Grafică pe calculator

Mihai-Sorin Stupariu

Sem. al II-lea, 2015-2016

Cuprins

Capitolul 1

Generalităţi

1.1 Exemplu de program OpenGL

```
#include <windows.h> // biblioteci care urmeaza sa fie incluse.
#include <gl/freeglut.h> // nu trebuie uitat freeglut.h
void init (void) // initializare ecran de vizualizare
{
   glClearColor (1.0, 1.0, 1.0, 0.0); // culoarea de fond a ecranului
   glMatrixMode (GL_PROJECTION); // reprezentare 2D; proiectie ortogonala
   gluOrtho2D (0.0, 1200.0, 0.0, 700.0);
}
void desen (void) // procedura desenare
{
   glColor3f (0.0, 0.0, 1.0); // culoarea punctelor:  albastru
   glBegin (GL_POINTS);
      glVertex2i (20, 20);
      glVertex2i (21, 21);
      glVertex2i (22, 22);
      glVertex2i (23, 23);
      glVertex2i (24, 24);
      glVertex2i (27, 27);
      glVertex2i (100, 100);
   glEnd ( );
   glColor3f (1.0, 0.0, 0.0); // culoarea liniei frante:  rosu
   glBegin (GL_LINE_STRIP);
      glVertex2i (0,0);
      glVertex2i (200, 100);
      glVertex2i (300, 120);
      glVertex2i (255, 290);
   glEnd ( );
   glColor3f (0.0, 1.0, 0.0); // culoarea reuniunii de segmente:  verde
   glBegin (GL_LINES);
      glVertex2i (400,400);
      glVertex2i (600, 500);
      glVertex2i (700, 520);
      glVertex2i (655, 690);
   glEnd ( );
   glFlush ( );
}
void main (int argc, char** argv)
{
   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
```

```
    glutInitWindowPosition (100, 100); // pozitia initiala a ferestrei de vizualizare
    glutInitWindowSize (1200, 700); // dimensiunile ferestrei
    glutCreateWindow ("Puncte & Segmente"); // titlul ferestrei
    init ( );
    glClear (GL_COLOR_BUFFER_BIT);
    glutDisplayFunc (desen);
    glutMainLoop ( );
}
```

Elemente importante

- Directive preprocesare
- Procedură iniţializare `init`
- Procedură desen `desen`
- Main
  - Iniţializări GLUT
  - Generare fereastră
  - Apelare procedură iniţializare
  - Apelare procedură desen
  - 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 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: `glClearColor ( ); glFlush ( ); glVertex ( ); glColor ( ); glBegin ( )`, etc.
- 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: [12], [7]
- Cărţi / tutoriale online: D. Eck, Introduction to Computer Graphics; http://nehe.gamedev.net/; 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 `GL_POLYGON` să producă efectul dorit:

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).

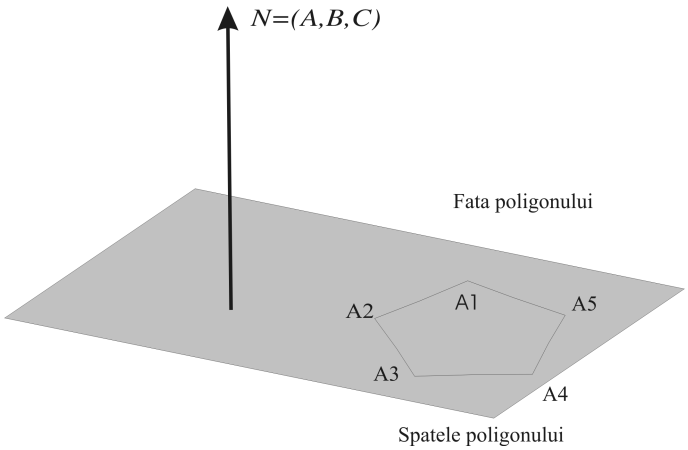

Figura 2.1: Orientarea unui poligon

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

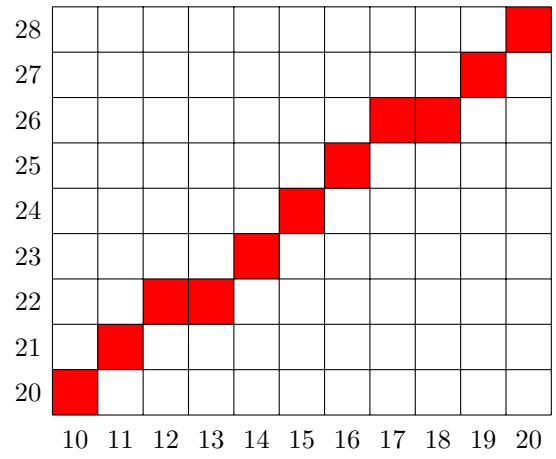

Figura 2.2: **Algoritmul DDA / algoritmul lui Bresenham.**
Pixelii selectaţi pentru a uni punctele $(10, 20)$ şi $(20, 28)$ sunt coloraţi cu roşu.

2.3 Funcţii OpenGL

2.3.1 Vârfuri

Primitivele grafice sunt realizate cu ajutorul **vârfurilor**. În OpenGL un vârf este definit cu ajutorul funcţiei

```
glVertex* (  );
```

Cu ajutorul sufixului * se indică:

- dimensiunea spaţiului în care lucrăm, $n \in \{2, 3, 4\}$
- tipul de date utilizat, care poate fi:
  - `i` (integer)
  - `s` (short)
  - `f` (float)
  - `d` (double)
- (opţional) posibila formă vectorială, indicată prin sufixul `v`.

Unui vârf îi sunt asociate:

- coordonatele (fac parte din definiţie),
- culoarea,
- normala (legată de funcţii de iluminare),
- coordonate de texturare

O funcţie de tipul `glVertex` poate fi apelată într-un cadru de tip

```
glBegin (*);

glEnd;
```

(unde * reprezintă tipul de primitivă generat).

2.3.2 Puncte

Astfel, cu ajutorul acestei funcţii pot fi generate puncte, într-un cadru de tipul

```
glBegin (GL_POINTS);

glEnd;
```

segmente de dreaptă (vezi paragraful 2.3.3) sau poligoane.

Exemplul 2.2 Următoarele două secvenţe de cod sursă sunt echivalente, având ca efect desenarea punctelor de coordonate (50,100), (70,80) şi (90,150).

I.

```
glBegin(GL_POINTS);
   glVertex2i (50, 100);
   glVertex2i (70, 80);
   glVertex2i (90, 150);
glEnd;
```

II.

```
int p1[ ] = {50, 100};
int p2[ ] = {70, 80};
int p3[ ] = {90, 150};
glBegin(GL_POINTS);
   glVertex2iv (p1);
   glVertex2iv (p2);
   glVertex2iv (p3);
glEnd;
```

Atribute ale punctelor

- Culoarea (dată de culoarea vârfului)
- Dimensiunea

```
glPointSize (dimens);
```

2.3.3 Segmente de dreaptă

Un singur segment de dreaptă, determinat de punctele de coordonate (x_1, y_1), (x_2, y_2) (vom lua $x_1, x_2, y_1, y_2 \in \mathbf{Z}$) poate fi trasat utilizând următoarea secvenţă de cod sursă:

```
glBegin (GL_LINES);
  glVertex2i (x1, y1);
  glVertex2i (x2, y2);
glEnd;
```

De asemenea, pentru trasarea segmentelor de dreaptă mai pot fi utilizate şi constantele simbolice `GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP`. Să considerăm o mulţime de puncte având coordonatele (pe care le presupunem numere întregi) $P_1 = (x_1, y_1), P_2 = (x_2, y_2), \ldots, P_n = (x_n, y_n)$.

• Putem desena segmentele de forma $[P_{2k+1}P_{2k+2}]$ (în cazul în care n este impar, ultimul punct nu este unit cu nimeni) folosind instrucţiunile

```
glBegin (GL_LINES);
  glVertex2i (x1, y1);
  glVertex2i (x2, y2);
  .........
  glVertex2i (xn, yn);
glEnd;
```

• Linia frântă ce uneşte punctele $P_1, P_2, \ldots, P_n$ este trasată astfel:

```
glBegin (GL_LINE_STRIP);
  glVertex2i (x1, y1);
  glVertex2i (x2, y2);
  .........
  glVertex2i (xn, yn);
glEnd;
```

• Linia poligonală închisă determinată de punctele $P_1, P_2, \ldots, P_n$ (poate conţine autointersecţii!) este desenată folosind comenzile:

```
glBegin (GL_LINE_LOOP);
  glVertex2i (x1, y1);
  glVertex2i (x2, y2);
  .........
  glVertex2i (xn, yn);
glEnd;
```

Atribute ale segmentelor de dreaptă

- Culoarea (dată de culorile extremităţilor segmentului); când acestea din urmă au culori diferite, culorile punctelor segmentului sunt determinate folosind **interpolarea afină**. Modul de trasare se controlează cu

  ```
  glShadeModel (...);
  ```
  ,

  fiind posibile variantele

  ```
  glShadeModel (GL_SMOOTH);
  ```

  ```
  glShadeModel (GL_FLAT);
  ```

- Lăţimea

  ```
  glLineWidth (width);
  ```

- Modul de desenare (şablon, model)

  Este definit prin

  ```
  glLineStipple(repeatfactor, pattern);
  ```

  şi activat / dezactivat prin

  ```
  glEnable (GL_LINE_STIPPLE);
  ......................
  glDisable (GL_LINE_STIPPLE);
  ```

2.3.4 Poligoane

Un singur triunghi, determinat de punctele de coordonate (x_1, y_1), (x_2, y_2), (x_3, y_3) (vom lua $x_1, x_2, y_1, y_2, x_3, y_3 \in \mathbf{Z}$), poate fi trasat utilizând următoarea secvenţă de cod sursă:

```
glBegin (GL_TRIANGLES);
  glVertex2i (x1, y1);
  glVertex2i (x2, y2);
  glVertex2i (x3, y3);
glEnd;
```

De asemenea, pentru trasarea segmentelor de dreaptă mai pot fi utilizate şi constantele simbolice `GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_QUADS, GL_QUAD_STRIP, GL_POLYGON`. Să considerăm o mulţime de puncte având coordonatele (pe care le presupunem numere întregi) $P_1 = (x_1, y_1), P_2 = (x_2, y_2), \ldots, P_n = (x_n, y_n)$.

• Putem desena triunghiuri de forma $[P_{3k+1}P_{3k+2}P_{3k+3}]$ (în cazul în care n nu este divizibil prin 3, există vârfuri care nu aparţin niciunui triunghi) folosind instrucţiunile

```
glBegin (GL_TRIANGLES);
  glVertex2i (x1, y1);
  glVertex2i (x2, y2);
  .........
  glVertex2i (xn, yn);
glEnd;
```

• Triunghiuri care au muchii in comun, definite de punctele $P_1, P_2, \ldots, P_n$ sunt trasate astfel:

```
glBegin (GL_TRIANGLE_STRIP);
  glVertex2i (x1, y1);
  glVertex2i (x2, y2);
  .........
  glVertex2i (xn, yn);
glEnd;
```

• Un evantai de triunghiuri având vârful P_1 comun este desenat folosind comenzile:

```
glBegin (GL_TRIANGLE_FAN);
  glVertex2i (x1, y1);
  glVertex2i (x2, y2);
  .........
  glVertex2i (xn, yn);
glEnd;
```

• Patrulatere pot fi desenate folosind instrucţiunile

```
glBegin (GL_QUADS);
  glVertex2i (x1, y1);
  glVertex2i (x2, y2);
  .........
  glVertex2i (xn, yn);
glEnd;
```

```
glBegin (GL_QUAD_STRIP);
  glVertex2i (x1, y1);
  glVertex2i (x2, y2);
  .........
  glVertex2i (xn, yn);
glEnd;
```

• Pentru poligoane convexe (vezi secţiunea 2.1.2 pentru condiţiile pe care trebuie să le verifice punctele) se poate folosi secveţa de cod

```
glBegin (GL_POLYGON);
  glVertex2i (x1, y1);
  glVertex2i (x2, y2);
  .........
  glVertex2i (xn, yn);
glEnd;
```

• Există o funcţie specială pentru desenarea unui dreptunghi care are ca vârfuri diagonal opuse punctele (x_1, y_1) şi (x_2, y_2).

```
glRect* (x1,y1,x2,y2)
```

Atribute ale triunghiurilor

- Culoarea (dată de culorile vârfurilor); când acestea din urmă au culori diferite, culorile punctelor segmentului sunt determinate folosind **interpolarea afină**. Modul de trasare se controlează, ca în cazul segmentelor, cu

  `glShadeModel (...);`,

  fiind posibile variantele

  ```
  glShadeModel (GL_SMOOTH);
  ```

  ```
  glShadeModel (GL_FLAT);
  ```

- Funcţii OpenGL pentru trasarea feţei / spatelui unui poligon

  – Precizarea modului de redare

  `glPolygonMode (`*face, mode*`);`

  Pentru *face* sunt posibile valorile `GL_FRONT, GL_BACK, GL_FRONT_AND_BACK`, iar pentru *mode* sunt posibile valorile `GL_POINTS, GL_LINE, GL_FILL`. Valoarea implicită este `GL_FILL`.

  – Schimbarea ordinii vârfurilor

  ```
  glFrontFace
  (GL_CW);
  ```

  ```
  glFrontFace(GL_CCW);
  ```

  Valoarea implicită este `GL_CCW` şi revenirea la aceasta trebuie precizată în mod explicit.

– Renunţare la trasarea feţei unor poligoane
Poligoanele văzute dinspre faţa *face* pot fi eliminate cu secvenţa

```
glEnable (GL_CULL_FACE);
glCullFace (face);
.....  listă poligoane convexe
glDisable (GL_CULL_FACE);
```

- Şablonul de desenare

2.4 Exerciţii

Exerciţiul 2.4.1 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.

Exerciţiul 2.4.2 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 2.4.3
a) Verificaţi echivalenţa dintre condiţiile (2.1) şi (2.2).
b) Care este interpretarea geometrică dacă egalitatea din condiţia (2.1) devine inegalitate?

Exerciţiul 2.4.4 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 2.4.5 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 2.4.6 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 2.4.7 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 2.4.8 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 2.4.9 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 2.4.10 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 2.4.11 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 2.4.12

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

Exerciţiul 2.4.14 Ilustraţi diferenţa de robusteţe şi de eficienţă pentru cele trei variante de algoritmi din seţiunea 2.2.

Exerciţiul 2.4.15 Care este secvenţa de cod sursă care generează culoarea galben pentru o primitivă grafică?

Exerciţiul 2.4.16 Câte triunghiuri vor fi desenate dacă între `glBegin(GL_TRIANGLES)` şi `glEnd( )` sunt enumerate 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 [3], [8] sau [4]. 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:

```
glMatrixMode (GL_MODELVIEW);
gluLookAt (x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);
```

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:

```
glMatrixMode (GL_PROJECTION);
glOrtho (xwmin, xwmax, ywmin, ywmax, dnear, dfar);
```

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

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

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:

```
glMatrixMode (GL_PROJECTION);
glFrustum (xwmin, xwmax, ywmin, ywmax, dnear, dfar);
```

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

```
glMatrixMode (GL_PROJECTION);
gluPerspective (theta, aspect, dnear, dfar);
```

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 Caracteristici ale surselor de lumină

Precizarea caracteristicilor unei surse de lumină se face cu ajutorul comenzii

```
glLight* (nume, proprietate, valoare)
```

Parametrii care pot fi luaţi în considerare pentru o sursă de lumină şi care sunt utilizaţi în formula (5.1), precum şi valorile lor implicite sunt descrişi în tabelul de mai jos:

Numele parametrului	**Valoarea implicită**
GL_AMBIENT	(0.0, 0.0, 0.0,1.0)
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0) sau (0.0, 0.0, 0.0,1.0)
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0) sau (0.0, 0.0, 0.0,1.0)
GL_POSITION	(0.0, 0.0, 1.0, 0.0)
GL_SPOT_DIRECTION (θ_l)	(0.0, 0.0, -1.0)
GL_SPOT_EXPONENT (a_l)	0.0
GL_SPOT_CUTOFF	180.0
GL_CONSTANT_ATTENUATION (a_0)	1.0
GL_LINEAR_ATTENUATION (a_1)	0.0
GL_QUADRATIC_ATTENUATION (a_2)	0.0

5.2 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 specular (separat de componenta ambientală şi cea difuză şi după texturare).

Selectarea proprietăţilor modelului de iluminare se face folosind comanda

```
glLightModel* (proprietate, valoare)
```

Tabelul de mai jos prezintă valorile parametrului `proprietate`.

Proprietatea modelului de iluminare	**Valoarea implicită**
GL_LIGHT_MODEL_AMBIENT	(0.2, 0.2, 0.2, 1.0)
GL_LIGHT_MODEL_LOCAL_VIEWER	GL_FALSE
GL_LIGHT_MODEL_TWO_SIDE	GL_FALSE
GL_LIGHT_MODEL_COLOR_CONTROL	GL_SINGLE_COLOR

5.3 Definirea caracteristicilor materialelor

Pentru fiecare obiect desenat trebuie precizată culoarea "materialului" din care este confecţionat, prin comanda

```
glMaterial* (fatasupr, proprietate, valoare)
```

Tabelul de mai jos prezintă valorile parametrului `proprietate`.

Proprietatea materialului	**Valoarea implicită**
GL_AMBIENT	(0.2, 0.2, 0.2, 1.0)
GL_DIFFUSE	(0.8, 0.8, 0.8, 1.0)
GL_SPECULAR	(0.0, 0.0, 0.0, 1.0)
GL_SHININESS	0.0
GL_EMISSION	(0.0, 0.0, 0.0, 1.0)
GL_COLOR_INDEXES	(0,1,1)

5.4 Formula generală de calcul

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

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 (se presupune că este activat modelul de iluminare cu observator local, i.e. `GL_LIGHT_MODEL_LOCAL_VIEWER` are valoarea `GL_TRUE` .

5.5 Umbre

5.6 Exerciţii

Exerciţiul 5.6.1 Care este vectorul de poziţie implicit pentru o sursă de lumină?

Exerciţiul 5.6.2 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 5.6.3 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 5.6.4 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 $z = 3$?

Capitolul 7

Texturare

7.1 Principii generale

Maparea texturilor presupune realizarea următoarelor patru etape:

1. crearea unei texturi sau a unui obiect textură şi specificarea caracteristicilor texturii;
2. indicarea modului în care textura este aplicată pe pixelii imaginii;
3. activarea texturării;
4. reprezentarea scenei, indicând atât coordonatele de texturare (s, t, r, q), cât şi cele geometrice (x, y, z, w).

7.2 Texturi 1D

În cazul 1-dimensional, coordonatele de texturare sunt situate în intervalul $[0.0, 1.0]$ (0.0 reprezintă începutul texturii, iar 1.0 sfârşitul acesteia). În momentul în care este apelată o funcţie de tipul

```
glTexCoord* (coord);
```

urmată de

```
glVertex* (pt);
```

coordonata `coord` a texturii este asociată vârfului `pt`.

Definirea unei texturi 1D se face apelând funcţia

```
glTexImage1D (.........);
```

având parametrii

- `target`: tipul de textură (`GL_TEXTURE_1D`);
- `level`: nivelul de texturare (0);
- `intformat`: numărul valorilor (de culoare) utilizate pentru fiecare pixel; în cazul codului RGBA acest număr este 4;

- `width`: lăţimea este o putere a lui 2; trebuie să fie coerentă şi consistentă cu modul în care a fost definită textura;
- `border`: este un număr egal cu 0,1 sau 2 şi indică numărul pixelilor de pe frontieră;
- `format`: poate fi o constantă simbolică de forma `GL_RGB`; `GL_RGBA`;
- `type`: tipul este indicat printr-o constantă simbolică `GL_UNSIGNED_BYTE`; `GL_BYTE`;
- `texels`: se indică unde este localizată textura.

7.3 Texturi 2D

În cazul texturilor 2D, prezentăm paşii urmaţi pentru manevrarea texturilor, care reflecă cele patru etape generale urmate pentru maparea texturilor.

7.3.1 Crearea unui obiect textură şi specificarea proprietăţilor acestuia

Obiectele textură memorează date referitoare la textură, făcându-le accesibile imediat. Fiecărui obiect îi este asociată o singură textură. Sunt parcurşi câţiva paşi intermediari, descrişi mai jos, împreună cu funcţiile OpenGL asociate.

- *Generarea numelor obiectelor textură*:

```
glGenTextures (n, *texNames);
```

  Prin această funcţie sunt *rezervaţi* n identificatori de textură în vectorul `texNames`, declarat explicit în procedura de iniţializare `init`. Rezervarea are ca semnificaţie faptul că numele din `texNames` sunt marcate ca fiind utilizate, ele primesc caracteristici efective atunci când sunt prima dată legate cu funcţiile specifice. **Observaţie.** De menţionat că funcţia OpenGL cu efect contrar celei descrise este funcţia de ştergere

```
glDeleteTextures (n, *texNames);
```

- *Crearea şi utilizarea obiectelor textură*:

```
glBindTexture (target, id);
```

  Aici `target` este parametrul care descrie dimensiunea texturii; fiind vorba de o textura 2-dimensională este `GL_TEXTURE_2D`, iar `id` este numele texturii. Prin această funcţie este *creat* un nou obiect textură, cu numele `id`. La apelarea lui `glBindTexture (...);` în cadrul funcţiei de desenare, textura este legată de obiectul desenat, ceea ce este corelat cu corespondenţa dintre coordonatele de texturare şi cele de modelare. De fapt, `glBindTexture (...)`; precizează obiectul textură activ.

- *Specificarea caracteristicilor texturii*:

```
glTexImage2D (.........);
```

având parametrii asemănători cu cei din cazul 1-dimensional

  - `target`: tipul de textură (`GL_TEXTURE_2D`);
  - `level`,`intformat`: similar cu cazul 1D;
  - `width, height`: lăţimea şi înălţimea texturii sunt de forma $2^w + b_w$, respectiv $2^h + b_h$ (altfel spus, puteri ale lui 2 la care se adaugă dimensiunile frontierei);
  - `border`: dimensiunea frontierei (b_w, b_h);
  - `format`, `type`, `texels`: similar cu cazul 1D.

Trebuie menţionat faptul că pot fi definite şi subtexturi, prin funcţia

```
glTexSubimage2D (.........);
```

având parametrii similari

- `target`: tipul de textură (`GL_TEXTURE_2D`);
- `level`: similar cu funcţia `glTexImage2D`;
- `xoffset, yoffset`: sunt numere întregi, pozitive, care precizează unde anume este aşezată subtextura în structura de texeli (cu convenţia că (0,0) este în stânga jos);
- `width, height`: lăţimea şi înălţimea subtexturii;
- `format`, `type`, `texels`: similar cu funcţia `glTexImage2D`.

Semnificaţia acesti funcţii este că se defineşte o structură prin care este înlocuită parţial / total o porţiune a texturii active (curente).

7.3.2 Precizarea unor reguli referitoare la modul în care texelii sunt aşezaţi pe structura de pixeli

Două reguli sunt luate în considerare: *repetarea texturii* şi *filtrare*. Ambele sunt legate de faptul că atât structura de texeli cât şi cea de pixeli sunt discrete, iar scopul lor este de a indica modul în care se procedează atunci când nu există o corespondenţă 1:1 între cele două structuri. Forma generală a funcţiei asociate este

```
glTexParameter* (target, pname, value);
```

Parametrul `target` indică tipul de textură, fiind `GL_TEXTURE_2D`. Ceilalţi parametri depind ca semnificaţie şi valoare de regula la care se face referire.

- *Repetarea texturii.* În acest caz (şi ţinând cont că suntem în context bidimensional) `pname` poate lua valorile

```
GL_TEXTURE_WRAP_S          GL_TEXTURE_WRAP_T
```

s, t sunt primele două coordonate ale texturii 2D, semnificaţia fiind că este precizată regula aplicată pentru coordonata indicată. În particular, pot fi utilizate reguli diferite pentru cele două coordonate de texturare.

Parametrul `value` poate avea una din valorile

`GL_CLAMP`	`GL_REPEAT`

În cazul lui `GL_CLAMP`, dacă se parcurge toată textura de-a lungul coordonatei considerate, valoarea ultimului texel este folosită în continuare pentru completarea tuturor pixelilor. În cazul lui `GL_REPEAT`, dacă se parcurge toată textura de-a lungul coordonatei considerate, se reporneşte parcurgerea texturii de la început, aceasta fiind repetată de câte ori este necesar.

- *Filtrare.* Aici apare explicit faptul ca se poate întâmpla ca la randarea unei scene sa nu existe o corespondenţă biunivocă între texeli şi pixeli. Astfel, se poate ca un pixel sa corespundă unei porţiuni mici a structurii de texeli sau, invers, un pixel sa corespundă unei structuri de texeli, iar rolul regulii de filtrare este de a stabili ce texel îi este asociat pixelului considerat. Parametrul `pname` poate avea, corespunzător celor două situaţii, valorile

`GL_TEXTURE_MAG_FILTER`	`GL_TEXTURE_MIN_FILTER`

Parametrul `value` poate avea una din valorile

`GL_NEAREST`	`GL_LINEAR`

Semnificaţia valorilor este următoarea: pentru `GL_NEAREST` se alege texelul având coordonatele cele mai apropiate de centrul pixelului, în timp ce pentru `GL_LINEAR` este utilizată o tehnică de tipul *moving window*, fiind considerat un tablou de 2×2 texeli apropiaţi de centrul pixelului, după care se face o mediere.

7.3.3 Coordonate de modelare şi coordonate de texturare

Coordonatele de modelare sunt cele utilizate în mod curent în funcţia de desenare. Fiecărui vârf îi sunt asociate, aşa cum se ştie, cele trei coordonate spaţiale (x, y, z). În cazul texturilor 2D coordonatele de texturare sunt s şi t, ambele în intervalul $[0.0, 1.0]$. Cu alte cuvinte, coordonatele de texturare posibile sunt reprezentate de pătratul $[0.0, 1.0] \times [0.0, 1.0]$. În cadrul funcţiei de desenare, fiecărui vârf îi sunt asociate anumite coordonate de texturare. Este suficient, pentru o primitivă grafică 2D, să fie indicate coordonatele de texturare pentru trei dintre vârfuri, coordonatele de texturare ale punctelor din interior rezultă automat. Motivaţie: dacă au fost fixate trei puncte necoliniare a, b, c în spaţiul de texturare şi trei puncte necoliniare A, B, C în spaţiul de modelare, există o unică aplicaţie afină care tranrsfomă punctele a, b, c in A, B, respectiv C.

În concluzie, o textură 2D este: (i) un tablou dreptunghiular de texeli având dimensiunile puteri ale lui 2; (ii) o mulţime înzestrată cu coordonate de texturare (pătratul standard). Atunci când tabloul de texeli are acelaşi număr de linii / coloane şi atunci când textura este aplicată direct pe un pătrat, corespondenţa dintre pixeli şi texeli este uşor de controlat şi nu apar fenomene de deformare.

În caz contrar (situaţie mai des întâlnită), trebuie manevraţi în mod convenabil parametrii disponibili.

Exemplu. Cum se poate proceda pentru a construi o textură care să acopere cât mai bine / fără deformări un dreptunghi $ABCD$ având raportul dimensiunilor $\frac{l}{L} = \frac{1}{3}$?

Fie h_T, l_T înălţimea şi lăţimea texturii (ambele sunt puteri ale lui 2). Se aleg h_T, l_T astfel ca $\frac{h_T}{l_T}$ să fie un raport de puteri ale lui 2 cât mai aproape de $\frac{1}{3}$: de exemplu luăm $h_T = 32, l_T = 64$, caz în care $\frac{h_T}{l_T} = \frac{1}{2}$. În continuare, se stabileşte înălţimea reală ξ până unde se umple textura, pe baza relaţiei

$$\frac{\xi}{l_T} = \frac{1}{3},$$

deci

$$\xi = \frac{l_T}{3} = \frac{64}{3} \approx 21.$$

În concluzie, pot fi umplute 22 de linii de texeli cu conţinutul efectiv al texturii, celorlalţi fiindu-le alocat negru. Asocierea coordonatelor de texturare pentru vârfurile dreptunghiului va fi

$$(0,0) \mapsto A, \quad (1,0) \mapsto B, \quad (1, \frac{2}{3}) \mapsto C, \quad (0, \frac{2}{3}) \mapsto D.$$

7.4 Exerciţii

Exerciţiul 7.4.1 Ce se întâmplă cu o textură 1D în cazul în care sunt asociate coordonate de texturare din afara intervalului $[0.0, 1.0]$?

Exerciţiul 7.4.2 Se presupune că punctelor $(8.0, 7.0)$, $(6.0, 11.0)$, $(13.0, 13.0)$ din spaţiul de modelare le sunt asociate coordonatele de texturare $(0.2, 0.4)$, $(0.6, 0.8)$, respectiv $(0.2, 0.2)$. Care sunt coordonatele de texturare ale punctului $(10, 11)$?

Capitolul 8

Reprezentarea unor cazuri particulare de curbe şi suprafeţe

8.1 Obiecte cuadrice

8.1.1 Conice şi cuadrice - breviar teoretic

Din motive de simetrie, vom nota coordonatele din plan cu x_1, x_2, iar pe cele din spaţiul tridimensional cu x_1, x_2, x_3. Descriem mai întâi conicele - locuri geometrice din plan, apoi, prin analogie, sunt introduse cuadricele.

Definiţii.

O **conică** (în planul $\mathbf{R}^2$) este o mulţime de puncte ale căror coordonate (x_1, x_2) verifică o ecuaţie de forma

$$a_{11}x_1^2 + 2a_{12}x_1x_2 + a_{22}x_2^2 + 2a_{13}x_1 + 2a_{23}x_2 + a_{33} = 0, \tag{8.1}$$

unde $(a_{ij})_{i,j}$ sunt coeficienţi reali astfel ca $(a_{11}, a_{12}, a_{22}) \neq (0, 0, 0)$.

Pentru conica descrisă de ecuaţia (8.1) vom nota

$$a = \begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix}, \qquad A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{pmatrix}.$$

Matricea a se numeşte **matricea conicei**, iar matricea A se numeşte **matricea extinsă a conicei**. Vom folosi, de asemenea, următoarele notaţii:

$$\delta := \det a, \quad \Delta := \det A, \quad r := \operatorname{rang} a, \quad R := \operatorname{rang} A.$$

Exemple.

(i) $\dfrac{x_1^2}{9} + \dfrac{x_2^2}{25} - 1 = 0$. Avem $\delta = \dfrac{1}{225}$, $\Delta = -\dfrac{1}{225}$, $r = 2$, $R = 3$.

(ii) $2x_1^2 + 8x_1x_2 + 10x_2^2 - 2x_1 + 2x_2 - 5 = 0$. Avem $\delta = 12$, $\Delta = -140$, $r = 2$, $R = 3$.

(iii) $x_1^2 - 2x_1x_2 - x_2^2 + 4x_1 - 4 = 0$. Avem $\delta = -2$, $\Delta = 12$, $r = 2$, $R = 3$.

(iv) $x_2^2 - 6x_1 = 0$. Avem $\delta = 0$, $\Delta = -9$, $r = 1$, $R = 3$.

(v) $x_1^2 + 2x_1x_2 + x_2^2 - 4 = 0$. Avem $\delta = 0$, $\Delta = 0$, $r = 1$, $R = 2$.

Definiţii.

O conică se numeşte **nedegenerată** dacă $\Delta \neq 0$. În cazul în care $\Delta = 0$, conica se numeşte **degenerată**.

Un punct $P_0 \in \mathbf{R}^2$ se numeşte **centru** al unei conice dacă simetria de centru P_0 invariază conica.

Un punct $P_O = (x_{10}, x_{20})$ este centru al conicei dacă şi numai dacă perechea (x_{10}, x_{20}) este soluţie a sistemului

$$\begin{cases} a_{11}x_{10} + a_{12}x_{20} + a_{13} = 0 \\ a_{21}x_{10} + a_{22}x_{20} + a_{23} = 0. \end{cases}$$

Rezultate:

(i) Mulţimea centrelor unei conice formează o varietate liniară (mulţimea vidă, un punct sau o dreaptă).

(ii) O conică are centru unic dacă şi numai dacă $\delta \neq 0$.

Propoziţia 8.1 (Clasificarea afină a conicelor)

(i) Numerele δ, Δ, r şi R asociate unei ecuaţii de forma (8.1) nu se modifică în urma unei schimbări afine de coordonate.

(ii) Printr-o schimbare de coordonate convenabil aleasă şi înmulţind, eventual, ecuaţia obţinută cu o constantă, orice ecuaţie de forma (8.1) poate fi adusă la una din formele de mai jos:

R	r	Forma canonică afină a conicei	Denumire
3	2	$x_1^2 + x_2^2 - 1 = 0$	Elipsă
		$x_1^2 - x_2^2 - 1 = 0$	Hiperbolă
		$-x_1^2 - x_2^2 - 1 = 0$	Elipsă vidă
3	1	$x_1^2 - 2x_2 = 0$	Parabolă
2	2	$x_1^2 + x_2^2 = 0$	Punct dublu
		$x_1^2 - x_2^2 = 0$	Pereche de drepte secante
2	1	$x_1^2 - 1 = 0$	Pereche de drepte paralele
		$-x_1^2 - 1 = 0$	Pereche de drepte vidă
1	1	$x_1^2 = 0$	Dreaptă dublă

Definiţie. O **cuadrică** este un loc geometric din spaţiul $\mathbb{R}^3$ dat prin anularea unui polinom de gradul II, adică printr-o ecuaţie analoagă lui (8.1), în care apar coordonatele x_1, x_2, x_3. În mod similar se construiesc matricele a, A şi definesc numerele r şi R, fiind aplicate considerente şi raţionamente analoage celor din cazul conicelor. Mai jos sunt câteva exemple de cuadrice scrise sub forma canonică.

R	r	Forma canonică afină a cuadricei	Denumire
4	3	$x_1^2 + x_2^2 + x_3^2 - 1 = 0$	Elipsoid
		$x_1^2 + x_2^2 - x_3^2 - 1 = 0$	Hiperboloid cu o pânză
		$x_1^2 - x_2^2 - x_3^2 - 1 = 0$	Hiperboloid cu două pânze
4	2	$x_1^2 - x_2^3 - 2x_3 = 0$	Paraboloid hiperbolic
3	3	$x_1^2 + x_2^2 - x_3^2 = 0$	Con
3	2	$x_1^2 + x_2^2 - 1 = 0$	Cilindru eliptic
		$x_1^2 - x_2^2 - 1 = 0$	Cilindru hiperbolic
3	1	$x_1^2 - 2x_2 = 0$	Cilindru parabolic
2	2	$x_1^2 + x_2^2 = 0$	Dreaptă dublă
		$x_1^2 - x_2^2 = 0$	Pereche de plane secante
2	1	$x_1^2 - 1 = 0$	Pereche de plane paralele
1	1	$x_1^2 = 0$	Plan dublu

8.1.2 Funcţii OpenGL pentru obiecte cuadrice

OpenGL are, în biblioteca GLU, funcţii dedicate pentru următoarele obiecte cuadrice: sferă, cilindru (trunchi de con), coroană circulară şi coroană circulară parţială. Pentru trasarea acestor obiecte sunt trei clase de funcţii OpenGL:

- *Gestionarea obiectelor cuadrice.* Crearea unui nou obiect cuadric se face cu ajutorul setului de funcţii

```
GLUquadricObj *qobj;
qobj=gluNewQuadric (  );
```

  Pentru ştergerea obiectului anterior creat `qobj` se foloseşte funcţia

```
gluDeleteQuadric (qobj) ;
```

- *Controlul proprietăţilor obiectului.* Sunt controlate două aspecte, iar funcţiile corespunzătoare trebuie să apară în codul sursă înaintea obiectelor propriu-zise.

  (i) Stilul de desenare a obiectului.

```
gluQuadricDrawStyle (qobj, drawstyle);
```

  Obiectul cuadric este indicat cu `obj`, iar `drawstyle` poate fi una dintre constantele simbolice `GLU_POINT, GLU_LINE, GLU_SILHOUETTE, GLU_FILL`. Stilul corespunzător lui `GLU_SILHOUETTE` este de a desena obiectul cu segmente de dreaptă, însă muchiile care separă feţe coplanare nu sunt desenate.

  (ii) Orientarea normalelor.

```
gluQuadricNormals (qobj, orientation);
```

  Obiectul cuadric este indicat prin `qobj`, iar orientarea (i.e. direcţia spre care sunt îndreptate normalele) este indicată folosind una dintre constantele simbolice `GLU_OUTSIDE, GLU_INSIDE`.

- *Desenarea propriu-zisă.*

  O *sferă* cu centrul în $(0, 0, 0)$, de rază `r`, pentru care sunt desenate `nLong` meridiane şi `nLat` paralele este desenată cu

```
gluSphere (qobj, r, nLong, nLat);
```

  Un *trunchi de con* (în cazuri particulare devine cilindru) având razele celor două baze `rBase`, respectiv `rTop`, înălţimea `height`, pentru care discurile situate în partea superioară şi inferioară nu sunt desenate, este randat cu

```
gluCylinder (qobj, rBase, rTop, height, nLong, nLat);
```

  O *coroană circulară* situată în planul orizontal, având razele celor două cercuri care o determină `rInner` şi `rOuter` şi centrul în origine este desenată cu

```
gluDisk (qobj, rInner, rOuter, slices, rings);
```

Mai general, se poate desena şi un *sector al unei coroane circulare* ca mai sus, prin indicarea unghiurilor de în ceput şi de sfârşit `startA`, `sweepA`, măsurate în grade, în sensul acelor de ceasornic:

```
gluDisk (qobj, rInner, rOuter, slices, rings, startA, sweepA);
```

8.2 Curbe şi suprafeţe Bézier

8.2.1 Curbe şi suprafeţe Bézier - breviar teoretic

Definiţii.

Pentru $n \in \mathbb{N}$ fixat, **polinoamele Bernstein de grad** n sunt definite prin

$$B_i^n(t) = C_n^i t^i (1-t)^{n-i}, \quad i \in \{0, \ldots, n\},$$

unde $C_n^i = \frac{n!}{i!(n-i)!}$. Prin convenţie, definim $B_i^n(t) = 0$, dacă $i \notin \{0, \ldots, n\}$.

Fie $(\mathbf{b}_0, \ldots, \mathbf{b}_n)$ o mulţime ordonată de puncte din $\mathbb{R}^m$, numită **poligon de control**. **Curba Bézier** $\mathbf{b} : [0,1] \to \mathbb{R}^m$ definită de poligonul de control $(\mathbf{b}_0, \ldots, \mathbf{b}_n)$ este dată de formula

$$\mathbf{b}(t) := \sum_{i=0}^{n} B_i^n(t)\mathbf{b}_i. \tag{8.2}$$

Exemplu Considerăm poligonul de control

$$\mathbf{b}_0 = (1,0), \quad \mathbf{b}_1 = (1,1), \quad \mathbf{b}_2 = (0,2).$$

Curba Bézier asociată $\mathbf{b} : [0,1] \to \mathbb{R}^2$ se scrie sub forma Bernstein

$$\mathbf{b}(t) = \sum_{i=0}^{2} B_i^2(t)\mathbf{b}_i = (1-t)^2(1,0) + 2t(1-t)(1,1) + t^2(0,2) =$$

$$(1 - 2t + t^2 + 2t - 2t^2, 2t - 2t^2 + 2t^2) = (1 - t^2, 2t).$$

Avem, de exemplu, $\mathbf{b}(\frac{1}{3}) = (\frac{8}{9}, \frac{2}{3})$, $\mathbf{b}(\frac{1}{4}) = (\frac{15}{16}, \frac{1}{2})$, etc.

Algoritmul de Casteljau

Fie $\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbf{R}^m$. Pentru $t \in [0,1]$ se notează $\mathbf{b}_i^0(t) := \mathbf{b}_i$ $(i = 0, \ldots, n)$ şi se definesc inductiv punctele de Casteljau

$$\mathbf{b}_i^r(t) := (1-t)\mathbf{b}_i^{r-1}(t) + t\mathbf{b}_{i+1}^{r-1}(t), \quad \begin{cases} r = 1, \ldots, n \\ i = 0, \ldots, n-r \end{cases} \tag{8.3}$$

Teoremă. Punctul $\mathbf{b}_0^n(t)$ descrie, când t variază, curba Bézier asociată poligonului de control $(\mathbf{b}_0, \ldots, \mathbf{b}_n)$ dată de ecuaţia (8.2).

Proprietăţi elementare

Fie $(\mathbf{b}_0, \ldots, \mathbf{b}_n)$ un poligon de control din $\mathbf{R}^m$. Curba Bézier asociată $\mathbf{b} : [0,1] \to \mathbf{R}^m$ are următoarele proprietăţi:

(i) $\mathbf{b}$ este o curbă polinomială, având gradul mai mic sau egal cu n;

(ii) curba $\mathbf{b}$ interpolează extremităţile poligonului de control, i.e. $\mathbf{b}(0) = \mathbf{b}_0$, $\mathbf{b}(1) = \mathbf{b}_n$; în particular, dacă poligonul de control este închis, curba Bézier asociată este închisă;

(iii) **proprietatea acoperirii convexe**: punctele curbei Bézier $\mathbf{b}$ se află în acoperirea convexă a punctelor de control;

(iv) **invarianţa la schimbări afine de parametru**: dacă $\varphi : [0,1] \to [\alpha, \beta]$, $\varphi(t) = \alpha + t(\beta - \alpha)$ este o schimbare afină de parametru şi dacă $\mathbf{b}^{[\alpha,\beta]}$ este curba Bézier asociată poligonului de control $(\mathbf{b}_0, \ldots, \mathbf{b}_n)$, dar definită pe intervalul $[\alpha, \beta]$, atunci $\mathbf{b} = \mathbf{b}^{[\alpha,\beta]} \circ \varphi$;

(v) **invarianţă afină**: dacă $\tau : \mathbf{R}^m \to \mathbf{R}^m$ este o aplicaţie afină, atunci curba Bézier asociată poligonului de control date de $(\tau(\mathbf{b}_0), \ldots, \tau(\mathbf{b}_n))$ este curba $\tau(\mathbf{b}^n)$;

(vi) **(Invarianţa la combinaţii baricentrice)**: fie $(\mathbf{b}_0, \ldots, \mathbf{b}_n)$, respectiv $(\widetilde{\mathbf{b}}_0, \ldots, \widetilde{\mathbf{b}}_n)$ două poligoane de control şi $\mathbf{b}$, respectiv $\widetilde{\mathbf{b}}$ curbele Bézier corespunzătoare. Pentru orice $\alpha \in \mathbf{R}$, curba Bézier asociată poligonului de control $((1-\alpha)\mathbf{b}_0 + \alpha\widetilde{\mathbf{b}}_0, \ldots, (1-\alpha)\mathbf{b}_n + \widetilde{\mathbf{b}}_n)$ este curba $(1-\alpha)\mathbf{b} + \alpha\widetilde{\mathbf{b}}$.

(vii) dacă $\widetilde{\mathbf{b}} : [0,1] \to \mathbf{R}^m$ este curba Bézier asociată poligonului de control $(\mathbf{b}_n, \ldots, \mathbf{b}_0)$, atunci $\widetilde{\mathbf{b}}(t) = \mathbf{b}(1-t)$, în particular, cele două curbe au aceeaşi imagine geometrică.

Suprafeţe Fie $m, n \in \mathbb{N}^*$ două numere naturale nenule şi

$$\begin{pmatrix} \mathbf{b}_{00} & \mathbf{b}_{01} & \ldots & \mathbf{b}_{0n} \\ \mathbf{b}_{10} & \mathbf{b}_{11} & \ldots & \mathbf{b}_{1n} \\ \ldots & \ldots & \ldots & \ldots \\ \mathbf{b}_{m0} & \mathbf{b}_{m1} & \ldots & \mathbf{b}_{mn} \end{pmatrix}$$

o matrice ale cărei elemente sunt puncte din $\mathbb{R}^3$, numită **reţea Bézier (poliedru de control)**. **Suprafaţa Bézier de tip produs tensorial** asociată acestor date este dată de formula:

$$\mathbf{s} : [0,1] \times [0,1] \to \mathbb{R}^3, \quad \mathbf{s}(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} B_i^m(u) B_j^n(v) \mathbf{b}_{ij}.$$

Proprietăţi elementare

(i) Prin analogie cu curbele Bézier, suprafaţa de tip produs tensorial are următoarele proprietăţi:

- interpolarea punctelor $\mathbf{b}_{00} = \mathbf{s}(0,0), \mathbf{b}_{0n} = \mathbf{s}(0,1), \mathbf{b}_{m0} = \mathbf{s}(1,0), \mathbf{b}_{mn} = \mathbf{s}(1,1)$;
- imaginea suprafeţei este inclusă în acoperirea convexă a punctelor poliedrului de control;
- invarianţă la transformări afine.

(ii) Curbele frontieră, i.e. curbele de coordonate $\mathbf{s}(0,\cdot)$, $\mathbf{s}(1,\cdot)$, $\mathbf{s}(\cdot,0)$ şi $\mathbf{s}(\cdot,1)$ sunt curbe Bézier având poligoane de control respectiv $(\mathbf{b}_{00}, \mathbf{b}_{01}, \ldots, \mathbf{b}_{0n})$, $(\mathbf{b}_{m0}, \mathbf{b}_{m1}, \ldots, \mathbf{b}_{mn})$, $(\mathbf{b}_{00}, \mathbf{b}_{10}, \ldots, \mathbf{b}_{m0})$, $(\mathbf{b}_{0n}, \mathbf{b}_{1n}, \ldots, \mathbf{b}_{mn})$. Restul curbelor de coordonate sunt, la rândul lor, curbe Bézier. Totuşi, acestea din urmă *nu* au drept puncte de control linii sau coloane din matricea $(\mathbf{b}_{ij})_{i,j}$.

Exemplu. Dacă $m = n = 1$ avem

$$\mathbf{s}(u,v) = \sum_{i=0}^{1} \sum_{j=0}^{1} B_i^1(u) B_j^1(v) \mathbf{b}_{ij} = \begin{pmatrix} 1-u & u \end{pmatrix} \begin{pmatrix} \mathbf{b}_{00} & \mathbf{b}_{01} \\ \mathbf{b}_{10} & \mathbf{b}_{11} \end{pmatrix} \begin{pmatrix} 1-v \\ v \end{pmatrix}.$$

De exemplu, dacă

$$\mathbf{b}_{00} = (0,0,0), \quad \mathbf{b}_{01} = (1,0,0), \quad \mathbf{b}_{10} = (0,1,0), \quad \mathbf{b}_{11} = (1,1,1),$$

un calcul direct arată că $\mathbf{s}(u,v) = (v,u,uv)$.

8.2.2 Evaluatori

Atât curbele cât şi suprafeţele Bézier sunt obţinute folosind tehnici de interpolare, care are loc la nivelul coordonatelor vârfurilor. Utilizând funcţii specifice OpenGL în loc de `glVertex ( )` (de exemplu `glNormal ( )`, `glColor ( )`, `glTexCoord* ( )`), folosind aceeaşi regulă de interpolare sunt determinate, pe lângă punctele curbei, proprietăţile lor, acestea fiind apoi manevrate ca nişte vârfuri obişnuite.

Atât în cazul curbelor, cât şi al suprafeţelor, funcţiile asociate pot fi grupate în două categorii: pentru definire şi pentru evaluare. Mai jos sunt prezentate funcţiile folosite în cazul curbelor; în cazul suprafeţelor acestea sunt similare.

Funcţii pentru definire.

```
glMap1* (target, umin, umax, stride, order, *points);
```

unde `target` poate fi una dintre constantele simbolice `GL_MAP1_VERTEX3`, `GL_MAP1_COLOR4`, `GL_MAP1_NORMAL`, etc. (în funcţie de ceea ce se interpolează); `umin`, `umax` sunt capetele intervalului de definiţie, iar `stride`, `order` sunt parametri legaţi de gradul curbei.
urmată de activare

```
glEnable (target);
```

Funcţie pentru evaluare

```
glEvalCoord1*(u);
```

este apelată în cadrul unei funcţii pentru trasarea unei primitive standard.

8.3 Exerciţii

Exerciţiul 8.3.1 Considerăm poligonul de control

$$\mathbf{b}_0 = (1,1), \quad \mathbf{b}_2 = (2,0), \quad \mathbf{b}_3 = (0,0)$$

şi fie $\mathbf{b} : [0,1] \to \mathbf{R}^2$ curba Bézier asociată. Calculaţi $\mathbf{b}(\frac{1}{3})$ şi stabiliţi dacă punctul $(1, \frac{1}{3})$ aparţine imaginii lui $\mathbf{b}$.

Exerciţiul 8.3.2 Considerăm punctele $\mathbf{b}_0 = (2,4), \mathbf{b}_1 = (4,2)$ şi $b_2 = (4,0)$. Calculaţi punctele $\mathbf{b}_0^1(t)$, $\mathbf{b}_1^1(t)$ şi $\mathbf{b}_0^2(t)$ corespunzătoare valorilor $t = \frac{1}{2}$ şi $t = \frac{1}{4}$.

Exerciţiul 8.3.3 Considerăm reţeaua Bézier

$$\mathbf{b}_{00} = (0,0,0), \quad \mathbf{b}_{01} = (2,0,0), \quad \mathbf{b}_{10} = (0,2,0), \quad \mathbf{b}_{11} = (2,2,0).$$

Determinaţi punctul $\mathbf{s}(\frac{1}{2}, \frac{1}{2})$.

Bibliografie

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

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

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

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

[5] 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/`

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

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

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

[9] 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`

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

[11] 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.

[12] D. Shreiner, M. Woo, J. Neider şi T. Davis. *OpenGL Programming Guide* (6th edition), Addison Wesley, 2008.
`http://www.opengl-redbook.com`

[13] `http://www.opengl.org`