^3$ o aplicaţie afină arbitrară a lui $\mathbb{R}^3$, dată prin

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \mapsto \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}. \tag{3.1}$$

Relaţia (3.1) poate fi rescrisă, introducând a patra coordonată şi ţinând cont de forma aplicaţiei ι, sub forma

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix}.$$

Trcând la coordonate omogene (X_1, X_2, X_3, X_0) se obţine transformarea

$$\begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_0 \end{pmatrix} \mapsto \begin{pmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_0 \end{pmatrix}.$$

Acestei transformări afine îi corespunde matricea 4×4

$$\mathcal{M}_f = \begin{pmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

În general, orice matrice inversabilă $A = (a_{i,j})_{i,j=1,2,3,0}$ cu 4 linii şi 4 coloane induce o transformare geometrică $\mathbf{T}_A : \mathbb{P}^3\mathbb{R} \to \mathbb{P}^3\mathbb{R}$, dată de formula

$$\begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_0 \end{pmatrix} \mapsto A \cdot \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_0 \end{pmatrix}.$$

În cazul în care elementele ultimei linii a matricei A sunt de forma $a_{01} = 0$, $a_{02} = 0$, $a_{03} = 0$, $a_{00} = \alpha$ ($\alpha \neq 0$), această transformare duce punctele reale în puncte reale şi punctele de la infinit în puncte de la infinit. În general, însă, transformările induse de matrice 4×4 pot transforma unele puncte reale în puncte imaginare şi reciproc. Geometric, este vorba de matrice induse de transformări perspective ale lui $\mathbb{R}^3$, care transformă (unele) drepte paralele în drepte concurente şi invers.

Exemplul 3.2 Să considerăm matricea

$$A = \begin{pmatrix} 1 & 2 & 0 & 1 \\ 0 & 1 & 3 & -1 \\ 1 & 0 & 0 & -1 \\ 1 & 1 & -1 & 2 \end{pmatrix}.$$

Atunci punctul real $P = [1, 1, 1, 1]$ este transformat în punctul real $[4, 3, 0, 3] = [\frac{4}{3}, 1, 0, 1]$. În schimb, imaginea punctului real $Q = [1, 3, 6, 1]$ prin $\mathbf{T}_A$ este punctul de la infinit $[8, 20, 0, 0]$ iar punctul de la infinit $R = [1, -1, 2, 0]$ are ca imagine punctul real $[-1, 5, 1, -2] = [\frac{1}{2}, -\frac{5}{2}, -\frac{1}{2}, 1]$.

3.3 Transformări de modelare standard în OpenGL. Utilizarea cuaternionilor

3.4 Exerciţii

Exerciţiul 3.4.1 Determinaţi λ astfel ca în planul proiectiv $\mathbb{P}^2\mathbb{R}$ să aibă loc egalitatea $[1, 2, 3] = [2, 4, \lambda^2 - \lambda]$.

Exerciţiul 3.4.2 Determinaţi punctul de la infinit al dreaptei d din $\mathbb{R}^3$ având ecuaţiile implicite

$$\begin{cases} 2x_1 + x_2 + 2x_3 - 6 = 0 \\ -x_1 + x_2 - 3x_3 + 2 = 0. \end{cases}$$

Exerciţiul 3.4.3 Pentru transformarea $\mathbf{T}_A$ de la exerciţiul 3.2 daţi şi alte exemple de puncte reale care au ca imagine puncte de la infinit şi reciproc. Determinaţi mulţimea tuturor punctelor de la infinit care au ca imagine puncte de la infinit şi stabiliţi dimensiunea acestui loc geometric.

Exerciţiul 3.4.4 Determinaţi matricea compunerii dintre translaţia de vector $\mathbf{t} = (2, 3, 1)$ şi scalarea de factori $(-1, 3, 5)$. Care este rezultatul dacă cele două transformări sunt aplicate în ordine inversă?

Exerciţiul 3.4.5 Determinaţi produsul cuaternionilor $q_1 = 1 - 2i - 2j - 4k$, $q_2 = 2 + i + 3j - k$.

Capitolul 4

Reprezentarea scenelor 3D

4.1 Coordonate de vizualizare

Precizarea faptului că se doreşte realizarea unei scene 3D se face indicând modul de lucru matriceal, precum şi precizând informaţii legate de poziţia observatorului şi de direcţia de observare.

Coordonatele observatorului în raport cu reperul de modelare sunt date de (x_0, y_0, z_0), iar **coordonatele punctului de referinţă** P_{ref} (punctul spre care priveşte observatorul) sunt $(x_{\text{ref}}, y_{\text{ref}}, z_{\text{ref}})$. Vectorul $V = (V_x, V_y, V_z)$ induce **verticala din planul de vizualizare**, i.e. planul care trece prin punctul P_0 şi este perpendicular pe vectorul $N = \overrightarrow{P_{\text{ref}}P_0} = P_0 - P_{\text{ref}}$ (se presupune că V şi N nu sunt coliniari, altminteri obţinem erori de reprezentare). Valorile implicite (utilizate inclusiv în cazul în care reprezentăm scene 2D, fără a schimba reperul de modelare) sunt $P_0 = (0, 0, 0)$, $P_{\text{ref}} = (0, 0, -1)$, respectiv $V = (0, 1, 0)$.

Indicarea punctelor P_0, P_{ref} şi a vectorului V generează un nou sistem de coordonate, numite **coordonate de vizualizare**. Originea acestui sistem este punctul P_0, iar axele sistemului sunt date de reperul ortonormat (u, v, n), obţinut după cum urmează:

$$n = \frac{N}{\|N\|}, \qquad u = \frac{V \times n}{\|V\|}, \qquad v = n \times u.$$

Vectorul v este coliniar cu proiecţia $V - \frac{\langle V,N\rangle \cdot N}{\|N\|}$ vectorului V pe planul de vizualizare, iar în cazul în care V este conţinut în acest plan (i.e. V este perpendicular pe N) are loc relaţia $v = \frac{V}{\|V\|}$. Trecerea de la coordonate de modelare la coordonate de vizualizare se face în două etape:

1. Se translatează originea reperului de modelare în originea reperului de vizualizare, iar matricea acestei transformări este

$$T = \begin{pmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

2. Se efectuează o rotaţie, astfel ca sistemul (u, v, n) să fie transformat în sistemul canonic (e_1, e_2, e_3) care direcţionează axele reperului de modelare. Matricea 3×3 care transformă sistemul (e_1, e_2, e_3) în (u, v, n) este matricea ortogonală

$$A = \begin{pmatrix} u_x & v_x & n_x \\ u_y & v_y & n_y \\ u_z & v_z & n_z \end{pmatrix},$$

deci trecerea de la (u, v, n) la (e_1, e_2, e_3) este realizată de $A^{-1} = A^t$. Matricea 4×4 corespunzătoare acestei schimbări de reper este

$$R = \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

În concluzie, trecerea de la reperul de modelare la reperul de vizualizare este dată de matricea

$$\mathcal{M}_{\text{coord.mod,coord.viz}} = R \cdot T = \begin{pmatrix} u_x & u_y & u_z & -\langle u, P_0 \rangle \\ v_x & v_y & v_z & -\langle v, P_0 \rangle \\ n_x & n_y & n_z & -\langle n, P_0 \rangle \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(prin abuz de notaţie am identificat punctul P_0 cu vectorul P_0 din $\mathbf{R}^3$).

4.2 Transformări de proiecţie

4.2.1 Proiecţii ortogonale

Utilizarea unei proiecţii ortogonale se face precizând modul matriceal de lucru (proiecţie), precum şi indicând paralelipipedul dreptunghic care urmează să fie decupat.

Coordonatele în raport cu care se lucrează sunt cele de vizualizare, deci originea este situată în punctul P_0, iar axele sistemului sunt date de vectorii u, v, n. Paralelipipedul care urmează să fie decupat este delimitat de planele $x = xw_{\min}, x = xw_{\max}$; $y = yw_{\min}, y = yw_{\max}$, respectiv $z = z_{\text{near}}$, unde $z_{\text{near}} = -d_{\text{near}}$, $z = z_{\text{far}}$, unde $z_{\text{far}} = -d_{\text{far}}$. Valorile implicite sunt $-1.0, 1.0, -1.0, 1.0, -1.0, 1.0$ (valori normalizate), iar pentru valori arbitrare se efectuează normalizarea (aducerea parametrilor indicaţi la valorile implicite), care este o scalare, iar matricea 4×4 asociată este

$$\mathcal{M}_{\text{orto,norm}} = \begin{pmatrix} \frac{2}{xw_{\max} - xw_{\min}} & 0 & 0 & -\frac{xw_{\max} + xw_{\min}}{xw_{\max} - xw_{\min}} \\ 0 & \frac{2}{yw_{\max} - yw_{\min}} & 0 & -\frac{yw_{\max} + yw_{\min}}{yw_{\max} - yw_{\min}} \\ 0 & 0 & -\frac{2}{z_{\text{near}} - z_{\text{far}}} & \frac{z_{\text{near}} + z_{\text{far}}}{z_{\text{near}} - z_{\text{far}}} \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.1}$$

4.2.2 Proiecţii paralele oblice

4.2.3 Proiecţii perspective

Bibliotecile OpenGL conţin două funcţii care permit realizarea unei proiecţii centrale (perspectivă) pentru o scenă dată. Prima dintre ele are un caracter general, indicând un trunchi de piramidă care urmează să fie decupat.

Cea de-a doua funcţie, conţinută în biblioteca GLU, permite realizarea unei proiecţii perspective simetrice.

4.3 Exerciţii

Exerciţiul 4.3.1 Se presupune că s-a apelat funcţia `gluLookAt (1,1,1,2,1,1,0,1,0)`. Calculaţi versorul care direcţionează versorul din planul de vizualizare.

Exerciţiul 4.3.2 Care sunt valorile implicite pentru verticala din planul de vizualizare, indicată în funcţia `gluLookAt`?

Exerciţiul 4.3.3 Se apelează funcţia `glOrtho (10, 30, 20, 40, 5, 10)`. Calculaţi suma elementelor de pe diagonala principală a matricei.

Exerciţiul 4.3.4 De ce în cazul funcţiei `glFrustum` sunt necesari şase parametri, iar în cazul funcţiei `gluPerspective` doar patru?.

Capitolul 5

Iluminarea scenelor

5.1 Modele de iluminare

Un model de iluminare este definit de patru componente:

- intensitatea luminii ambientale globale;
- poziţia punctului de vizualizare faţă de scenă;
- diferenţierea feţelor obiectelor;
- modul în care este calculată culoarea specular̆ă (separat de componenta ambientală şi cea difuză şi după texturare).

$$\begin{aligned}\text{vertex color} = \\ &\text{emission}_{\text{material}} + \\ &\text{ambient}_{\text{light model}} * \text{ambient}_{\text{material}} + \\ &\sum_{i=0}^{N-1} \text{attenuation factor}_i \cdot \text{spotlight effect}_i \cdot \\ &(\text{ambient term} + \text{diffuse term} + \text{specular term})_i,\end{aligned} \tag{5.1}$$

unde N este numărul surselor de lumină.

Pentru o sursă (punctuală) fixată factorul de atenuare (attenuation factor) se calculează cu formula

$$\text{attenuation factor} = \frac{1}{a_0 + a_1 d + a_2 d^2},$$

unde d este distanţa de la sursa de lumină la vârful considerat.

Efectul de tip spot este cuantificat de factorul

$$\text{spotlight effect} = \begin{cases} 1, & \text{dacă } \theta_l = 180^0 \\ 0, & \text{dacă } \mathbf{v}_{\text{obj}} \cdot \mathbf{v}_{\text{light}} < \cos\theta_l, \\ (\mathbf{v}_{\text{obj}} \cdot \mathbf{v}_{\text{light}})^{a_l}, & \text{în celelalte cazuri.} \end{cases}$$

Cu $\mathbf{v}_{\text{obj}}$ este notat vectorul unitar orientat de la sursa de lumină la obiectul iluminat, iar cu $\mathbf{v}_{\text{light}}$ este notat versorul direcţiei spotului de lumină `GL_SPOT_DIRECTION`.

Termenul ambiental corespunzător unei surse de lumină este

$$\text{ambient term} = \text{ambient}_{\text{light}} * \text{ambient}_{\text{material}}.$$

Reflexia difuză pentru o sursă de lumină este descrisă de factorul

$$\text{diffuse term} = \begin{cases} (\mathbf{L}\cdot\mathbf{n})\cdot \text{diffuse}_{\text{light}} * \text{diffuse}_{\text{material}}, & \text{dacă } \mathbf{L}\cdot\mathbf{n} > 0 \\ 0, & \text{dacă } \mathbf{L}\cdot\mathbf{n} \leq 0, \end{cases}$$

unde $\mathbf{L}$ este vectorul unitar orientat de la vârf la sursa de lumină (în cazul surselor direcţionale este opusul direcţiei acesteia, normat), iar $\mathbf{n}$ este normala la suprafaţă în vârful considerat. Reflexia speculară este dată de

$$\text{specular term} = \begin{cases} (\mathbf{H}\cdot\mathbf{n})^{\text{shininess}}\cdot \text{specular}_{\text{light}} * \text{specular}_{\text{material}}, & \text{dacă } \mathbf{L}\cdot\mathbf{n} > 0 \\ 0, & \text{dacă } \mathbf{L}\cdot\mathbf{n} \leq 0, \end{cases}$$

unde $\mathbf{H} = \dfrac{\mathbf{L}+\mathbf{V}}{\|\mathbf{L}+\mathbf{V}\|}$, iar $\mathbf{V}$ este versorul determinat de vârful considerat şi poziţia observatorului.

5.2 Exerciţii

Exerciţiul 5.2.1 Care este vectorul de poziţie implicit pentru o sursă de lumină?

Exerciţiul 5.2.2 Care este combinaţia RGB corespunzătoare unui vârf cu proprietăţile de material `GL_EMISSION` $(0.3, 0.4, 0.1)$, `GL_AMBIENT` $(1.0, 0.4, 0.5)$, dacă `GL_LIGHT_MODEL_AMBIENT` este $(0.4, 0.5, 0.2)$ şi nu este activată nicio sursă de lumină?

Exerciţiul 5.2.3 Determinaţi valoarea termenului difuz (*diffuse term*) pentru un vârf de coordonate $(2, 3, 4)$ cu proprietăţile de material `GL_DIFFUSE`=(0.3, 0.4, 0.2) ştiind că normala la suprafaţă în vârful respectiv este $(0, 0, 1)$ şi sursa de lumină este situată în punctul $(3, 4, 3)$?

Exerciţiul 5.2.4 Care este factorul de atenuare implicit pentru o sursă aflată la distanţa 5 de un vârf?

Capitolul 6

Efecte vizuale

6.1 Transparenţă; amestecare; factor α

Factorul destinaţie (fragmentul deja procesat) şi factorul sursă (obiectul care urmează să fie procesat şi înregistrat) sunt "amestecate" utilizând o funcţie $\varphi(D, F_d, S, F_s)$. Combinarea se realizează după formula

$$\varphi(D, F_d, S, F_s) = F_d * D + F_s * S, \tag{6.1}$$

urmată de 'clamp'.

Constanta simbolică	**Factor RGB**	**Factor A**
GL_ZERO	$(0, 0, 0)$	0
GL_ONE	$(1, 1, 1)$	1
GL_SRC_ALPHA	(A_s, A_s, A_s)	A_s
GL_ONE_MINUS_SRC_ALPHA	$(1, 1, 1) - (A_s, A_s, A_s)$	$1 - A_s$
GL_DST_ALPHA	(A_d, A_d, A_d)	A_d
GL_ONE_MINUS_DST_ALPHA	$(1, 1, 1) - (A_d, A_d, A_d)$	$1 - A_d$
GL_SRC_COLOR	(R_s, G_s, B_s)	A_s
GL_ONE_MINUS_SRC_COLOR	$(1, 1, 1) - (R_s, G_s, B_s)$	$1 - A_s$
GL_DST_COLOR	(R_d, G_d, B_d)	A_d
GL_ONE_MINUS_DST_COLOR	$(1, 1, 1) - (R_d, G_d, B_d)$	$1 - A_d$
GL_CONSTANT_COLOR	(R_c, G_c, B_c)	A_c
GL_ONE_MINUS_CONSTANT_COLOR	$(1, 1, 1) - (R_c, G_c, B_c)$	$1 - A_c$
GL_CONSTANT_ALPHA	(A_c, A_c, A_c)	A_c
GL_ONE_MINUS_CONSTANT_ALPHA	$(1, 1, 1) - (A_c, A_c, A_c)$	$1 - A_c$
GL_SRC_ALPHA_SATURATE	$(f, f, f);\ f = \min(A_s, 1 - A_d)$	1

Mod de amestecare	**Expresie matematică (φ)**
GL_FUNC_ADD	$F_s \cdot S + F_d \cdot D$
GL_FUNC_SUBTRACT	$F_s \cdot S - F_d \cdot D$
GL_FUNC_REVERSE_SUBTRACT	$F_d \cdot D - F_s \cdot S$
GL_MIN	$\min(F_s \cdot S, F_d \cdot D)$
GL_MAX	$\max(F_s \cdot S, F_d \cdot D)$
GL_LOGIC_OP	F_s *op* F_d

6.2 Teste de adâncime

O altă clasă de funcţii standard în OpenGL sunt cele care permit efectuarea unor teste de adâncime pentru selectarea suprafeţelor vizibile. Pentru a putea utiliza aceste facilităţi, trebuie în primul rând, modificată funcţia GLUT de iniţializare `glutInitDisplayMode`, adăugându-i argumentul `GLUT_DEPTH`, care specifică utilizarea buffer-ului de adâncime (depth buffer). De exemplu, dacă utilizăm un singur buffer de culoare şi dorim să activăm buffer-ul de adâncime, în programul principal va figura comanda:

```
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
```

Valorile pentru buffer-ul de adâncime pot fi iniţializate cu funcţia

```
glClear (GL_DEPTH_BUFFER_BIT);
```

Alternativ, acestea pot fi iniţializate simultan cu cele ale buffer-ului de culoare:

```
glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Activarea propriu-zisă a rutinelor de detectare a vizibilităţii se face cu funcţia

```
glEnable (GL_DEPTH_TEST);
```

iar dezactivarea lor se face, în mod corespunzător, cu funcţia

```
glDisable (GL_DEPTH_TEST);
```

În OpenGL, valorile pentru care se efectuează testul de adâncime sunt normalizate, fiind situate în intervalul $[0.0, 1.0]$ (0.0 corespunde planului apropiat de decupare, iar 1.0 planului îndepărtat de decupare). Valoarea maximă iniţială pentru care se efectuează testarea poate fi modificată adăugând înainte de comanda `glClear (GL_DEPTH_BUFFER_BIT);` funcţia

```
glClearDepth (maxDepth);
```

În acest fel, testul de adâncime va fi efectuat doar pentru obiecte având adâncimea situată intervalul $[0, maxDepth]$. În particular, această funcţie este utilă atunci când scena pe care vrem să o reprezentăm conţine obiecte foarte îndepărtate de obiectele din prim plan.

O altă facilitate disponibilă în OpenGL este modificarea condiţiei de testare efectuată de rutinele de detectare a vizibilităţii. O astfel de condiţie poate fi specificată apleând funcţia

```
glDepthFunc (test);
```

Parametrul test poate avea ca valoare oricare dintre constantele simbolice

`GL_LESS, GL_GREATER, GL_EQUAL, GL_NOTEQUAL, GL_LEQUAL, GL_GEQUAL, GL_NEVER, GL_ALWAYS`.

Valoarea implicită a lui `test` este `GL_LESS`, cu alte cuvinte sunt testate obiectele având adâncimea mai mică decât $maxDepth$.

6.3 Efectul de ceaţă

Culoarea unui fragment este stabilită cu o formulă de tipul

$$C = f \cdot C_i + (1 - f) \cdot C_f. \tag{6.2}$$

Factorul ceaţă f este dependent de distanţă, $f = f(z)$; aceasta depinde de modelul utilizat (liniar, exponenţial, exponenţial pătratic).

Funcţia OpenGL specifică este

```
glFog* (parametru, valoare_parametru);
```

parametru	**valoare_parametru**
GL_FOG_MODE	GL_LINEAR GL_EXP GL_EXP2
GL_FOG_DENSITY (ρ)	
GL_FOG_START	
GL_FOG_END	
GL_FOG_COLOR	

6.4 Exerciţii

Exerciţiul 6.4.1 Se presupune că se utilizează modelul de amestecare în care factorul sursă are toate componentele egale cu `GL_SRC_ALPHA`, iar factorul destinaţie `GL_ONE_MINUS_SRC_ALPHA`. Se desenează un pătrat verde, apoi un pătrat roşu. Care va fi combinaţia RGB în zona de suprapunere, dacă fundalul este $(0.0, 0.0, 0.0, 1.0)$ şi ambele pătrate au componenta `ALPHA`$= 0.5$?

Exerciţiul 6.4.2 Care este valoarea implicită a funcţiei f (factor ceaţă) pentru $z = 3$?

Bibliografie

[1] G. Albeanu, *Grafica pe calculator. Algoritmi fundamentali*, Editura Universităţii din Bucureşti, 2001.

[2] W. Boehm şi H. Prautzsch, *Geometric Concepts for Geometric Design*, AK Peters, Wellesley, 1994.

[3] J. Foley, A. van Dam, S. Feiner şi J. Hughes, *Computer Graphics: Principles and Practice* (2nd edition in C), Addison Wesley, 1995.

[4] P. Schneider şi D. Eberly, *Geometric Tools for Computer Graphics*, Morgan Kaufmann, 2003.

---

[5] R. Baciu, *Programarea aplicaţiilor grafice 3D cu OpenGL*, Editura Albastră, 2005.

[6] D. Hearn şi M. Baker, *Computer Graphics with OpenGL*, Prentice Hall, 2003.

[7] G. Sellers, R. S. Wright Jr., N. Haemel, *OpenGL: SuperBible. Comprehensive Tutorial and Reference*, 7th edition, Addison-Wesley, 2015, http://www.openglsuperbible.com/

[8] P. Shirley, M. Ashikhmin, M. Gleicher, S. Marschner, E. Reinhard, K. Sung, W. Thompson şi P. Willemsen, *Fundamentals of Computer Graphics* (2nd edition), AK Peters, Wellesley, 2005.

[9] D. Shreiner, G. Sellers, J. Kessenich, B. Licea-Kane *OpenGL Programming Guide* (8th edition), Addison Wesley, 2013. http://www.opengl-redbook.com

---

[10] L. Bădescu, *Lecţii de Geometrie*, Editura Universităţii Bucureşti, 2000.

[11] G. Farin, *Curves and Surfaces for CAGD - A practical guide,* Academic Press, 2002. http://www.farinhansford.com/books/cagd/materials.html http://www.vis.uni-stuttgart.de/~kraus/LiveGraphics3D/cagd/

[12] L. Ornea şi A. Turtoi, *O introducere în geometrie*, Editura Theta, Bucureşti, 2000.

[13] H. Prautzsch, W. Boehm şi M. Paluszny, *Bézier and B-Spline Techniques*, Springer, 2002. http://i33www.ira.uka.de/applets/mocca/html/noplugin/inhalt.html