

Grafica pe calculator. Preliminarii

Mihai-Sorin Stupariu

Sem. I, 2022 - 2023

Motivație

Aspecte introductive

Despre OpenGL

Exemple de programe

De ce un curs de grafică? / Aplicații?

- ▶ Filme.

De ce un curs de grafică? / Aplicații?

- ▶ Filme.
- ▶ Dezvoltarea de jocuri.

De ce un curs de grafică? / Aplicații?

- ▶ Filme.
- ▶ Dezvoltarea de jocuri.
- ▶ Imagistică medicală.

Ce înseamnă un curs de grafică?

- **Abordări posibile:**

Ce înseamnă un curs de grafică?

► Abordări posibile:

- background + dezvoltare “low level” (de exemplu *OpenGL*)



Figura: Sursa: <https://www.iamag.co/real-time-graphics-in-pixar-film-production/>

Ce înseamnă un curs de grafică?

► Abordări posibile:

- background + dezvoltare “low level” (de exemplu *OpenGL*)



Figura: Sursa: <https://www.iamag.co/real-time-graphics-in-pixar-film-production/>

► tehnologii dedicate (de exemplu *Unity*)

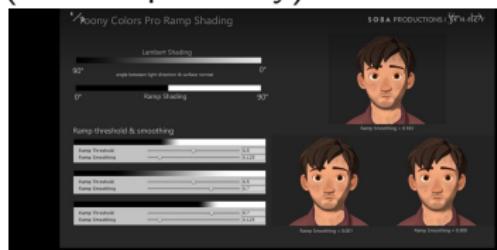


Figura: Sursa: <https://unity.com/madewith/sonder>

Cunoștințe necesare?

- ▶ cunoștințe elementare de programare în C++ (inclusiv utilizarea unui mediu de dezvoltare integrat) - vom folosi **Microsoft Visual Studio**;
- ▶ elemente de bază de Algebră liniară și Geometrie analitică.

Ce teme vom aborda la curs+laborator? Care ar fi obiectivul?

- Primitive grafice

Ce teme vom aborda la curs+laborator? Care ar fi obiectivul?

- Primitive grafice
- Transformări

Ce teme vom aborda la curs+laborator? Care ar fi obiectivul?

- Primitive grafice
- Transformări
- Texturare

Ce teme vom aborda la curs+laborator? Care ar fi obiectivul?

- Primitive grafice
- Transformări
- Texturare
- Reprezentarea scenelor 3D

Ce teme vom aborda la curs+laborator? Care ar fi obiectivul?

- Primitive grafice
- Transformări
- Texturare
- Reprezentarea scenelor 3D
- Iluminare

Ce teme vom aborda la curs+laborator? Care ar fi obiectivul?

- Primitive grafice
- Transformări
- Texturare
- Reprezentarea scenelor 3D
- Iluminare
- Efecte vizuale

Ce teme vom aborda la curs+laborator? Care ar fi obiectivul?

- Primitive grafice
- Transformări
- Texturare
- Reprezentarea scenelor 3D
- Iluminare
- Efecte vizuale
- Ideal: la finalul cursului să puteți dezvolta o aplicație complexă 3D

Resurse

- ▶ Site-ul oficial OpenGL

Resurse

- ▶ Site-ul oficial OpenGL
- ▶ Cărți:
 - D. Hearn si M. Baker, *Computer Graphics with OpenGL*, 2003
 - D. Shreiner, M.Woo, J. Neider si T. Davis. *OpenGL Programming Guide* (2nd edition), Addison Wesley, 1997
 - D. Shreiner, G. Sellers, J. Kessenich, B. Licea-Kane *OpenGL Programming Guide* (9th edition), Addison Wesley, 2016
 - G. Sellers, R. S. Wright Jr., N. Haemel, *OpenGL: SuperBible. Comprehensive Tutorial and Reference* (7th edition), Addison-Wesley, 2015,

Resurse

- ▶ Site-ul oficial OpenGL
- ▶ Cărți:
 - D. Hearn si M. Baker, *Computer Graphics with OpenGL*, 2003
 - D. Shreiner, M.Woo, J. Neider si T. Davis. *OpenGL Programming Guide* (2nd edition), Addison Wesley, 1997
 - D. Shreiner, G. Sellers, J. Kessenich, B. Licea-Kane *OpenGL Programming Guide* (9th edition), Addison Wesley, 2016
 - G. Sellers, R. S. Wright Jr., N. Haemel, *OpenGL: SuperBible. Comprehensive Tutorial and Reference* (7th edition), Addison-Wesley, 2015,
- ▶ Cărți / tutoriale online:
 - <http://learnopengl.com/>
 - <http://www.opengl-tutorial.org/>
 - D. Eck, *Introduction to Computer Graphics*
 - <http://nehe.gamedev.net/>
 - A. Gerdan - *OpenGL 4 tutorials*;
 - etc.

Elemente generale

- OpenGL este o interfață de programare (Application Programming Interface API)

Elemente generale

- ▶ OpenGL este o interfață de programare (Application Programming Interface API)
- ▶ Pentru a funcționa are nevoie de o serie de biblioteci; versiunea suportată depinde de placa grafică a calculatorului - se poate folosi codul sursa `00_test_version.cpp`

Elemente generale

- ▶ OpenGL este o interfață de programare (Application Programming Interface API)
- ▶ Pentru a funcționa are nevoie de o serie de biblioteci; versiunea suportată depinde de placa grafică a calculatorului - se poate folosi codul sursa `00_test_version.cpp`
- ▶ OpenGL **nu** este un limbaj de programare (există GLSL)

Elemente generale

- ▶ OpenGL este o interfață de programare (Application Programming Interface API)
- ▶ Pentru a funcționa are nevoie de o serie de biblioteci; versiunea suportată depinde de placa grafică a calculatorului - se poate folosi codul sursa `00_test_version.cpp`
- ▶ OpenGL **nu** este un limbaj de programare (există GLSL)
- ▶ Arhitectura de tip *client-server* (CPU-GPU) - element cheie: “comunicarea” dintre cele două componente *hardware*

Scurt istoric

- ▶ 1992: versiunea 1.0 a OpenGL (derivat din IRIS GL), lansat de SGI; “open standard”

Scurt istoric

- ▶ 1992: versiunea 1.0 a OpenGL (derivat din IRIS GL), lansat de SGI; “open standard”
- ▶ 1992: OpenGL Architecture Review Board (elaborarea specificațiilor)

Scurt istoric

- ▶ 1992: versiunea 1.0 a OpenGL (derivat din IRIS GL), lansat de SGI;
“open standard”
- ▶ 1992: OpenGL Architecture Review Board (elaborarea specificațiilor)
- ▶ 1997: versiunea 1.1

Scurt istoric

- ▶ 1992: versiunea 1.0 a OpenGL (derivat din IRIS GL), lansat de SGI; “open standard”
- ▶ 1992: OpenGL Architecture Review Board (elaborarea specificațiilor)
- ▶ 1997: versiunea 1.1
- ▶ 2002/2003: versiunile 1.4, 1.5; GLSL (GL Shading Language) apare ca o extensie a funcționalității de bază

Scurt istoric

- ▶ 1992: versiunea 1.0 a OpenGL (derivat din IRIS GL), lansat de SGI; “open standard”
- ▶ 1992: OpenGL Architecture Review Board (elaborarea specificațiilor)
- ▶ 1997: versiunea 1.1
- ▶ 2002/2003: versiunile 1.4, 1.5; GLSL (GL Shading Language) apare ca o extensie a funcționalității de bază
- ▶ 2004: **OpenGL 2.0; GLSL 1.10.59 inclus în “core”**

Scurt istoric

- ▶ 1992: versiunea 1.0 a OpenGL (derivat din IRIS GL), lansat de SGI; “open standard”
- ▶ 1992: OpenGL Architecture Review Board (elaborarea specificațiilor)
- ▶ 1997: versiunea 1.1
- ▶ 2002/2003: versiunile 1.4, 1.5; GLSL (GL Shading Language) apare ca o extensie a funcționalității de bază
- ▶ 2004: **OpenGL 2.0; GLSL 1.10.59 inclus în “core”**
- ▶ 2008: OpenGL 3.0: conceptul de “funcționalități depreciate”, legate de modul de redare imediat; mecanismul de depreciere a fost implementat / extins în 2009, când au fost lansate versiunile 3.1 și 3.2

Scurt istoric

- ▶ 1992: versiunea 1.0 a OpenGL (derivat din IRIS GL), lansat de SGI; “open standard”
- ▶ 1992: OpenGL Architecture Review Board (elaborarea specificațiilor)
- ▶ 1997: versiunea 1.1
- ▶ 2002/2003: versiunile 1.4, 1.5; GLSL (GL Shading Language) apare ca o extensie a funcționalității de bază
- ▶ 2004: **OpenGL 2.0; GLSL 1.10.59 inclus în “core”**
- ▶ 2008: OpenGL 3.0: conceptul de “funcționalități depreciate”, legate de modul de redare imediat; mecanismul de depreciere a fost implementat / extins în 2009, când au fost lansate versiunile 3.1 și 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)

Scurt istoric

- ▶ 1992: versiunea 1.0 a OpenGL (derivat din IRIS GL), lansat de SGI; “open standard”
- ▶ 1992: OpenGL Architecture Review Board (elaborarea specificațiilor)
- ▶ 1997: versiunea 1.1
- ▶ 2002/2003: versiunile 1.4, 1.5; GLSL (GL Shading Language) apare ca o extensie a funcționalității de bază
- ▶ 2004: **OpenGL 2.0; GLSL 1.10.59 inclus în “core”**
- ▶ 2008: OpenGL 3.0: conceptul de “funcționalități depreciate”, legate de modul de redare imediat; mecanismul de depreciere a fost implementat / extins în 2009, când au fost lansate versiunile 3.1 și 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)

Scurt istoric

- ▶ 1992: versiunea 1.0 a OpenGL (derivat din IRIS GL), lansat de SGI; “open standard”
- ▶ 1992: OpenGL Architecture Review Board (elaborarea specificațiilor)
- ▶ 1997: versiunea 1.1
- ▶ 2002/2003: versiunile 1.4, 1.5; GLSL (GL Shading Language) apare ca o extensie a funcționalității de bază
- ▶ 2004: **OpenGL 2.0; GLSL 1.10.59 inclus în “core”**
- ▶ 2008: OpenGL 3.0: conceptul de “funcționalități depreciate”, legate de modul de redare imediat; mecanismul de depreciere a fost implementat / extins în 2009, când au fost lansate versiunile 3.1 și 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)
- ▶ 2017 OpenGL 4.6 (respectiv GLSL 4.60)

Biblioteci utilizate de OpenGL și funcții asociate

- ▶ bibliotecă fundamentală (core library): este independentă de platforma pe care se lucrează; funcțiile corespunzătoare au prefixul gl. De exemplu: (i) `glClearColor ()`; `glFlush ()` – comune; (ii) `glVertex ()`; `glColor ()`; `glBegin ()` – depreciate; (iii) `glGenVertexArrays ()`; `glDrawArrays ()` – OpenGL nou.

Biblioteci utilizate de OpenGL și funcții asociate

- ▶ bibliotecă fundamentală (core library): este independentă de platforma pe care se lucrează; funcțiile corespunzătoare au prefixul gl. De exemplu: (i) `glClearColor` (); `glFlush` () – comune; (ii) `glVertex` (); `glColor` (); `glBegin` () – depreciate; (iii) `glGenVertexArrays` (); `glDrawArrays` () – OpenGL nou.
- ▶ GLU (OpenGL Utility): conține mai ales proceduri / funcții legate de proiecție, precum și funcții pentru conice și cuadrice; funcțiile asociate au prefixul glu

Biblioteci utilizate de OpenGL și funcții asociate

- ▶ bibliotecă fundamentală (core library): este independentă de platforma pe care se lucrează; funcțiile corespunzătoare au prefixul gl. De exemplu: (i) `glClearColor()`; `glFlush()` – comune; (ii) `glVertex()`; `glColor()`; `glBegin()` – depreciate; (iii) `glGenVertexArrays()`; `glDrawArrays()` – OpenGL nou.
- ▶ GLU (OpenGL Utility): conține mai ales proceduri / funcții legate de proiecție, precum și funcții pentru conice și cuadrice; funcțiile asociate au prefixul glu
- ▶ GLUT (OpenGL Utility Toolkit) / freeglut: bibliotecă *dependentă de sistem*, utilizată pentru a realiza fereastra de vizualizare; poate interacționa cu sisteme de operare bazate pe ferestre de vizualizare; funcțiile specifice au prefixul glut. Există și alte biblioteci / pachete, cum ar fi GLFW.

Exemplu de program care utilizează OpenGL “vechi”

Codul sursă 01_01_oldOpenGL.cpp

// Directive preprocessare

Exemplu de program care utilizează OpenGL “vechi”

Codul sursă 01_01_oldOpenGL.cpp

```
// Directive preprocessare  
// Procedură inițializare – init
```

Exemplu de program care utilizează OpenGL “vechi”

Codul sursă 01_01_oldOpenGL.cpp

```
// Directive preprocessare  
// Procedură inițializare – init  
// Procedură desen – desen
```

Exemplu de program care utilizează OpenGL “vechi”

Codul sursă 01_01_oldOpenGL.cpp

```
// Directive preprocessare  
// Procedură inițializare – init  
// Procedură desen – desen  
// Main
```

Exemplu de program care utilizează OpenGL “vechi”

Codul sursă 01_01_oldOpenGL.cpp

```
// Directive preprocessare
// Procedură inițializare – init
// Procedură desen – desen
// Main
    // Inițializări GLUT
    // Generare fereastră
    // Apelare procedură inițializare
    // Apelare procedură desen
    // Apelare glutMainLoop
```

Exemplu de program care utilizează OpenGL “nou”

Codul sursă 01_02_newOpenGL.cpp

// Directive preprocessare

Exemplu de program care utilizează OpenGL “nou”

Codul sursă 01_02_newOpenGL.cpp

```
// Directive preprocessare  
// Shader-e, date (coordonate, culori)
```

Exemplu de program care utilizează OpenGL “nou”

Codul sursă 01_02_newOpenGL.cpp

```
// Directive preprocessare
// Shader-e, date (coordonate, culori)
// Proceduri (initializare, desen)
```

Exemplu de program care utilizează OpenGL “nou”

Codul sursă 01_02_newOpenGL.cpp

```
// Directive preprocessare
// Shader-e, date (coordonate, culori)
// Proceduri (initializare, desen)
// Main
```

Exemplu de program care utilizează OpenGL “nou”

Codul sursă 01_02_newOpenGL.cpp

```
// Directive preprocessare
// Shader-e, date (coordonate, culori)
// Proceduri (inițializare, desen)
// Main
    // Inițializări GLUT
    // Generare fereastră
    // Apelare procedură inițializare
        // Creare VBO / VAO
        // Creare / Apelare shader-e
```

Exemplu de program care utilizează OpenGL “nou”

Codul sursă 01_02_newOpenGL.cpp

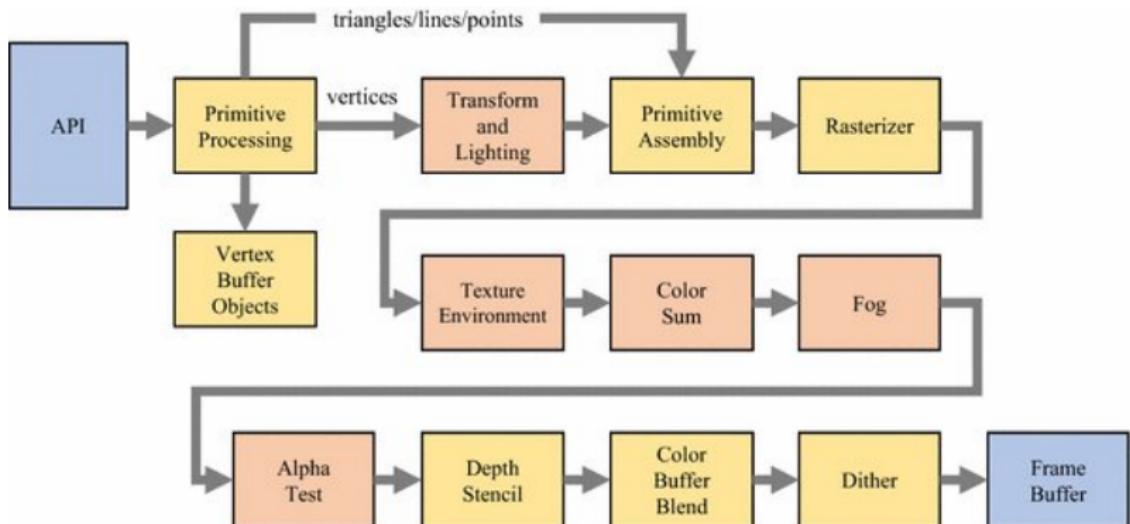
```
// Directive preprocessare
// Shader-e, date (coordonate, culori)
// Proceduri (inițializare, desen)
// Main
    // Inițializări GLUT
    // Generare fereastră
    // Apelare procedură inițializare
        // Creare VBO / VAO
        // Creare / Apelare shader-e
    // Apelare procedură desen
```

Exemplu de program care utilizează OpenGL “nou”

Codul sursă 01_02_newOpenGL.cpp

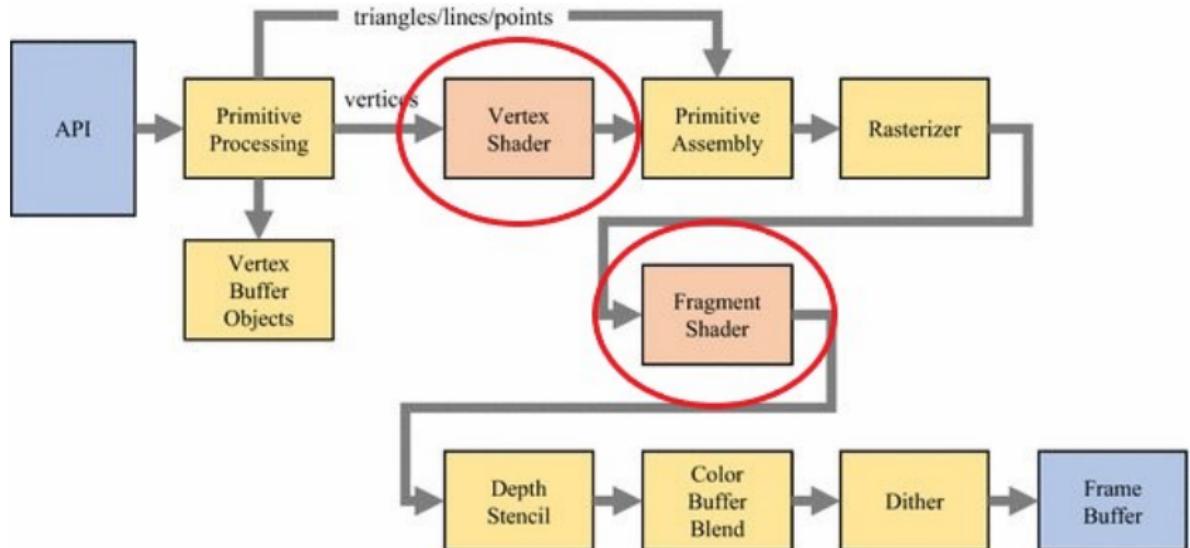
```
// Directive preprocessare
// Shader-e, date (coordonate, culori)
// Proceduri (inițializare, desen)
// Main
    // Inițializări GLUT
    // Generare fereastră
    // Apelare procedură inițializare
        // Creare VBO / VAO
        // Creare / Apelare shader-e
    // Apelare procedură desen
    // Stergere Shader-e, VBO / VAO
    // Apelare glutMainLoop
```

“Fixed versus programmable pipeline”



Sursa: [Baek and Kim, 2019](#)

“Fixed versus programmable pipeline”



Sursa: [Baek and Kim, 2019](#)

Primitive grafice

Mihai-Sorin Stupariu

Sem. I, 2022 - 2023

Spațiul culorilor

Vârfuri

Primitive grafice - generalități

Algoritmi de rasterizare

Preliminarii, notații

Algoritmii

Spațiul culorilor

- ▶ Culorile sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – **Cubul RGB**.

Spațiul culorilor

- ▶ Culoare sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – **Cubul RGB**.
- ▶ În OpenGL o culoare este indicată prin:

Spațiul culorilor

- ▶ Culoarele sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – **Cubul RGB**.
 - ▶ În OpenGL o culoare este indicată prin:
 - în OpenGL “vechi” folosind funcția
`glColor* ();` – “deprecated”
- Sufixul * poate indica:

Spațiul culorilor

- ▶ Culorile sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – **Cubul RGB**.
- ▶ În OpenGL o culoare este indicată prin:
 - în OpenGL “vechi” folosind funcția

```
glColor* ( );
```

– “deprecated”

Sufixul * poate indica:

- dimensiunea n a spațiului de culori în care lucrăm, $n = 3$ (RGB) sau $n = 4$ (RGBA); A=factorul “alpha”, legat de opacitate/transparență.

Spațiul culorilor

- ▶ Culorile sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – **Cubul RGB**.
- ▶ În OpenGL o culoare este indicată prin:
 - în OpenGL “vechi” folosind funcția

```
glColor* ( );      — “deprecated”
```

Sufixul * poate indica:

- dimensiunea n a spațiului de culori în care lucrăm, $n = 3$ (RGB) sau $n = 4$ (RGBA); A=factorul “alpha”, legat de opacitate/transparență.
- tipul de date utilizat, care poate fi:
 - i (integer) ($i \in \{0, 1, \dots, 255\}$)
 - f (float)
 - d (double) ($f, d \in [0.0, 1.0]$)

Spațiul culorilor

- ▶ Culoarele sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – **Cubul RGB**.
- ▶ În OpenGL o culoare este indicată prin:
 - în OpenGL “vechi” folosind funcția

```
glColor* ( );
```

– “deprecated”

Sufixul * poate indica:

- dimensiunea n a spațiului de culori în care lucrăm, $n = 3$ (RGB) sau $n = 4$ (RGBA); A=factorul “alpha”, legat de opacitate/transparență.
- tipul de date utilizat, care poate fi:
 - i (integer) ($i \in \{0, 1, \dots, 255\}$)
 - f (float)
 - d (double) ($f, d \in [0.0, 1.0]$)
- (optional) posibila formă vectorială, indicată prin sufixul v.

Spațiul culorilor

- ▶ Culoarele sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – **Cubul RGB**.
- ▶ În OpenGL o culoare este indicată prin:
 - în OpenGL “vechi” folosind funcția

```
glColor* ( );
```

– “deprecated”

Sufixul * poate indica:

- dimensiunea n a spațiului de culori în care lucrăm, $n = 3$ (RGB) sau $n = 4$ (RGBA); A=factorul “alpha”, legat de opacitate/transparență.
- tipul de date utilizat, care poate fi:
 - i (integer) ($i \in \{0, 1, \dots, 255\}$)
 - f (float)
 - d (double) ($f, d \in [0.0, 1.0]$)
- (optional) posibila formă vectorială, indicată prin sufixul v.
- în OpenGL “nou”: culorile sunt manevrate într-un mod asemănător vârfurilor (folosind VBO), odată cu acestea (atribute ale vârfurilor).

Spațiul culorilor

- ▶ Culoarele sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – **Cubul RGB**.
- ▶ În OpenGL o culoare este indicată prin:
 - în OpenGL “vechi” folosind funcția

`glColor* ()` – “deprecated”

Sufixul * poate indica:

- dimensiunea n a spațiului de culori în care lucrăm, $n = 3$ (RGB) sau $n = 4$ (RGBA); A=factorul “alpha”, legat de opacitate/transparență.
- tipul de date utilizat, care poate fi:
 - i (integer) ($i \in \{0, 1, \dots, 255\}$)
 - f (float)
 - d (double) ($f, d \in [0.0, 1.0]$)
- (optional) posibila formă vectorială, indicată prin sufixul v.
- în OpenGL “nou”: culorile sunt manevrate într-un mod asemănător vârfurilor (folosind VBO), odată cu acestea (atribute ale vârfurilor).
- elementul comun este faptul că, în ambele cazuri, culorile conțin informații legate de canalele R, G, B, precum și de canalul A (factorul α , legat de opacitate).

Vârfuri - funcții pentru indicare

Primitivele grafice sunt trasate cu ajutorul **vârfurilor** (*entități abstracte, a nu fi confundate cu punctele!*). În OpenGL un vârf este definit:

Vârfuri - funcții pentru indicare

Primitivele grafice sunt trasate cu ajutorul **vârfurilor** (*entități abstracte, a nu fi confundate cu punctele!*). În OpenGL un vârf este definit:

- în OpenGL “vechi” cu ajutorul funcției

```
glVertex* ( );
```

— “deprecated”

Vârfuri - funcții pentru indicare

Primitivele grafice sunt trasate cu ajutorul **vârfurilor** (*entități abstracte, a nu fi confundate cu punctele!*). În OpenGL un vârf este definit:

- în OpenGL “vechi” cu ajutorul funcției

```
glVertex* ( );
```

— “deprecated”

- în OpenGL “nou”:
vârfurile sunt stocate în matrice;
împachetate în VAO (Vertex Array Objects);
trimise plăcii grafice sub formă de VBO (Vertex Buffer Objects).

Caracteristici ale vârfurilor

- Unui vârf îi sunt asociate:

Caracteristici ale vârfurilor

- ▶ Unui vârf îi sunt asociate:
 - ▶ coordonate (fac parte din definiție),

Caracteristici ale vârfurilor

- ▶ Unui vârf îi sunt asociate:
 - ▶ coordonate (fac parte din definiție),
 - ▶ o culoare (v. secțiunea Culori...),

Caracteristici ale vârfurilor

- ▶ Unui vârf îi sunt asociate:
 - ▶ coordonate (fac parte din definiție),
 - ▶ o culoare (v. secțiunea Culori...),
 - ▶ o normală (legată de funcții de iluminare),

Caracteristici ale vârfurilor

- ▶ Unui vârf îi sunt asociate:
 - ▶ coordonate (fac parte din definiție),
 - ▶ o culoare (v. secțiunea Culori...),
 - ▶ o normală (legată de funcții de iluminare),
 - ▶ coordonate de texturare

Caracteristici ale vârfurilor

- ▶ Unui vârf îi sunt asociate:
 - ▶ coordonate (fac parte din definiție),
 - ▶ o culoare (v. secțiunea Culori...),
 - ▶ o normală (legată de funcții de iluminare),
 - ▶ coordonate de texturare
- ▶ În OpenGL “vechi”: pentru o anumită caracteristică, este considerată valoarea *currentă* a respectivei caracteristici. Altfel spus, ea trebuie indicată în codul sursă înaintea vârfului.

Functii pentru primitive

Vârfurile sunt utilizate pentru trasarea primitivelor grafice. Funcțiile folosite sunt diferite pentru cele două moduri de randare:

- în OpenGL “vechi”, o funcție de tipul `glVertex ()` poate fi apelată într-un cadru de tip

```
glBegin (*);  
    — “deprecated”  
glEnd;
```

(unde * reprezintă tipul de primitivă generat);

Funcții pentru primitive

Vârfurile sunt utilizate pentru trasarea primitivelor grafice. Funcțiile folosite sunt diferite pentru cele două moduri de randare:

- în OpenGL “vechi”, o funcție de tipul `glVertex ()` poate fi apelată într-un cadru de tip

```
glBegin (*);  
          – “deprecated”  
glEnd;
```

(unde * reprezintă tipul de primitivă generat);

- în OpenGL “nou” este utilizată o funcție de tipul

```
glDrawArrays (GLenum mode, GLint first, GLsizei count);
```

unde

`mode`: tipul primitivei;

`first`: primul vârf;

`count`: câte vârfuri se iau în considerare.

Tipuri de primitive

- ▶ Puncte: GL_POINTS

Tipuri de primitive

- ▶ Puncte: GL_POINTS
- ▶ Segmente de dreaptă: GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP

Tipuri de primitive

- ▶ Puncte: GL_POINTS
- ▶ Segmente de dreaptă: GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP
- ▶ Triunghiuri: GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN

Tipuri de primitive

- ▶ Puncte: GL_POINTS
- ▶ Segmente de dreaptă: GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP
- ▶ Triunghiuri: GL_TRIANGLES, GL_TRIANGLE_STRIP,
GL_TRIANGLE_FAN
- ▶ Dreptunghiuri: GL_QUADS, GL_QUAD_STRIP

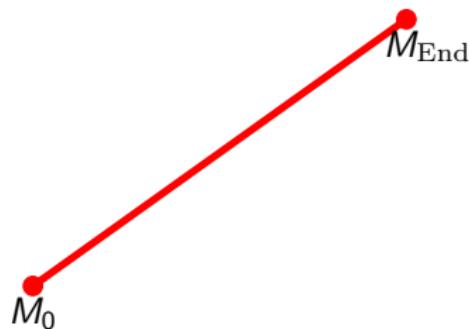
Tipuri de primitive

- ▶ Puncte: GL_POINTS
- ▶ Segmente de dreaptă: GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP
- ▶ Triunghiuri: GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN
- ▶ Dreptunghiuri: GL_QUADS, GL_QUAD_STRIP
- ▶ Poligoane (convexe!): GL_POLYGON

Motivație și problematizare

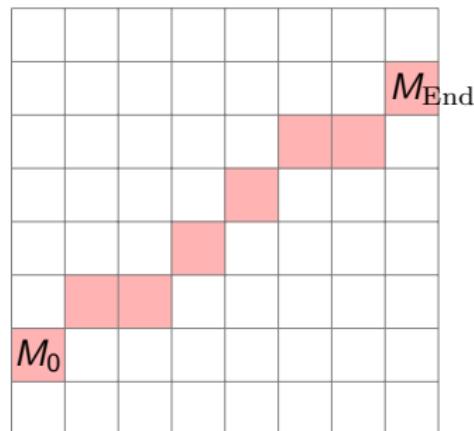
“Continuu”

“Grafica vectorială”



“Discret”

“Grafica rasterială”



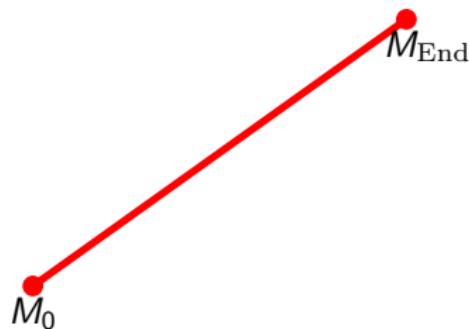
$$M_0 = (x_0, y_0), M_{\text{End}} = (x_{\text{End}}, y_{\text{End}}) \\ x_0, y_0, x_{\text{End}}, y_{\text{End}} \in \mathbb{R}$$

$$M_0 = (x_0, y_0), M_{\text{End}} = (x_{\text{End}}, y_{\text{End}}) \\ x_0, y_0, x_{\text{End}}, y_{\text{End}} \in \mathbb{N}(\mathbb{Z})$$

Motivație și problematizare

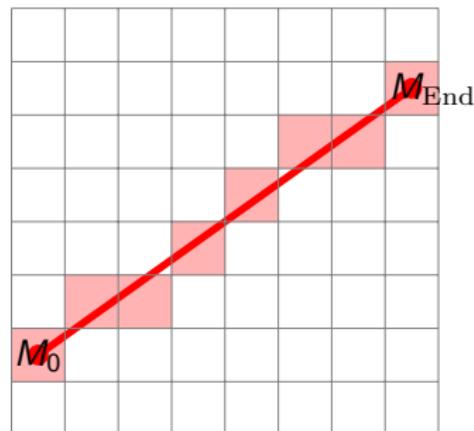
“Continuu”

“Grafica vectorială”



“Discret”

“Grafica rasterială”



$$M_0 = (x_0, y_0), M_{\text{End}} = (x_{\text{End}}, y_{\text{End}}) \\ x_0, y_0, x_{\text{End}}, y_{\text{End}} \in \mathbb{R}$$

$$M_0 = (x_0, y_0), M_{\text{End}} = (x_{\text{End}}, y_{\text{End}}) \\ x_0, y_0, x_{\text{End}}, y_{\text{End}} \in \mathbb{N}(\mathbb{Z})$$

Ce este un algoritm de rasterizare?

Un algoritm de rasterizare are ca efect reprezentarea grafică a unei primitive într-un sistem de reprezentare de tip grilă (monitor), care este format dintr-o structură discretă de pixeli. Pentru un segment, un algoritm de rasterizare are ca:

Input: Coodonatele $x_0, y_0, x_{\text{End}}, y_{\text{End}} \in \mathbb{N}$ ($\in \mathbb{Z}$) ale extremităților segmentului care urmează să fie reprezentat - altfel spus, pixelii inițial $M_0 = (x_0, y_0)$, respectiv final $M_{\text{End}} = (x_{\text{End}}, y_{\text{End}})$.

Ce este un algoritm de rasterizare?

Un algoritm de rasterizare are ca efect reprezentarea grafică a unei primitive într-un sistem de reprezentare de tip grilă (monitor), care este format dintr-o structură discretă de pixeli. Pentru un segment, un algoritm de rasterizare are ca:

Input: Coodonatele $x_0, y_0, x_{\text{End}}, y_{\text{End}} \in \mathbb{N} (\in \mathbb{Z})$ ale extremităților segmentului care urmează să fie reprezentat - altfel spus, pixelii inițial $M_0 = (x_0, y_0)$, respectiv final $M_{\text{End}} = (x_{\text{End}}, y_{\text{End}})$.

Output: Pixelii selectați pentru a trasa segmentul de la M_0 la M_{End}

Ipoteze / restricții

Raționamentele se pot adapta la restul situațiilor/cazurilor. Ipoteze (simplificatoare) făcute:

Ip. 1 Intuitiv: “deplasarea se face înspre dreapta/sus”

$$\begin{aligned}x_{\text{End}} > x_0, \quad y_{\text{End}} > y_0 &\Leftrightarrow \\&\Leftrightarrow \Delta x > 0, \quad \Delta y > 0.\end{aligned}$$

Ipoteze / restricții

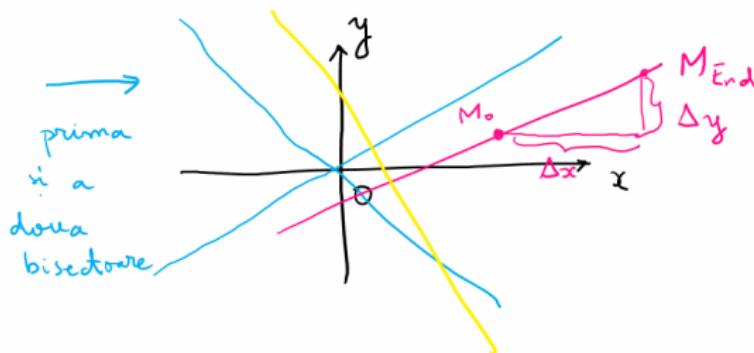
Raționamentele se pot adapta la restul situațiilor/cazurilor. Ipoteze (simplificatoare) făcute:

Ip. 1 Intuitiv: "deplasarea se face înspre dreapta/sus"

$$\begin{aligned}x_{\text{End}} > x_0, \quad y_{\text{End}} > y_0 &\Leftrightarrow \\&\Leftrightarrow \Delta x > 0, \quad \Delta y > 0.\end{aligned}$$

Ip. 2 Intuitiv: "dreapta trăsată prin origine și paralelă cu dreapta suport este situată sub prima bisectoare"

$$\Leftrightarrow \Delta x > \Delta y > 0.$$



Ecuația dreptei

Ecuația dreptei care unește punctele M_0 și M_{End}

$$y = mx + n. \quad (1)$$

Ecuăția dreptei

Ecuăția dreptei care unește punctele M_0 și M_{End}

$$y = mx + n. \quad (1)$$

Elemente importante: panta m și coeficientul n , care poate fi exprimat în funcție de pantă și de coordonatele lui M_0

$$m = \frac{\Delta y}{\Delta x} \text{ (panta)}$$

$$y_0 = mx_0 + n \Rightarrow n = y_0 - mx_0 \Rightarrow m = y_0 - \frac{\Delta y}{\Delta x} \cdot x_0$$

Ecuăția dreptei

Ecuăția dreptei care unește punctele M_0 și M_{End}

$$y = mx + n. \quad (1)$$

Elemente importante: panta m și coeficientul n , care poate fi exprimat în funcție de pantă și de coordonatele lui M_0

$$m = \frac{\Delta y}{\Delta x} \text{ (panta)}$$

$$y_0 = mx_0 + n \Rightarrow n = y_0 - mx_0 \Rightarrow m = y_0 - \frac{\Delta y}{\Delta x} \cdot x_0$$

Concluzie: În cazul continuu avem

$$y = mx + n \stackrel{NOT}{=} f(x).$$

Varianta 1 - înlocuire în ecuația dreptei

Algoritm

- Initializare $x_0, y_0 = f(x_0), m, n$

Varianta 1 - înlocuire în ecuația dreptei

Algoritm

- Initializare $x_0, y_0 = f(x_0), m, n$
- Pasul $k \rightarrow k + 1$ ($k \geq 0$)

Varianta 1 - înlocuire în ecuația dreptei

Algoritm

- Initializare $x_0, y_0 = f(x_0), m, n$
- Pasul $k \rightarrow k + 1 (k \geq 0)$
 $x_{k+1} \leftarrow x_k + 1$

Varianta 1 - înlocuire în ecuația dreptei

Algoritm

- Initializare $x_0, y_0 = f(x_0), m, n$

- Pasul $k \rightarrow k + 1$ ($k \geq 0$)

$$x_{k+1} \leftarrow x_k + 1$$

$$f(x_{k+1}) \leftarrow m \cdot x_{k+1} + n // \text{ formula (1)}$$

Varianta 1 - înlocuire în ecuația dreptei

Algoritm

- Initializare $x_0, y_0 = f(x_0), m, n$
- Pasul $k \rightarrow k + 1$ ($k \geq 0$)

$$x_{k+1} \leftarrow x_k + 1$$

$$f(x_{k+1}) \leftarrow m \cdot x_{k+1} + n \text{ // formula (1)}$$

$$y_{k+1} \leftarrow \text{round}(f(x_{k+1}))$$

Varianta 1 - înlocuire în ecuația dreptei

Algoritm

- Initializare $x_0, y_0 = f(x_0), m, n$
- Pasul $k \rightarrow k + 1$ ($k \geq 0$)

$$x_{k+1} \leftarrow x_k + 1$$
$$f(x_{k+1}) \leftarrow m \cdot x_{k+1} + n \quad // \text{ formula (1)}$$
$$y_{k+1} \leftarrow \text{round}(f(x_{k+1}))$$

Calcule...

$$\text{ptr} \cdot f(x_{k+1})$$

$$\underbrace{f(x_{k+1})}_{= m \cdot \underbrace{x_{k+1}}_{x_k + 1} + m} = m (x_k + 1) + m =$$

$$= m \cdot x_k + m + m =$$

$$= \underbrace{m \cdot x_k + m}_{f(x_k)} + m = \underbrace{f(x_k) + m}$$

Varianta 2 - algoritmul Digital Differential Analyzer (DDA)

Algoritm

- Initializare $x_0, y_0 = f(x_0), m$
- Pasul $k \rightarrow k + 1$ ($k \geq 0$)

$$x_{k+1} \leftarrow x_k + 1$$

$f(x_{k+1}) \leftarrow f(x_k) + m$ // formula (1), calculele anterioare

$$y_{k+1} \leftarrow \text{round}(f(x_{k+1}))$$

Exemplu

$$f(x_{k+1}) = f(x_k) + m$$

$$M_0 = (10, 20), M_{\text{End}} = (20, 28)$$

$$\Delta x = 10; \Delta y = 8; m = \frac{\Delta y}{\Delta x} = \frac{8}{10} = 0.8$$

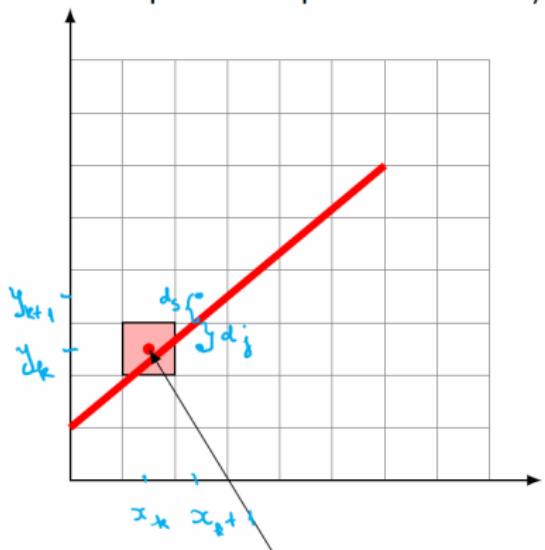
k	0	1	2	3	4	5	6	7	8	9	10
x_k	10	11	12	13	14	15	16	17	18	19	20
$f(x_k)$	20	20.8	21.6	22.4	23.2	24	24.8	25.6	26.4	27.2	28
y_k	20	21	22	22	23	24	25	26	26	27	28



 $(11, 21)$ $(12, 22)$ $(13, 22)$

Varianta 3 - algoritmul lui Bresenham. Idei fundamentale

Conceptul de “pixeli candidați”



Presupunem că pixelul (x_k, y_k) a fost selectat.
Care este pixelul următor?

Doi “candidați” pentru pixelul următor :

$$\begin{array}{l} \text{(ipoteza privind puncta dreptei)} \\ j : (x_k + 1, y_k) \\ s : (x_k + 1, y_k + 1) \end{array}$$

Idea: se calculează distanța de la punctul de pe dreaptă corespunzător abscisei $x_k + 1$ la centrele pixelilor candidați:

$$d_s = y_k + 1 - f(x_k + 1)$$

$$d_j = f(x_k + 1) - y_k$$

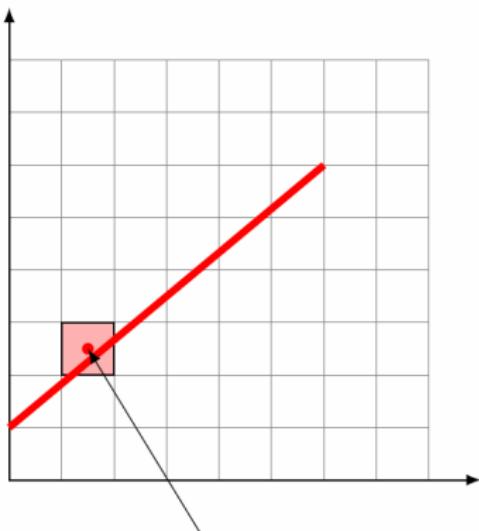
(sunt diferențe de ordonate !)

Algoritmul lui Bresenham. Observație

Varianta originală (1965) avea în vedere în special segmentele de dreaptă. Folosind conceptul de vector tangent, algoritmul poate fi adaptat și aplicat și în cazul altor curbe (v. art 1977)!



Algoritmul lui Bresenham. Factorul de decizie



$$d_s = \underbrace{y_{k+1}}_{\text{ordonata centrului pixelului de sus}} - \underbrace{f(x_{k+1})}_{\text{ordonata pt. de pe dreapta}}$$

$$d_j = f(x_{k+1}) - y_k$$

$$d_s = y_{k+1} - m \cdot x_{k+1} - m - m$$

$$d_j = m \cdot x_k + m + m - y_k$$

Presupunem că pixelul (x_k, y_k) a fost selectat.
Care este pixelul următor?

$d_s \geq d_j ?$

Algoritmul lui Bresenham. Semnul factorului de decizie

Ne interesează semnul numărului $d_j - d_s \in \mathbb{Q}$

$$d_j - d_s = 2m(x_k + 1) - 2y_k + 2m - 1 =$$

$$= \frac{2\Delta y(x_k + 1) - 2y_k \cdot \Delta x + 2m \Delta x - \Delta x}{\Delta x} \quad (\Delta x > 0)$$

$$\boxed{\gamma = 2\Delta y + 2\Delta x m - \Delta x}$$

Deci: semnul lui $d_j - d_s$ este semnul lui γ

$$\boxed{\gamma \stackrel{\text{def}}{=} 2\Delta y x_k - 2\Delta x y_k + \gamma}$$

$$\gamma < 0 \Leftrightarrow d_j - d_s < 0 \Leftrightarrow d_j < d_s \rightarrow \text{se alege } (x_k + 1, y_k)$$

$$\gamma \geq 0 \Leftrightarrow d_j - d_s \geq 0 \Leftrightarrow d_j \geq d_s \rightarrow \checkmark \quad (x_k + 1, y_k + 1)$$

Algoritmul lui Bresenham. Parametrul de decizie

Semnul lui $d_j - d_s$ este semnul **parametrului de decizie**

$$p_k \stackrel{DEF}{=} 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + \gamma$$

Dacă $p_k < 0 \Rightarrow d_j < d_s \Rightarrow$ se alege pixelul $(x_k + 1, y_k)$

Dacă $p_k \geq 0 \Rightarrow d_j \geq d_s \Rightarrow$ se alege pixelul $(x_k + 1, y_k + 1)$

Algoritmul lui Bresenham. Parametrul de decizie - rescriere

Evaluăm $p_{k+1} - p_k =$

$$= 2 \Delta y \cdot x_{k+1} - 2 \Delta x \cdot y_{k+1} + \cancel{2 \Delta y \cdot x_k} - \cancel{2 \Delta x \cdot y_k}$$

$$\parallel x_{k+1} = x_k + 1$$

$$2 \Delta y$$

$$y_{k+1} - y_k = \begin{cases} 0, & p_k < 0 \\ 1, & p_k \geq 0 \end{cases}$$

$$= \begin{cases} 2 \Delta y, & \text{dacă } p_k < 0 \quad (\text{adică } y_{k+1} = y_k) \\ 2 \Delta y - 2 \Delta x, & \text{dacă } p_k \geq 0 \quad (\text{adică } y_{k+1} = y_k + 1) \end{cases}$$

Algoritmul lui Bresenham. Parametrul de decizie - rescriere

Evaluăm

$$p_0 = 2\Delta y \cdot x_0 - 2\Delta x \cdot y_0 + \gamma =$$

$$= 2\Delta y \cdot x_0 - 2\Delta x \cdot (m \cdot x_0 + n) + 2\Delta y + 2\Delta x \cdot n - \Delta x \Rightarrow$$

$$p_0 = 2\Delta y - \Delta x.$$

Algoritmul lui Bresenham

Algoritm

- Calcul preliminar $\Delta x, \Delta y, 2\Delta y, 2\Delta y - 2\Delta x, p_0 = 2\Delta y - \Delta x$

Algoritmul lui Bresenham

Algoritm

- Calcul preliminar $\Delta x, \Delta y, 2\Delta y, 2\Delta y - 2\Delta x, p_0 = 2\Delta y - \Delta x$
- Pasul $k \rightarrow k + 1$ ($k \geq 0$). // Pp. că avem $x_k, y_k, p_k \in \mathbb{Z}$

Algoritmul lui Bresenham

Algoritm

- Calcul preliminar $\Delta x, \Delta y, 2\Delta y, 2\Delta y - 2\Delta x, p_0 = 2\Delta y - \Delta x$
- Pasul $k \rightarrow k + 1$ ($k \geq 0$). // Pp. că avem $x_k, y_k, p_k \in \mathbb{Z}$
Dacă $p_k < 0$ (''stagnare pe orizontală'')

Algoritmul lui Bresenham

Algoritm

- Calcul preliminar $\Delta x, \Delta y, 2\Delta y, 2\Delta y - 2\Delta x, p_0 = 2\Delta y - \Delta x$
- Pasul $k \rightarrow k + 1$ ($k \geq 0$). // Pp. că avem $x_k, y_k, p_k \in \mathbb{Z}$
Dacă $p_k < 0$ (''stagnare pe orizontală'')

$$x_{k+1} \leftarrow x_k + 1$$

$$y_{k+1} \leftarrow y_k$$

$$p_{k+1} \leftarrow p_k + 2\Delta y$$

Algoritmul lui Bresenham

Algoritm

- Calcul preliminar $\Delta x, \Delta y, 2\Delta y, 2\Delta y - 2\Delta x, p_0 = 2\Delta y - \Delta x$
- Pasul $k \rightarrow k + 1$ ($k \geq 0$). // Pp. că avem $x_k, y_k, p_k \in \mathbb{Z}$
Dacă $p_k < 0$ ("stagnare pe orizontală")

$$x_{k+1} \leftarrow x_k + 1$$

$$y_{k+1} \leftarrow y_k$$

$$p_{k+1} \leftarrow p_k + 2\Delta y$$

Dacă $p_k \geq 0$ ("în sus")

Algoritmul lui Bresenham

Algoritm

- Calcul preliminar $\Delta x, \Delta y, 2\Delta y, 2\Delta y - 2\Delta x, p_0 = 2\Delta y - \Delta x$
- Pasul $k \rightarrow k + 1$ ($k \geq 0$). // Pp. că avem $x_k, y_k, p_k \in \mathbb{Z}$
Dacă $p_k < 0$ (''stagnare pe orizontală'')

$$x_{k+1} \leftarrow x_k + 1$$

$$y_{k+1} \leftarrow y_k$$

$$p_{k+1} \leftarrow p_k + 2\Delta y$$

- Dacă $p_k \geq 0$ (''în sus'')

$$x_{k+1} \leftarrow x_k + 1$$

$$y_{k+1} \leftarrow y_k + 1$$

$$p_{k+1} \leftarrow p_k + 2\Delta y - 2\Delta x$$

Algoritmul lui Bresenham. Exemplu

$$M_0 = (10, 20), M_{\text{End}} = (20, 28)$$

$$x_0 = 10, y_0 = 20, x_{\text{End}} = 20, y_{\text{End}} = 28.$$

$$\Delta x = 10, \Delta y = 8$$

$$p_0 = 2\Delta y - \Delta x = 6$$

$$2\Delta y = 16$$

$$2\Delta y - 2\Delta x = -4$$

k	0	1	2	3	4	5	6	7	8	9	10
x_k	10	11	12	13	14	15	16	17	18	19	20
y_k	20	21	22	22	23	24	25	26	26	27	28
p_k	6	2	-2	14	10	6	2	-2	14	10	6

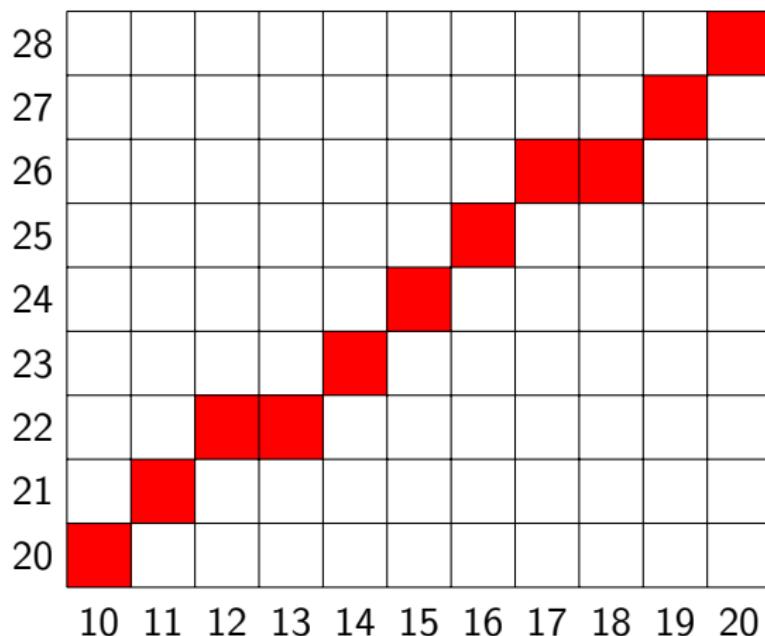


Figura: Algoritmul DDA / algoritmul lui Bresenham. Pixelii selectați pentru a uni punctele $(10, 20)$ și $(20, 28)$ sunt colorați cu roșu.

Primitive grafice. Fața și spatele unui poligon convex

Mihai-Sorin Stupariu

Sem. I, 2022 - 2023

Condiții pentru poligoane

Vector normal. Față și spatele unui poligon convex

Exemple

Linii poligonale închise cu autointersecții

Motivație

- ▶ Codurile sursă 02_02_fata_spatele_polig.cpp, 02_03_poligoane3d_old.cpp, 02_04_poligoane3d_new.cpp.

Motivație

- ▶ Codurile sursă 02_02_fata_spatele_polig.cpp, 02_03_poligoane3d_old.cpp, 02_04_poligoane3d_new.cpp.
- ▶ Ce proprietăți geometrice sunt / NU sunt implementate în OpenGL?

Motivație

- ▶ Codurile sursă 02_02_fata_spatele_polig.cpp, 02_03_poligoane3d_old.cpp, 02_04_poligoane3d_new.cpp.
- ▶ Ce proprietăți geometrice sunt / NU sunt implementate în OpenGL?
 - ▶ **NU:** reguli pentru aplicarea reguli pentru aplicarea funcției GL_POLYGON
 - se presupune că vârfurile determină un poligon convex

Motivație

- ▶ Codurile sursă 02_02_fata_spatele_polig.cpp, 02_03_poligoane3d_old.cpp, 02_04_poligoane3d_new.cpp.
- ▶ Ce proprietăți geometrice sunt / NU sunt implementate în OpenGL?
 - ▶ **NU:** reguli pentru aplicarea reguli pentru aplicarea funcției GL_POLYGON
 - se presupune că vârfurile determină un poligon convex
 - ▶ **DA:** față și spatele unui poligon convex

Reguli pentru aplicarea opțiunii GL_POLYGON

Se presupune că opțiunea `GL_POLYGON` este utilizată pentru un sir de vârfuri P_1, P_2, \dots, P_N , distincte două câte două. Reguli referitoare la vârfurile indicate, pentru ca poligonul să poată fi desenat:

1. Punctele trebuie să fie coplanare, dar nu coliniare.

Reguli pentru aplicarea opțiunii GL_POLYGON

Se presupune că opțiunea `GL_POLYGON` este utilizată pentru un sir de vârfuri P_1, P_2, \dots, P_N , distincte două câte două. Reguli referitoare la vârfurile indicate, pentru ca poligonul să poată fi desenat:

1. Punctele trebuie să fie coplanare, dar nu coliniare.
2. Vârfurile trebuie indicate în ordinea corectă, astfel încât linia poligonală să nu aibă autointersecții.

Reguli pentru aplicarea opțiunii GL_POLYGON

Se presupune că opțiunea `GL_POLYGON` este utilizată pentru un sir de vârfuri P_1, P_2, \dots, P_N , distincte două câte două. Reguli referitoare la vârfurile indicate, pentru ca poligonul să poată fi desenat:

1. Punctele trebuie să fie coplanare, dar nu coliniare.
2. Vârfurile trebuie indicate în ordinea corectă, astfel încât linia poligonală să nu aibă autointersecții.
3. Poligonul trebuie să fie convex.

1. Coplanaritatea

De verificat: condiția de coplanaritate

$$\text{rang} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ x_{P_1} & x_{P_2} & x_{P_3} & \dots & x_{P_N} \\ y_{P_1} & y_{P_2} & y_{P_3} & \dots & y_{P_N} \\ z_{P_1} & z_{P_2} & z_{P_3} & \dots & z_{P_N} \end{pmatrix} = 3 \quad (1)$$

sau faptul că

$$\dim_{\mathbb{R}} \langle \overrightarrow{P_1P_2}, \overrightarrow{P_1P_3}, \dots, \overrightarrow{P_1P_N} \rangle = 2. \quad (2)$$

Fapt: O condiție alternativă este coliniaritatea vectorilor $\overrightarrow{P_1P_2} \times \overrightarrow{P_2P_3}$, $\overrightarrow{P_2P_3} \times \overrightarrow{P_3P_4}$, \dots , $\overrightarrow{P_{N-1}P_N} \times \overrightarrow{P_NP_1}$, $\overrightarrow{P_NP_1} \times \overrightarrow{P_1P_2}$. Altfel spus: punctele P_1, P_2, \dots, P_N sunt coplanare dacă și numai dacă vectorii $\overrightarrow{P_{i-1}P_i} \times \overrightarrow{P_iP_{i+1}}$ ($i = 1, \dots, N$, cu convenții modulo N) sunt coliniari.

Exemplu

Punctele $P_1 = (7, 1, 1)$, $P_2 = (-3, 3, 9)$, $P_3 = (1, -1, 9)$, $P_4 = (8, -4, 5)$ sunt coplanare.

$$\overrightarrow{P_1 P_2} \times \overrightarrow{P_2 P_3} = (32, 32, 32)$$

$$\overrightarrow{P_2 P_3} \times \overrightarrow{P_3 P_4} = (16, 16, 16)$$

$$\overrightarrow{P_3 P_4} \times \overrightarrow{P_4 P_1} = (32, 32, 32)$$

$$\overrightarrow{P_4 P_1} \times \overrightarrow{P_1 P_2} = (48, 48, 48)$$

vectorii sunt proporționali,
deci coliniari
 \iff punctele sunt coplanare

$$\overrightarrow{P_1 P_2} = \overrightarrow{P_2} - \overrightarrow{P_1} = (-3, 3, 9) - (7, 1, 1) = (-10, 2, 8)$$

$$\overrightarrow{P_2 P_3} = \overrightarrow{P_3} - \overrightarrow{P_2} = (1, -1, 9) - (-3, 3, 9) = (4, -4, 0)$$

$$\begin{aligned} \overrightarrow{P_1 P_2} \times \overrightarrow{P_2 P_3} &= \begin{vmatrix} -10 & 4 & e_1 \\ 2 & -4 & e_2 \\ 8 & 0 & e_3 \end{vmatrix} \stackrel{\text{dezv.}}{\overline{\overline{\text{ultima}}} \atop \text{coloană}} \quad | \begin{matrix} 2 & -4 \\ 8 & 0 \end{matrix} | e_1 - \begin{vmatrix} -10 & 4 \\ 8 & 0 \end{vmatrix} e_2 + \\ &+ \begin{vmatrix} -10 & 4 \\ 2 & -4 \end{vmatrix} e_3 = \\ &= 32e_1 + 32e_2 + 32e_3 = (32, 32, 32) \end{aligned}$$

Exemplu

Punctele $P_1 = (7, 1, 1)$, $P_2 = (-3, 3, 9)$, $P_3 = (1, -1, 9)$, $P_4 = (11, -3, 1)$ sunt coplanare.

$$\overrightarrow{P_1P_2} \times \overrightarrow{P_2P_3} = (32, 32, 32)$$

$$\overrightarrow{P_2P_3} \times \overrightarrow{P_3P_4} = (16, 16, 16)$$

$$\overrightarrow{P_3P_4} \times \overrightarrow{P_4P_1} = (32, 32, 32)$$

$$\overrightarrow{P_4P_1} \times \overrightarrow{P_1P_2} = (48, 48, 48)$$

2. Linie poligonală fără autointersecții

De verificat: intersecții de segmente.

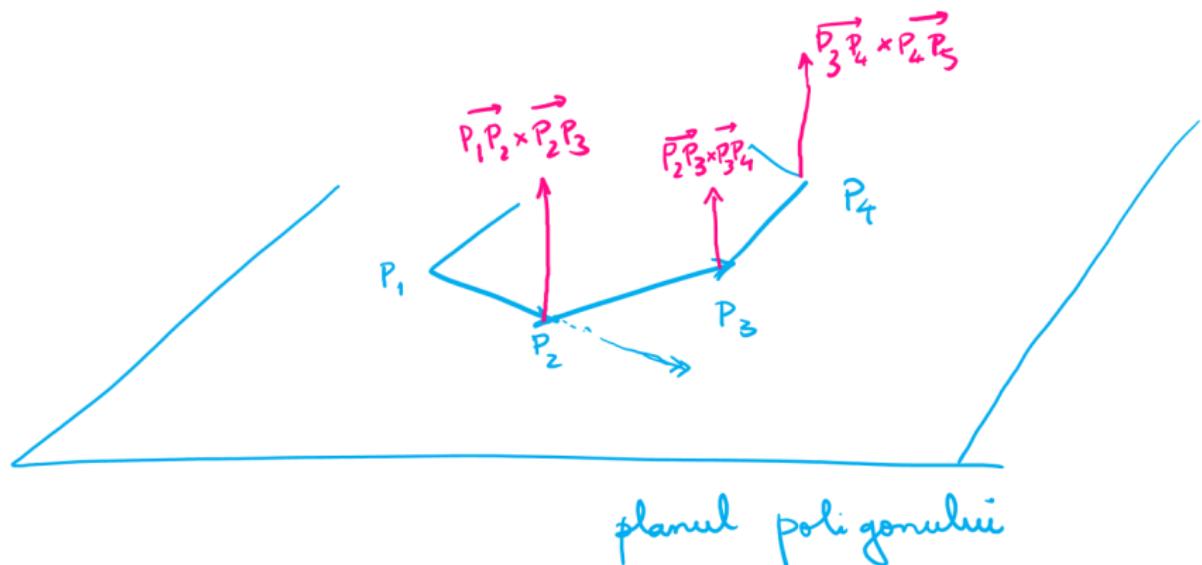
Varianta 1 Segmentele $[AB]$ și $[CD]$ se intersectează $\Leftrightarrow A$ și B sunt de o parte și de alta a dreptei CD și C și D sunt de o parte și de alta a dreptei AB . Două puncte M și N sunt de o parte și de alta a dreptei d de ecuație $f(x, y) = \alpha x + \beta y + \gamma = 0 \Leftrightarrow f(M) \cdot f(N) < 0$.

Varianta 2 Se folosește reprezentarea segmentelor cu ajutorul combinațiilor afine. Segmentele $[AB]$ și $[CD]$ se intersectează \Leftrightarrow

$$\exists s_0, t_0 \in [0, 1] \quad \text{a.î.} \quad (1 - t_0)A + t_0B = (1 - s_0)C + s_0D.$$

Această variantă poate fi aplicată și în context 3D.

3. Convexitatea poligonului - figura



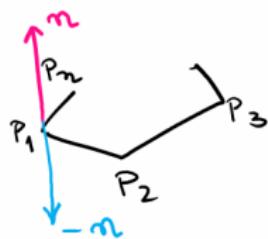
În cazul unui poligon convex, vectorii $\vec{P_1P_2} \times \vec{P_2P_3}, \vec{P_2P_3} \times \vec{P_3P_4}, \dots$
au același sens (și reciproc!)

Convexitatea poligonului - observație

Obs. Fie P_1, P_2, \dots, P_n un poligon convex.

(i) Parcurgerea $P_1, P_2, \dots, P_n \rightsquigarrow$ vector normal n

(ii) Parcurgerea $P_n, P_{n-1}, \dots, P_1 \rightsquigarrow -n$



Justificare:

Prin (i) calculăm $\overrightarrow{P_1P_2} \times \overrightarrow{P_2P_3}$, apoi normalizare

Prin (ii) calculăm $\overrightarrow{P_3P_2} \times \overrightarrow{P_2P_1} = (-\overrightarrow{P_2P_3}) \times (-\overrightarrow{P_1P_2})$

$$= \overrightarrow{P_2P_3} \times \overrightarrow{P_1P_2} = - \overrightarrow{P_1P_2} \times \overrightarrow{P_2P_3}$$

3. Convexitatea poligonului

De verificat: convexitatea (folosind produse vectoriale).

Observație. (i) Fie (P_1, P_2, \dots, P_N) un poligon (sensul de parcursere este important!). Poligonul \mathcal{P} este convex dacă și numai dacă pentru orice trei vârfuri consecutive P_{i-1}, P_i, P_{i+1} (modulo N) ale poligonului sensul vectorul $\overrightarrow{P_{i-1}P_i} \times \overrightarrow{P_iP_{i+1}}$ este independent de i .

- (ii) Vectorii menționați au toți aceeași direcție (perpendiculari pe planul poligonului), deoarece punctele sunt coplanare (vezi condiția 1).
- (iii) Pentru un poligon convex, vectorul

$$\mathbf{n} = \frac{\overrightarrow{P_{i-1}P_i} \times \overrightarrow{P_iP_{i+1}}}{\| \overrightarrow{P_{i-1}P_i} \times \overrightarrow{P_iP_{i+1}} \|}$$

este independent de i .

Exemplu. Punctele $P_1 = (7, 1, 1)$, $P_2 = (-3, 3, 9)$, $P_3 = (1, -1, 9)$, $P_4 = (11, -3, 1)$ determină un poligon convex.

Definiție - vector normal

Lemă. Pentru un poligon convex, vectorul

$$\mathbf{n} = \frac{\overrightarrow{P_{i-1}P_i} \times \overrightarrow{P_iP_{i+1}}}{\| \overrightarrow{P_{i-1}P_i} \times \overrightarrow{P_iP_{i+1}} \|}$$

este independent de i .

Definiție. Fie (P_1, P_2, \dots, P_N) un poligon convex. Se alege $i = 1, \dots, n$.

Vectorul

$$\mathbf{n} = \frac{\overrightarrow{P_{i-1}P_i} \times \overrightarrow{P_iP_{i+1}}}{\| \overrightarrow{P_{i-1}P_i} \times \overrightarrow{P_iP_{i+1}} \|}$$

se numește **vector normal (normală)** la planul poligonului / poligonul (P_1, P_2, \dots, P_N) .

Modalitate de calcul (I)

1. Se aleg trei vârfuri consecutive, de exemplu P_1, P_2, P_3 , având coordonatele $P_1 = (x_{P_1}, y_{P_1}, z_{P_1})$, $A_2 = (x_{P_2}, y_{P_2}, z_{P_2})$, respectiv $P_3 = (x_{P_3}, y_{P_3}, z_{P_3})$.

Modalitate de calcul (I)

- Se aleg trei vârfuri consecutive, de exemplu P_1, P_2, P_3 , având coordonatele $P_1 = (x_{P_1}, y_{P_1}, z_{P_1})$, $A_2 = (x_{P_2}, y_{P_2}, z_{P_2})$, respectiv $P_3 = (x_{P_3}, y_{P_3}, z_{P_3})$.
- Se scrie ecuația planului determinat de cele trei puncte sub forma

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

unde coeficienții A, B, C și D sunt date de formulele

$$A = \begin{vmatrix} y_{P_1} & z_{P_1} & 1 \\ y_{P_2} & z_{P_2} & 1 \\ y_{P_3} & z_{P_3} & 1 \end{vmatrix}, \quad B = - \begin{vmatrix} x_{P_1} & z_{P_1} & 1 \\ x_{P_2} & z_{P_2} & 1 \\ x_{P_3} & z_{P_3} & 1 \end{vmatrix} = \begin{vmatrix} x_{P_1} & 1 & z_{P_1} \\ x_{P_2} & 1 & z_{P_2} \\ x_{P_3} & 1 & z_{P_3} \end{vmatrix},$$

$$C = \begin{vmatrix} x_{P_1} & y_{P_1} & 1 \\ x_{P_2} & y_{P_2} & 1 \\ x_{P_3} & y_{P_3} & 1 \end{vmatrix}, \quad D = - \begin{vmatrix} x_{P_1} & y_{P_1} & z_{P_1} \\ x_{P_2} & y_{P_2} & z_{P_2} \\ x_{P_3} & y_{P_3} & z_{P_3} \end{vmatrix},$$

fiind deduși din condiția de coliniaritate

$$\begin{vmatrix} x & y & z & 1 \\ x_{P_1} & y_{P_1} & z_{P_1} & 1 \\ x_{P_2} & y_{P_2} & z_{P_2} & 1 \\ x_{P_3} & y_{P_3} & z_{P_3} & 1 \end{vmatrix} = 0.$$

Pe scurt: se dezvoltă după linia I determinantul de mai sus.

Modalitate de calcul (II)

3 Are loc relația

$$\overrightarrow{P_1P_2} \times \overrightarrow{P_2P_3} = (A, B, C).$$

Modalitate de calcul (II)

3 Are loc relația

$$\overrightarrow{P_1P_2} \times \overrightarrow{P_2P_3} = (A, B, C).$$

4 În final:

$$n = \frac{1}{\sqrt{A^2 + B^2 + C^2}}(A, B, C).$$

Conceptul de față / spate al unui poligon convex

Definiție. Pentru un punct $M = (x, y, z) \in \mathbb{R}^3$ notăm

$$\pi(M) = \pi(x, y, z) = Ax + By + Cz + D.$$

Noțiunile de **față/spate** a planului poligonului (și, implicit, a poligonului convex fixat) sunt definite astfel:

- $M = (x, y, z)$ se află **în fața planului (poligonului)**
 $\Leftrightarrow \pi(M) = \pi(x, y, z) > 0;$

Conceptul de față / spate al unui poligon convex

Definiție. Pentru un punct $M = (x, y, z) \in \mathbb{R}^3$ notăm

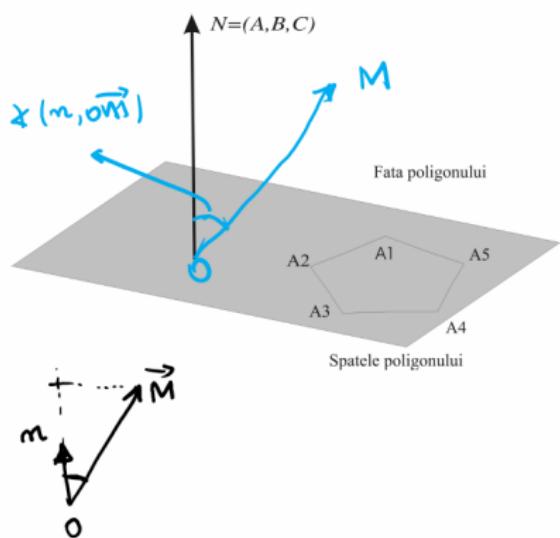
$$\pi(M) = \pi(x, y, z) = Ax + By + Cz + D.$$

Noțiunile de **față/spate** a planului poligonului (și, implicit, a poligonului convex fixat) sunt definite astfel:

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

Interpretare - “normala indică fața poligonului”

Presupunem că $D = 0$, adică planul trece prin originea $O = (0, 0, 0)$.



proiecția lui M pe dreapta det. de n are același sens cu $n \Leftrightarrow n \text{ și } \overrightarrow{OM}$ de aceeași parte a planului

$$\begin{aligned}
 M \text{ este în fața poligonului} &\Leftrightarrow \\
 &\Leftrightarrow A_x + B_y + C_z > 0 \\
 &\quad (\text{unde } M = (x, y, z)) \\
 &\Rightarrow \text{produs scalar între vectorii} \\
 &\quad (A, B, C) \text{ și } (x, y, z) \\
 &\Leftrightarrow \underbrace{\langle (A, B, C), (x, y, z) \rangle}_{\text{produs scalar}} > 0 \\
 &\Leftrightarrow \langle n, \overrightarrow{OM} \rangle > 0 \\
 &\quad (\overrightarrow{OM} = (x, y, z))
 \end{aligned}$$

$$\begin{aligned}
 &\Leftrightarrow \cos(\angle(n, \overrightarrow{OM})) > 0 \Leftrightarrow \\
 &\Leftrightarrow \angle(n, \overrightarrow{OM}) < 90^\circ \Leftrightarrow
 \end{aligned}$$

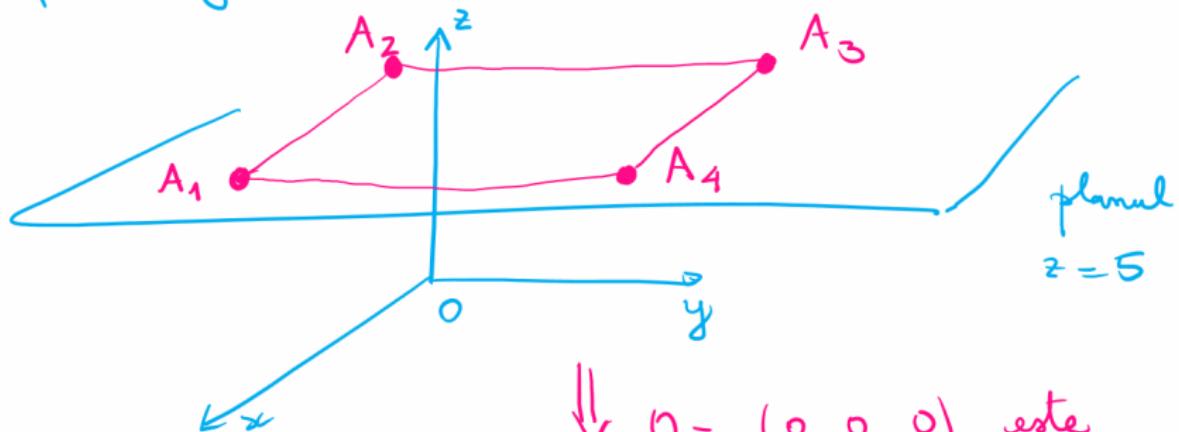
Intrepretare - sinteză

- ▶ Presupunem că $D = 0$, deci planul trece prin origine, iar ecuația sa este $\pi(x, y, z) = Ax + By + Cz = 0$.
- ▶ Considerând vectorul $n = (A, B, C)$ care direcționează normala la plan, avem $\pi(A, B, C) > 0$, deci vectorul n indică partea din față a poligonului (planului).
- ▶ În general, un vector (x, y, z) este orientat înspre partea din față a planului dacă $\pi(x, y, z) > 0$, i.e. $\langle (x, y, z), n, \rangle > 0$, ceea ce înseamnă că proiecția vectorului (x, y, z) pe N este la fel orientată ca și n .
- ▶ Prin translație, aceste rezultate pot fi extinse pentru un plan arbitrar. Mai mult, presupunând că parcurgem poligonul (A_1, A_2, \dots, A_n) în sens trigonometric și că rotim un burghiu drept în sensul indicat de această parcurgere, acesta se va deplasa în sensul indicat de vectorul N , deci înspre față poligonului (vezi figura).
- ▶ **Intuitiv + concluzie:** din față un poligon este văzut ca fiind parcurs în sens trigonometric, iar din spate un poligon este văzut ca fiind parcurs în sens orar.

Exemplul 1. Cod sursă 02_03_poligoane3d_old.cpp

$$A_1 = (5, -5, 5), A_2 = (-5, -5, 5), A_3 = (-5, 5, 5), A_4 = (5, 5, 5)$$

Explicație geometrică:

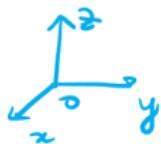
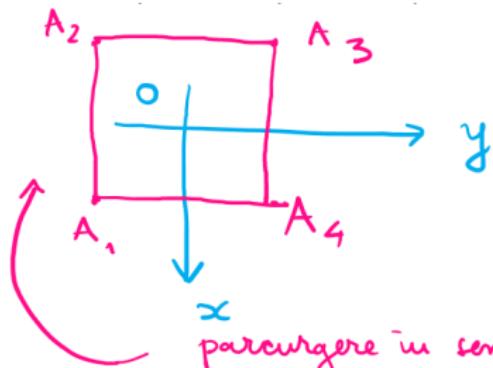


↓ $O = (0, 0, 0)$ este
în fața poligonului

Exemplul 1. Cod sursă 02_03_poligoane3d_old.cpp

$$A_1 = (5, -5, 5), A_2 = (-5, -5, 5), A_3 = (-5, 5, 5), A_4 = (5, 5, 5)$$

"văzut de sus"

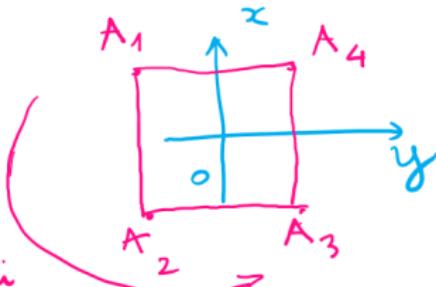


parcurgere în sens orar

"văzut de jos"

Concluzie

$(0, 0, 0)$ este în
fata poligonului



parcurgere
în sens
trigonometric

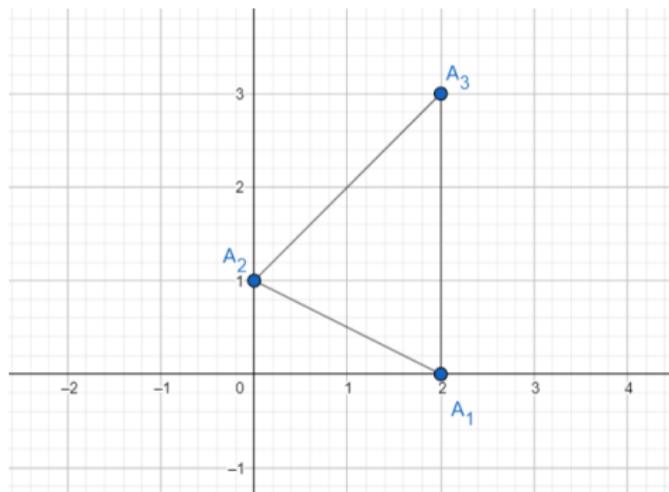
Exemplul 1. Cod sursă 02_03_poligoane3d_old.cpp

$$A_1 = (5, -5, 5), A_2 = (-5, -5, 5), A_3 = (-5, 5, 5), A_4 = (5, 5, 5)$$

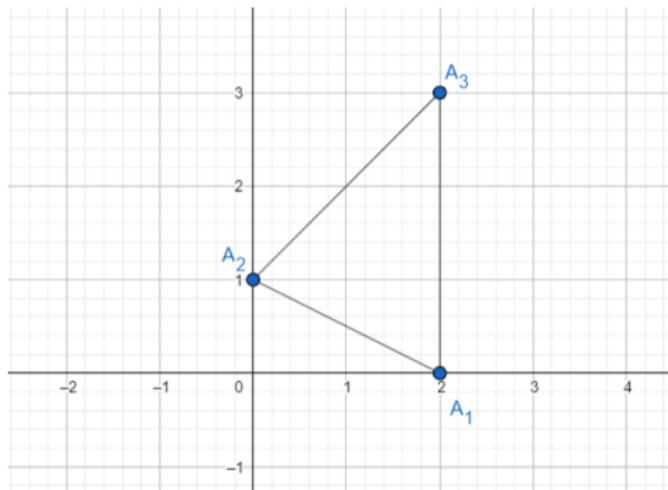
Exemplul 1. Cod sursă 02_03_poligoane3d_old.cpp

$$A_1 = (5, -5, 5), A_2 = (-5, -5, 5), A_3 = (-5, 5, 5), A_4 = (5, 5, 5)$$

De reținut (și de aplicat la cerința 2, L2)!

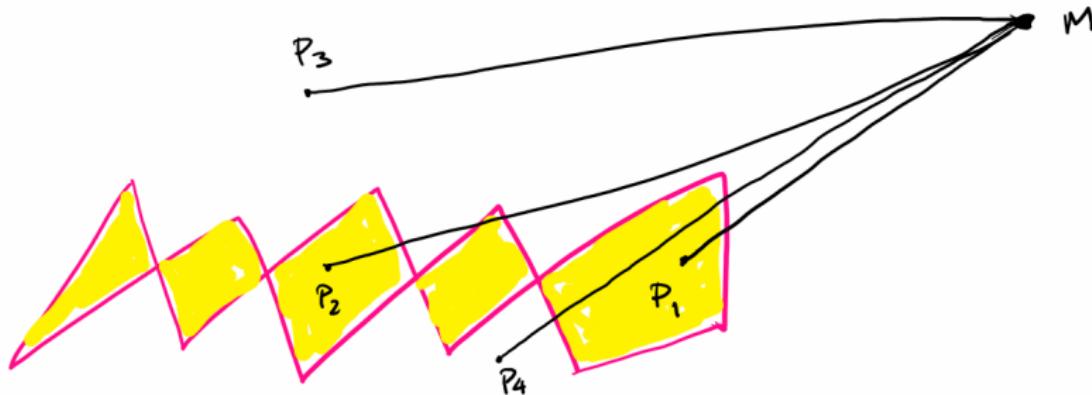


De reținut (și de aplicat la cerința 2, L2)!



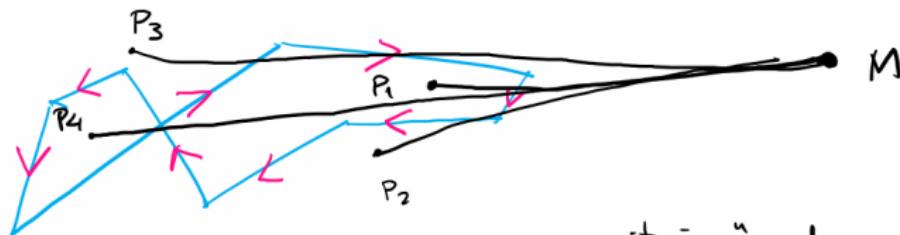
Dacă în codul sursă vârfurile sunt indicate în ordinea A_1, A_3, A_2 , atunci triunghiul $A_1A_2A_3$ este “văzut din față” și se aplică regulile pentru GL_FRONT, iar dacă sunt indicate în ordinea A_1, A_2, A_3 , atunci triunghiul este “văzut din spate” și se aplică regulile pentru GL_BACK. Ordinea de parcursare face referire la modul implicit (GL_CCW).

Linii poligonale închise cu autointersecții: interior/exterior

Regula par-impar (*odd-even rule*)

- se alege un punct M "departe" în afara poligonului
 - pentru un punct P nerăstăruit pe laturi: paritatea nr. de intersecții dintre segmentul $[PM]$ și linia poligonată \rightarrow decizie
- | | | | |
|---------------|-----------------------|---------------|----------|
| <u>nu par</u> | <u>de intersecții</u> | \rightarrow | exterior |
| <u>impar</u> | <u>—</u> | \rightarrow | interior |

Linii poligonale închise cu autointersecții: interior/exterior

Regula indexului nenul (*non-zero winding number rule*)

ne uităm "în dreapta": +

Convenție: ne deplasăm de la M la P : - → "în stânga": -Notăm: n_+ : nr. de intersecții cu semnul "+"
 n_- : - , - , - , - "Indexul unui punct P este $i_p = n_+ - n_-$ Ptr. regula în drepturi : un punct este exterior dacă indexul este 0

- →

interior →

→ ≠ 0

Linii poligonale închise cu autointersecții: interior/exterior

Observație

Legătura dintre cele două reguli

Obs : $(n_+ + n_-)$ = numărul total de intersecții,
iar paritatea acestuia ne dă regula par/impar.

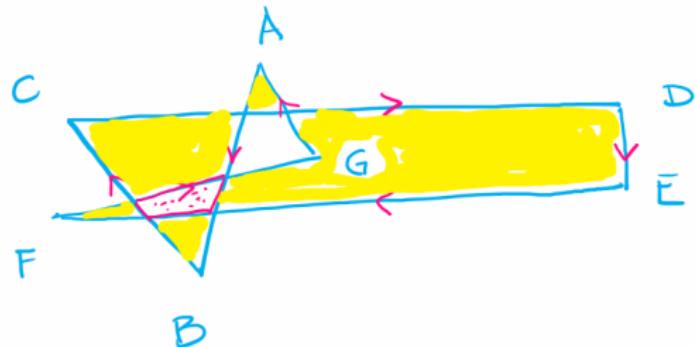
- P exterior pînă la regula înde xului $\Rightarrow n_+ = n_-$
 $\Rightarrow (n_+ + n_-)$ este par \rightarrow P exterior pînă la
par / impar



NU este neapărat
adevărat

Linii poligonale închise cu autointersecții: interior/exterior

Exemplu



exterior în
ptr · p/i în
ptr .index



interior în
ptr.index
în ptr.p/i



exterior
ptr.

par_impar
interior ptr
index

Transformări

Mihai-Sorin Stupariu

Sem. I, 2022 - 2023

Motivație

- ▶ Cum desenăm primitive atunci când vârfurile au coordonatele în afara intervalului $[-1, 1] \times [-1, 1]$?

Motivație

- ▶ Cum desenăm primitive atunci când vârfurile au coordonatele în afara intervalului $[-1, 1] \times [-1, 1]$?
- ▶ Cum procedăm pentru a “deplasa” primitivele în scenă?

În OpenGL "vechi" - codul sursă 03_01_animatie_old.cpp

```
void init(void)
{
    glClearColor(1.0, 1.0, 1.0, 0.0); // Coordonata x intre 0 si 800, coordonata
    gluOrtho2D(0.0, 800.0, 0.0, 600.0); // y intre 0 si 600 ("dreptunghi decupat")
}
```

// dreptunghiul rosu

```
glTranslated(i, 200.0, 0.0);
glPushMatrix();
```



Functii specifice pentru deplasare
(translatie, rotatie) - atentie la ordinea in
care sunt aplicate!

```
glRotated(j, 0.0, 0.0, 1.0);
glColor3f(1.0, 0.0, 0.0);
glRecti(-5, 30, 5, 40);
```

În OpenGL "nou" - codul sursă 03_02_animatie_new.cpp

```
resizeMatrix = glm::ortho(-width, width, -height, height); // scalam, "a
matrTransl = glm::translate(glm::mat4(1.0f), glm::vec3(i, 0.0, 0.0)); //
matrDepl = glm::translate(glm::mat4(1.0f), glm::vec3(0, 80.0, 0.0)); //
matrScale1 = glm::scale(glm::mat4(1.0f), glm::vec3(1.1, 0.3, 0.0)); // f
matrScale2 = glm::scale(glm::mat4(1.0f), glm::vec3(0.25, 0.25, 0.0)); //
matrRot = glm::rotate(glm::mat4(1.0f), angle, glm::vec3(0.0, 0.0, 1.0));

// Matricea pentru dreptunghiul albastru
myMatrix = resizeMatrix * matrTransl * matrScale1;
// Creare shader + transmitere variabile uniforme
CreateShaders();
myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
codCol = 1;
codColLocation = glGetUniformLocation(ProgramId, "codCuloare");
glUniform1i(codColLocation, codCol);
// Apelare DrawArrays
glDrawArrays(GL_POLYGON, 4, 4);
```

In programul principal

```
out vec4 gl_Position;
out vec4 ex_Color;
uniform mat4 myMatrix;

void main(void)
{
    gl_Position = myMatrix*in_Position;
    ex_Color = in_Color;
}
```

In vertex shader

Functii pentru transformari in OpenGL

- `glTranslate*(t); // glm::translate` Translația T_t de vector $t = (t_1, t_2, t_3)$

Functii pentru transformari in OpenGL

- `glTranslate*(t); // glm::translate` Translația T_t de vector $t = (t_1, t_2, t_3)$

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

Funcții pentru transformări în OpenGL

- `glTranslate*(t); // glm::translate` Translația T_t de vector $t = (t_1, t_2, t_3)$

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

- `glScale*(s); // glm::scale` Scalarea σ_s de factor $s = (s_1, s_2, s_3)$ (de-a lungul celor trei axe, centrul scalării fiind în origine - punct fix al transformării)

Functii pentru transformari in OpenGL

- `glTranslate*(t); // glm::translate` Translația T_t de vector $t = (t_1, t_2, t_3)$

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

- `glScale*(s); // glm::scale` Scalarea σ_s de factor $s = (s_1, s_2, s_3)$ (de-a lungul celor trei axe, centrul scalării fiind în origine - punct fix al transformării)

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

Functii pentru transformari in OpenGL

- `glTranslate*(t); // glm::translate` Translația T_t de vector $t = (t_1, t_2, t_3)$

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

- `glScale*(s); // glm::scale` Scalarea σ_s de factor $s = (s_1, s_2, s_3)$ (de-a lungul celor trei axe, centrul scalării fiind în origine - punct fix al transformării)

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

- `glRotate*(theta, u); // glm::rotate` Rotația $R_{u,\theta}$ de unghi θ și axă dată de vectorul u // Rotația 2D $R_{3,\theta}$ de axă Ox_3 (adică $u = (0, 0, 1)$ și unghi θ (centrul rotației fiind în origine - punct fix al transformării) este dată de

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \xrightarrow{R_{Ox_3,\theta}} \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

Functii pentru transformari in OpenGL

- `glTranslate*(t); // glm::translate` Translația T_t de vector $t = (t_1, t_2, t_3)$

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

- `glScale*(s); // glm::scale` Scalarea σ_s de factor $s = (s_1, s_2, s_3)$ (de-a lungul celor trei axe, centrul scalării fiind în origine - punct fix al transformării)

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

- `glRotate*(theta, u); // glm::rotate` Rotația $R_{u,\theta}$ de unghi θ și axă dată de vectorul u // Rotația 2D $R_{3,\theta}$ de axă Ox_3 (adică $u = (0, 0, 1)$ și unghi θ (centrul rotației fiind în origine - punct fix al transformării) este dată de

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \xrightarrow{R_{Ox_3,\theta}} \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

- Scalările și rotațiile au centrul în $O = (0, 0, 0)$, acesta este punct fix!

Aspecte teoretice

- ▶ **Motivație:** Transformările pot fi modelate cu ajutorul matricelor, **dar** este necesar un cadru în care transformările să fie reprezentate în mod uniform și compunerea lor să fie ușor de descris: **folosind "coordonate omogene" și considerând 4 coordonate.**

Aspecte teoretice

- ▶ **Motivație:** Transformările pot fi modelate cu ajutorul matricelor, **dar** este necesar un cadru în care transformările să fie reprezentate în mod uniform și compunerea lor să fie ușor de descris: **folosind "coordonate omogene" și considerând 4 coordonate.**
- ▶ **De reținut!**
 - (i) orice vârf are 4 coordonate
 - (ii) orice transformare este reprezentată (intern) folosind o matrice 4×4
 - (iii) compunerea transformărilor \leftrightarrow înmulțirea matricelor (în particular, ordinea contează!)

Din punctul de vedere al implementării...

- În OpenGL “vechi”: apelarea unor funcții specifice (**deprecated**) duce la generarea internă a matricelor corespunzătoare transformărilor, realizarea operațiilor cu acestea și aplicarea asupra vârfurilor, rezultând efectul de deplasare dorit

Din punctul de vedere al implementării...

- În OpenGL “vechi”: apelarea unor funcții specifice (**deprecated**) duce la generarea internă a matricelor corespunzătoare transformărilor, realizarea operațiilor cu acestea și aplicarea asupra vârfurilor, rezultând efectul de deplasare dorit
- În OpenGL “nou”: **nu există funcții specifice, ci trebuie manevrate în mod explicit matricele.** Se poate folosi biblioteca `glm` (OpenGL Mathematics)

Din punctul de vedere al implementării...

- În OpenGL “vechi”: apelarea unor funcții specifice (**deprecated**) duce la generarea internă a matricelor corespunzătoare transformărilor, realizarea operațiilor cu acestea și aplicarea asupra vârfurilor, rezultând efectul de deplasare dorit
- În OpenGL “nou”: **nu există funcții specifice, ci trebuie manevrate în mod explicit matricele.** Se poate folosi biblioteca **glm (OpenGL Mathematics)**

Q: unde/cum indicăm matricele pentru transformări?

Din punctul de vedere al implementării...

- În OpenGL “vechi”: apelarea unor funcții specifice (**deprecated**) duce la generarea internă a matricelor corespunzătoare transformărilor, realizarea operațiilor cu acestea și aplicarea asupra vârfurilor, rezultând efectul de deplasare dorit
- În OpenGL “nou”: **nu există funcții specifice, ci trebuie manevrate în mod explicit matricele.** Se poate folosi biblioteca `glm` ([OpenGL Mathematics](#))

Q: unde/cum indicăm matricele pentru transformări?
unde/cum efectuăm operațiile?

Din punctul de vedere al implementării...

- În OpenGL “vechi”: apelarea unor funcții specifice (**deprecated**) duce la generarea internă a matricelor corespunzătoare transformărilor, realizarea operațiilor cu acestea și aplicarea asupra vârfurilor, rezultând efectul de deplasare dorit
- În OpenGL “nou”: **nu există funcții specifice, ci trebuie manevrate în mod explicit matricele.** Se poate folosi biblioteca **glm (OpenGL Mathematics)**

Q: unde/cum indicăm matricele pentru transformări?
unde/cum efectuăm operațiile?

A: pot fi utilizate programul principal, shader-ele sau o combinație

Din punctul de vedere al implementării...

- În OpenGL “vechi”: apelarea unor funcții specifice (**deprecated**) duce la generarea internă a matricelor corespunzătoare transformărilor, realizarea operațiilor cu acestea și aplicarea asupra vârfurilor, rezultând efectul de deplasare dorit
- În OpenGL “nou”: **nu există funcții specifice, ci trebuie manevrate în mod explicit matricele.** Se poate folosi biblioteca `glm` ([OpenGL Mathematics](#))

Q: unde/cum indicăm matricele pentru transformări?
unde/cum efectuăm operațiile?

A: pot fi utilizate programul principal, shader-ele sau o combinație pot fi utilizate mai multe shader-e

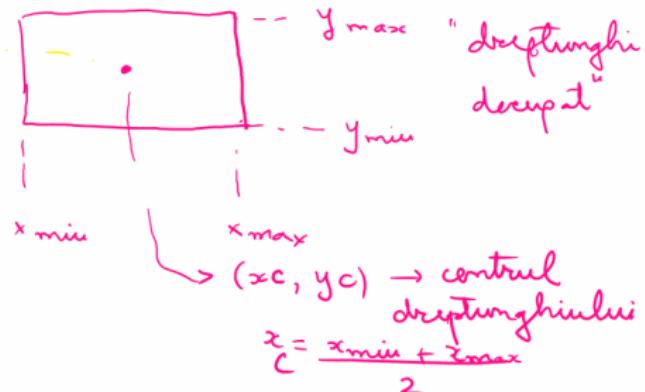
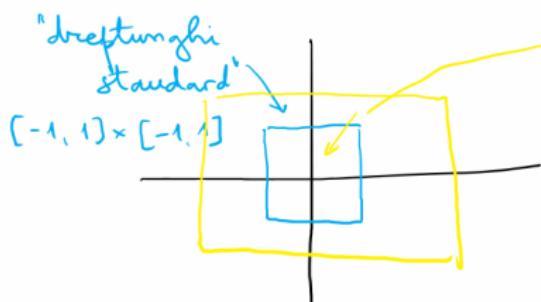
Despre `glm::ortho` - cerința (i) L3. Codul sursă 03_03_resize.cpp

- Dorim să desenăm o scenă 2D cu vârfuri având coordonata x între x_{min} și x_{max} și coordonata y între y_{min} și y_{max} . Se aplică funcția `glm::ortho(xmin, xmax, ymin, ymax)`. În codul sursă:
 $x_{min} = -400, x_{max} = 500, y_{min} = -200, y_{max} = 400$. Efectul funcției este transformarea dreptunghiului “decupat”
 $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$ în dreptunghiul “standard”
 $[-1, 1] \times [-1, 1]$.

Despre `glm::ortho` - cerința (i) L3. Codul sursă 03_03_resize.cpp

- ▶ Dorim să desenăm o scenă 2D cu vârfuri având coordonata x între x_{min} și x_{max} și coordonata y între y_{min} și y_{max} . Se aplică funcția `glm::ortho(xmin, xmax, ymin, ymax)`. În codul sursă:
 $x_{min} = -400, x_{max} = 500, y_{min} = -200, y_{max} = 400$. Efectul funcției este transformarea dreptunghiului “decupat”
 $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$ în dreptunghiul “standard”
 $[-1, 1] \times [-1, 1]$.
- ▶ Funcția `glm::ortho` este dată de compunerea dintre o translație și o scalare. **Atenție la ordine!**

Despre `glm::ortho` - cerință (i) L3. Codul sursă 03_03_resize.cpp



- "recenterare": $T(-x_c, -y_c) \Rightarrow \underline{\underline{M_T}}$ $x_c = \frac{x_{\min} + x_{\max}}{2}$
 $y_c = \frac{y_{\min} + y_{\max}}{2}$
 $(\Rightarrow$ dreptunghiul se centraza in origine)
 - "scalarm": Se multiplică cu factorii $\frac{2}{\Delta x}, \frac{2}{\Delta y} \Rightarrow \underline{\underline{M_S}}$

\Rightarrow matrices

$$\underline{M_S \times M_T}$$

Obs. importantă: Rotatiile și scalările au originea punctul foc. Dacă dorim să aplicăm o rotație sau o scalare cu centru oricare, avem de realizat o compunere:

fie C centrul rotației

- aplicăm translația de vector \overrightarrow{OC} : $T_{\overrightarrow{OC}}(M_1)$
 - aplicăm rotație / scalarea (M_2)
 - aplicăm translație de vector \overrightarrow{OC} : $T_{\overrightarrow{OC}}(M_3)$
- ⇒ matricea $M_3 \cdot M_2 \cdot M_1$

Transformări

Mihai-Sorin Stupariu

Sem. I, 2022 - 2023

Exemple de transformări

Coordonate omogene

Coordonate omogene - brevar teoretic

Functii pentru transformări în OpenGL

- `glm::translate` Translația T_t de vector $t = (t_1, t_2, t_3)$

Functii pentru transformări în OpenGL

- `glm::translate` Translația T_t de vector $t = (t_1, t_2, t_3)$

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

Funcții pentru transformări în OpenGL

- **glm::translate** Translația T_t de vector $t = (t_1, t_2, t_3)$

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

- **glm::scale** Scalarea σ_s de factor $s = (s_1, s_2, s_3)$ (de-a lungul celor trei axe, centrul scalării fiind în origine - punct fix al transformării)

Funcții pentru transformări în OpenGL

- **glm::translate** Translația T_t de vector $t = (t_1, t_2, t_3)$

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

- **glm::scale** Scalarea σ_s de factor $s = (s_1, s_2, s_3)$ (de-a lungul celor trei axe, centrul scalării fiind în origine - punct fix al transformării)

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

Funcții pentru transformări în OpenGL

- **glm::translate** Translația T_t de vector $t = (t_1, t_2, t_3)$

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

- **glm::scale** Scalarea σ_s de factor $s = (s_1, s_2, s_3)$ (de-a lungul celor trei axe, centrul scalării fiind în origine - punct fix al transformării)

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

- **glm::rotate** Rotația $\mathbb{R}_{u,\theta}$ de unghi θ și axă dată de vesorul u //Rotația 2D $\mathbb{R}_{3,\theta}$ de axă Ox_3 (adică $u = (0, 0, 1)$ și unghi θ (centrul rotației fiind în origine - punct fix al transformării) este dată de

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \xrightarrow{\mathbb{R}_{Ox_3, \theta}} \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

Funcții pentru transformări în OpenGL

- **glm::translate** Translația T_t de vector $t = (t_1, t_2, t_3)$

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

- **glm::scale** Scalarea σ_s de factor $s = (s_1, s_2, s_3)$ (de-a lungul celor trei axe, centrul scalării fiind în origine - punct fix al transformării)

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

- **glm::rotate** Rotația $\mathbb{R}_{u,\theta}$ de unghi θ și axă dată de vesorul u //Rotația 2D $\mathbb{R}_{3,\theta}$ de axă Ox_3 (adică $u = (0, 0, 1)$ și unghi θ (centrul rotației fiind în origine - punct fix al transformării) este dată de

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \xrightarrow{\mathbb{R}_{Ox_3, \theta}} \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

- Pot fi reprezentate în mod uniform transformările de mai sus?

Exemplu (1)

Fie aplicația afină f dată de

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} 2x_1 + 4x_2 \\ 3x_1 - x_2 \end{pmatrix}. \quad (1)$$

Obs. Utilizăm formalismul cu coloane pentru consistența lucrului cu matrice; aplicația se scrie și $f : \mathbb{R} \rightarrow \mathbb{R}$, $f(x_1, x_2) = (2x_1 + 4x_2, 3x_1 - x_2)$.

Exemplu (1)

Fie aplicația afină f dată de

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} 2x_1 + 4x_2 \\ 3x_1 - x_2 \end{pmatrix}. \quad (1)$$

Obs. Utilizăm formalismul cu coloane pentru consistența lucrului cu matrice; aplicația se scrie și $f : \mathbb{R} \rightarrow \mathbb{R}$, $f(x_1, x_2) = (2x_1 + 4x_2, 3x_1 - x_2)$.

- (i) Calculați $f(0, 0)$, $f(2, 5)$, $f(e_1)$, $f(e_2)$.

Exemplu (1)

Fie aplicația afină f dată de

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} 2x_1 + 4x_2 \\ 3x_1 - x_2 \end{pmatrix}. \quad (1)$$

Obs. Utilizăm formalismul cu coloane pentru consistența lucrului cu matrice; aplicația se scrie și $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, $f(x_1, x_2) = (2x_1 + 4x_2, 3x_1 - x_2)$.

- Calculați $f(0, 0)$, $f(2, 5)$, $f(e_1)$, $f(e_2)$.
- Scrieți relația (1) sub forma matriceală. Ce observați? (legătura cu $f(e_1), f(e_2)$).

Exemplu (1) - Calcule

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} 2x_1 + 4x_2 \\ 3x_1 - x_2 \end{pmatrix}; \quad f(x_1, x_2) = (2x_1 + 4x_2, 3x_1 - x_2)$$

(i) Calculați $f(0, 0)$, $f(2, 5)$, $f(e_1)$, $f(e_2)$.

$$f(0, 0) = (0, 0); \quad \begin{pmatrix} 0 \\ 0 \end{pmatrix} \mapsto \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$f(2, 5) = (24, 1); \quad \begin{pmatrix} 2 \\ 5 \end{pmatrix} \mapsto \begin{pmatrix} 24 \\ 1 \end{pmatrix}$$

$$f(e_1) = f(1, 0) = (2, 3); \quad \begin{pmatrix} 1 \\ 0 \end{pmatrix} \mapsto \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

$$f(e_2) = f(0, 1) = (4, -1); \quad \begin{pmatrix} 0 \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} 4 \\ -1 \end{pmatrix}$$

Exemplu (1) - Calcule

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} 2x_1 + 4x_2 \\ 3x_1 - x_2 \end{pmatrix}; \quad f(x_1, x_2) = (2x_1 + 4x_2, 3x_1 - x_2)$$

(ii) Scrieți f folosind reprezentarea matriceală. Ce observați? (legătura cu $f(e_1), f(e_2)$).

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} 2x_1 + 4x_2 \\ 3x_1 - x_2 \end{pmatrix} = \begin{pmatrix} 2 & 4 \\ 3 & -1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$


coloanele lui M_f
 $f(e_1)$ $f(e_2)$

nu sunt
 $f(e_1) \wedge f(e_2)$

Exemplu (2)

Aceleași cerințe pentru aplicația afină f dată de

$$f(x_1, x_2) = (2x_1 + 4x_2 + 5, 3x_1 - x_2 - 2) \quad (2)$$

Proprietăți - de reținut!

(i) Pentru f aplicație afină 2D dată de

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad (3)$$

au loc relațiile

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} 0 \\ 0 \end{pmatrix}; \quad \begin{pmatrix} 1 \\ 0 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} a_{11} \\ a_{21} \end{pmatrix}; \quad \begin{pmatrix} 0 \\ 1 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} a_{12} \\ a_{22} \end{pmatrix}$$

Coloanele matricei sunt exact $f(e_1), f(e_2)$.

Proprietăți - de reținut!

(i) Pentru f aplicație afină 2D dată de

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad (3)$$

au loc relațiile

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} 0 \\ 0 \end{pmatrix}; \quad \begin{pmatrix} 1 \\ 0 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} a_{11} \\ a_{21} \end{pmatrix}; \quad \begin{pmatrix} 0 \\ 1 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} a_{12} \\ a_{22} \end{pmatrix}$$

Coloanele matricei sunt exact $f(e_1), f(e_2)$.

(ii) Pentru f aplicație afină 2D dată de

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \quad (4)$$

au loc relațiile

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}; \quad \begin{pmatrix} 1 \\ 0 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} a_{11} \\ a_{21} \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}; \quad \begin{pmatrix} 0 \\ 1 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} a_{12} \\ a_{22} \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

Rotații 2D

Rotația 2D (cu axa de rotație $u = (0, 0, 1)$) de unghi θ , cu centrul în origine are matricea 2×2 asociată dată de

$$\begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix},$$

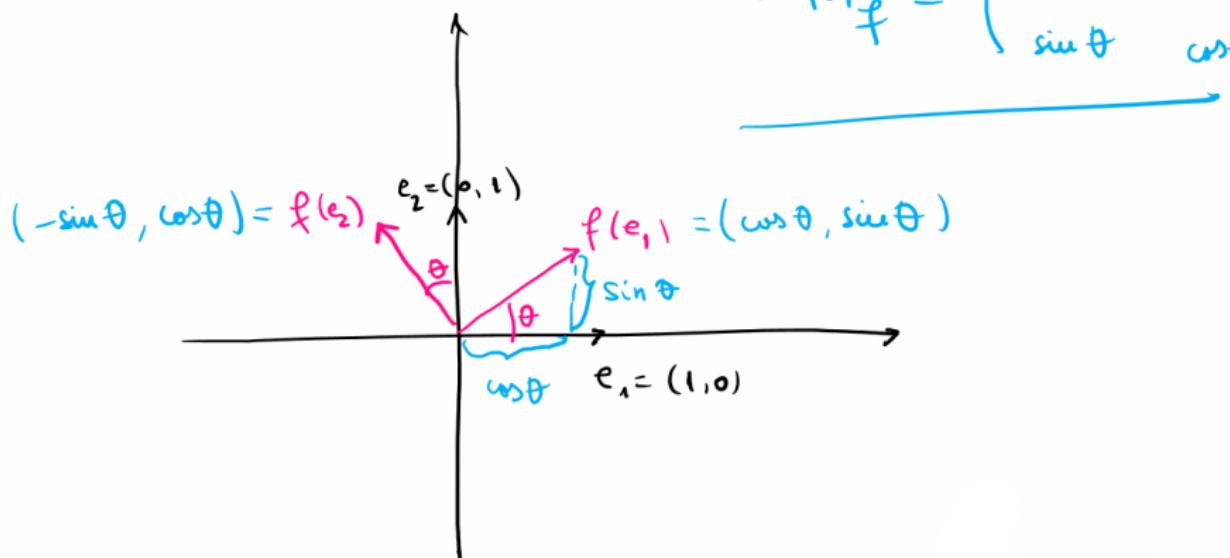
iar matricea 3×3 este

$$\begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Figura - rotații 2D

$$f = R_{0,\theta} \quad \text{in plane 2D}$$

$$\Rightarrow M_f = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$



Vârfuri și direcții - rolul celei de-a 4 coordonate

- **Motivație:** Pot fi reprezentate în mod uniform translațiile, scalările, rotațiile? Este necesar un cadru în care transformările să fie reprezentate în mod uniform și compunerea lor să fie ușor de descris: folosind “coordonate omogene” și considerând 4 coordonate.

Vârfuri și direcții - rolul celei de-a 4 coordonate

- ▶ **Motivație:** Pot fi reprezentate în mod uniform translațiile, scalările, rotațiile? Este necesar un cadru în care transformările să fie reprezentate în mod uniform și compunerea lor să fie ușor de descris: folosind “**coordonate omogene**” și considerând **4 coordonate**.
- ▶ Coordonatele omogene verifică proprietatea fundamentală

$$[\alpha : \beta : \gamma : \delta] = [\lambda\alpha : \lambda\beta : \lambda\gamma : \lambda\delta], \quad \forall \lambda \neq 0, (\alpha, \beta, \gamma, \delta) \neq (0, 0, 0, 0).$$

De exemplu, $[2 : 3 : -1 : 4] = [4 : 6 : -2 : 8] = [-6 : -9 : 3 : -12]$, dar $[2 : 3 : -1 : 4] \neq [4 : -1 : 3 : 2]$.

Vârfuri și direcții - rolul celei de-a 4 coordonate

- ▶ **Motivație:** Pot fi reprezentate în mod uniform translațiile, scalările, rotațiile? Este necesar un cadru în care transformările să fie reprezentate în mod uniform și compunerea lor să fie ușor de descris: folosind “**coordonate omogene**” și considerând **4 coordonate**.
- ▶ **Coordonatele omogene verifică proprietatea fundamentală**

$$[\alpha : \beta : \gamma : \delta] = [\lambda\alpha : \lambda\beta : \lambda\gamma : \lambda\delta], \quad \forall \lambda \neq 0, (\alpha, \beta, \gamma, \delta) \neq (0, 0, 0, 0).$$

De exemplu, $[2 : 3 : -1 : 4] = [4 : 6 : -2 : 8] = [-6 : -9 : 3 : -12]$, dar $[2 : 3 : -1 : 4] \neq [4 : -1 : 3 : 2]$.

- ▶ Unui **vârf** de coordonate (x_1, x_2, x_3) , notate și (x, y, z) î se asociază **coordonatele omogene**

$$[x_1 : x_2 : x_3 : 1] \text{ (sau } [x : y : z : 1]).$$

Vârfuri și direcții - rolul celei de-a 4 coordonate

- ▶ **Motivație:** Pot fi reprezentate în mod uniform translațiile, scalările, rotațiile? Este necesar un cadru în care transformările să fie reprezentate în mod uniform și compunerea lor să fie ușor de descris: folosind “**coordonate omogene” și considerând 4 coordonate.**
- ▶ **Coordonatele omogene verifică proprietatea fundamentală**

$$[\alpha : \beta : \gamma : \delta] = [\lambda\alpha : \lambda\beta : \lambda\gamma : \lambda\delta], \quad \forall \lambda \neq 0, (\alpha, \beta, \gamma, \delta) \neq (0, 0, 0, 0).$$

De exemplu, $[2 : 3 : -1 : 4] = [4 : 6 : -2 : 8] = [-6 : -9 : 3 : -12]$, dar $[2 : 3 : -1 : 4] \neq [4 : -1 : 3 : 2]$.

- ▶ Unui **vârf** de coordonate (x_1, x_2, x_3) , notate și (x, y, z) i se asociază **coordonatele omogene**

$$[x_1 : x_2 : x_3 : 1] \text{ (sau } [x : y : z : 1]).$$

- ▶ Unei **direcții** date de vectorul (v_1, v_2, v_3) , i se asociază **coordonatele omogene**

$$[v_1 : v_2 : v_3 : 0].$$

Transformări – reprezentare matriceală

- Fie $f : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ o aplicație afină arbitrară a lui \mathbb{R}^3 , dată prin

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}. \quad (5)$$

(a da o aplicație afină f revine la a da matricele $(a_{ij})_{i,j}$ și $(b_i)_i$.

Transformări – reprezentare matriceală

- Fie $f : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ o aplicație afină arbitrară a lui \mathbb{R}^3 , dată prin

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}. \quad (5)$$

(a da o aplicație afină f revine la a da matricele $(a_{ij})_{i,j}$ și $(b_i)_i$.

- Lui f îi corespunde o matrice 4×4

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

Transformări – reprezentare matriceală

- Fie $f : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ o aplicație afină arbitrară a lui \mathbb{R}^3 , dată prin

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}. \quad (5)$$

(a da o aplicație afină f revine la a da matricele $(a_{ij})_{i,j}$ și $(b_i)_i$).

- Lui f îi corespunde o matrice 4×4

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

- **Principiu:** Dat un vârf / o direcție având coordonate omogene reprezentate de un vector coloană

$$\xi = \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_0 \end{pmatrix},$$

aplicarea transformării f generează un nou vector coloană, și anume

$$M_f \cdot \xi.$$

Exemple

- În momentul apelării funcției `glm::translate3f(t1, t2, t3)`, se generează (și manevrează) matricea 4×4

$$M_{T_t} = \begin{pmatrix} 1 & 0 & 0 & t_1 \\ 0 & 1 & 0 & t_2 \\ 0 & 0 & 1 & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Exemple

- ▶ În momentul apelării funcției `glm::translate3f(t1, t2, t3)`, se generează (și manevrează) matricea 4×4

$$M_{T_t} = \begin{pmatrix} 1 & 0 & 0 & t_1 \\ 0 & 1 & 0 & t_2 \\ 0 & 0 & 1 & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

- ▶ În momentul apelării funcției `glm::scale3f(s1, s2, s3)`, se generează (și manevrează) matricea 4×4

$$M_{\sigma_s} = \begin{pmatrix} s_1 & 0 & 0 & 0 \\ 0 & s_2 & 0 & 0 \\ 0 & 0 & s_3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Completere

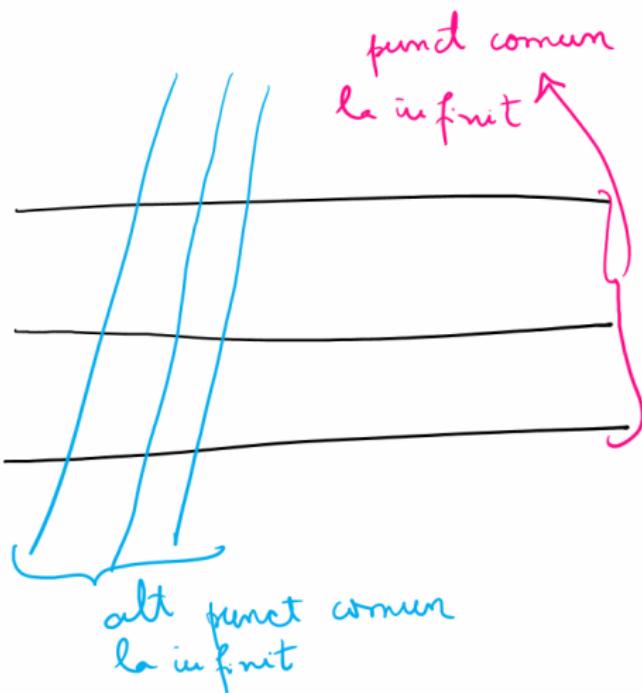
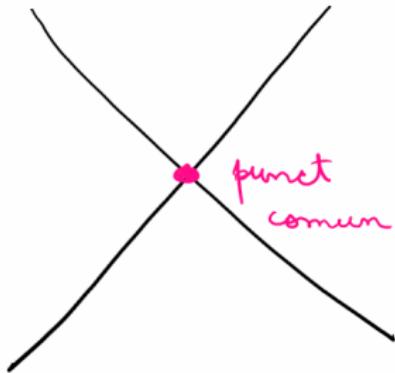
Pe lângă funcțiile din biblioteca glm, transformările de modelare pot fi apelate în codul sursă indicând explicit matricea 4×4 asociată. Prin convenție, elementele sunt introduse pe coloane, de sus în jos, de la stânga la dreapta

$$\begin{pmatrix} 0 & 4 & 8 & 12 \\ 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{pmatrix}.$$

În continuare...

Cum se ajunge la introducerea coordonatelor omogene? Care sunt fundamentele teoretice care stau la baza utilizării lor?

Construcție I - geometrică



⇒ dreapta de la infinit

Construcție I - geometrică

- ▶ Se obține **dreapta de la infinit** (dată de punctele de la infinit)

Construcție I - geometrică

- ▶ Se obține **dreapta de la infinit** (dată de punctele de la infinit)
- ▶ Planul \mathbb{R}^2 , prin reuniune cu dreapta de la infinit (dată de punctele de la infinit) formează **planul proiectiv real, notat $\mathbb{P}^2\mathbb{R}$** .

Construcție I - geometrică

- ▶ Se obține **dreapta de la infinit** (dată de punctele de la infinit)
- ▶ Planul \mathbb{R}^2 , prin reuniune cu dreapta de la infinit (dată de punctele de la infinit) formează **planul proiectiv real, notat $\mathbb{P}^2\mathbb{R}$** .
- ▶ Este greu să ne “**reprezentăm intuitiv**” planul proiectiv real. Motivul: “nu admite scufundare în spațiul \mathbb{R}^3 ”. Alte exemple de suprafețe care nu admit astfel de scufundări: banda lui Möbius (infinită); **sticla lui Klein**.

Construcție I - geometrică

- ▶ Se obține **dreapta de la infinit** (dată de punctele de la infinit)
- ▶ Planul \mathbb{R}^2 , prin reuniune cu dreapta de la infinit (dată de punctele de la infinit) formează **planul proiectiv real, notat $\mathbb{P}^2\mathbb{R}$** .
- ▶ Este greu să ne “**reprezentăm intuitiv**” planul proiectiv real. Motivul: “nu admite scufundare în spațiul \mathbb{R}^3 ”. Alte exemple de suprafețe care nu admit astfel de scufundări: banda lui Möbius (infinită); **sticla lui Klein**.
- ▶ Puteți descrie dreapta proiectivă reală $\mathbb{P}^1\mathbb{R}$? Dar dreapta proiectivă complexă $\mathbb{P}^1\mathbb{C}$?

Construcție I - geometrică

- ▶ Se obține **dreapta de la infinit** (dată de punctele de la infinit)
- ▶ Planul \mathbb{R}^2 , prin reuniune cu dreapta de la infinit (dată de punctele de la infinit) formează **planul proiectiv real, notat $\mathbb{P}^2\mathbb{R}$** .
- ▶ Este greu să ne “**reprezentăm intuitiv**” planul proiectiv real. Motivul: “nu admite scufundare în spațiul \mathbb{R}^3 ”. Alte exemple de suprafețe care nu admit astfel de scufundări: banda lui Möbius (infinită); **sticla lui Klein**.
- ▶ Putem descrie dreapta proiectivă reală $\mathbb{P}^1\mathbb{R}$? Dar dreapta proiectivă complexă $\mathbb{P}^1\mathbb{C}$?
- ▶ Aplicabilitate a geometriei proiective (de exemplu): **curbe eliptice** → **criptografie și securitate**.

Construcție II - algebrică

- $\mathbb{P}^2\mathbb{R}$ poate fi descris **algebric**, folosind **coordonate omogene**.

Construcție II - algebrică

- ▶ $\mathbb{P}^2\mathbb{R}$ poate fi descris **algebric**, folosind **coordonate omogene**.
- ▶ Pe $\mathbb{R}^3 \setminus \{(0, 0, 0)\}$ se introduce relația de echivalență \sim dată prin $(X_1, X_2, X_0) \sim (Y_1, Y_2, Y_0)$ dacă și numai dacă există $\lambda \neq 0$ astfel ca $Y_0 = \lambda X_0$, $Y_1 = \lambda X_1$, $Y_2 = \lambda X_2$ (este același λ , ca factor de proporționalitate!).

Construcție II - algebrică

- ▶ $\mathbb{P}^2\mathbb{R}$ poate fi descris **algebric**, folosind **coordonate omogene**.
- ▶ Pe $\mathbb{R}^3 \setminus \{(0, 0, 0)\}$ se introduce relația de echivalență \sim dată prin $(X_1, X_2, X_0) \sim (Y_1, Y_2, Y_0)$ dacă și numai dacă există $\lambda \neq 0$ astfel ca $Y_0 = \lambda X_0$, $Y_1 = \lambda X_1$, $Y_2 = \lambda X_2$ (este același λ , ca factor de proporționalitate!).
- ▶ Clasa de echivalență a unui element $X = (X_1, X_2, X_0)$ va fi notată cu $[X_1 : X_2 : X_0]$. Astfel, de exemplu, avem $[1 : 2 : -5] = [2 : 4 : -10]$. În schimb, $[1 : 2 : -5] \neq [1 : 2 : 5]$ și $[1 : 2 : -5] \neq [-5 : 2 : 1]$.

Construcție II - algebrică

- ▶ $\mathbb{P}^2\mathbb{R}$ poate fi descris **algebric**, folosind **coordonate omogene**.
- ▶ Pe $\mathbb{R}^3 \setminus \{(0, 0, 0)\}$ se introduce relația de echivalență \sim dată prin $(X_1, X_2, X_0) \sim (Y_1, Y_2, Y_0)$ dacă și numai dacă există $\lambda \neq 0$ astfel ca $Y_0 = \lambda X_0$, $Y_1 = \lambda X_1$, $Y_2 = \lambda X_2$ (este același λ , ca factor de proporționalitate!).
- ▶ Clasa de echivalență a unui element $X = (X_1, X_2, X_0)$ va fi notată cu $[X_1 : X_2 : X_0]$. Astfel, de exemplu, avem $[1 : 2 : -5] = [2 : 4 : -10]$. În schimb, $[1 : 2 : -5] \neq [1 : 2 : 5]$ și $[1 : 2 : -5] \neq [-5 : 2 : 1]$.
- ▶ Se consideră mulțimea claselor de echivalență
$$\mathcal{C} = \{[X_1 : X_2 : X_0] | (X_1, X_2, X_0) \in \mathbb{R}^3 \setminus \{0\}\}.$$

Construcție II - algebrică

- ▶ $\mathbb{P}^2\mathbb{R}$ poate fi descris **algebric**, folosind **coordonate omogene**.
- ▶ Pe $\mathbb{R}^3 \setminus \{(0, 0, 0)\}$ se introduce relația de echivalență \sim dată prin $(X_1, X_2, X_0) \sim (Y_1, Y_2, Y_0)$ dacă și numai dacă există $\lambda \neq 0$ astfel ca $Y_0 = \lambda X_0$, $Y_1 = \lambda X_1$, $Y_2 = \lambda X_2$ (este același λ , ca factor de proporționalitate!).
- ▶ Clasa de echivalență a unui element $X = (X_1, X_2, X_0)$ va fi notată cu $[X_1 : X_2 : X_0]$. Astfel, de exemplu, avem $[1 : 2 : -5] = [2 : 4 : -10]$. În schimb, $[1 : 2 : -5] \neq [1 : 2 : 5]$ și $[1 : 2 : -5] \neq [-5 : 2 : 1]$.
- ▶ Se consideră mulțimea claselor de echivalență

$$\mathcal{C} = \{[X_1 : X_2 : X_0] | (X_1, X_2, X_0) \in \mathbb{R}^3 \setminus \{0\}\}.$$
- ▶ **Terminologie.** Pentru un element ξ , dacă $\xi = [X_1 : X_2 : X_0]$, atunci (X_1, X_2, X_0) se numesc **coordonatele omogene ale lui ξ** . Coordonatele omogene sunt definite și sunt unice până la înmulțirea cu un scalar nenul.

Legătura dintre cele două construcții

(i) Există o aplicație de incluziune naturală

$$\iota : \mathbb{R}^2 \hookrightarrow \mathcal{C}, \quad \iota(x_1, x_2) = [x_1 : x_2 : 1].$$

Aplicația ι este injectivă.

Justificare:

Îi se arată că (x_1, x_2) și (y_1, y_2) sunt a.s. $\iota(x_1, x_2) = \iota(y_1, y_2) \Rightarrow$

$$\Rightarrow [x_1 : x_2 : 1] = [y_1 : y_2 : 1] \Rightarrow \exists \lambda \neq 0 \text{ a.s.}$$

$$(x_1, x_2, 1) = (\lambda y_1, \lambda y_2, \lambda \cdot 1) \Rightarrow \begin{aligned} x_1 &= \lambda y_1 \\ x_2 &= \lambda y_2 \\ 1 &= \lambda \cdot 1 \end{aligned} \Rightarrow$$

$$\lambda = 1 \Rightarrow x_1 = y_1, x_2 = y_2 \Rightarrow (x_1, x_2) = (y_1, y_2)$$

Legătura dintre cele două construcții

- (i) Există o aplicație de incluziune naturală

$$\iota : \mathbb{R}^2 \hookrightarrow \mathcal{C}, \quad \iota(x_1, x_2) = [x_1 : x_2 : 1].$$

Aplicația ι este injectivă.

Legătura dintre cele două construcții

- (i) Există o aplicație de incluziune naturală

$$\iota : \mathbb{R}^2 \hookrightarrow \mathcal{C}, \quad \iota(x_1, x_2) = [x_1 : x_2 : 1].$$

Aplicația ι este injectivă.

- (ii) Pe de altă parte, fie $[\delta]$ clasa de echivalență a unei drepte δ modulo relația de paralelism (intuitiv, o astfel de clasă de echivalență 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) un vector director al lui δ (atunci orice vector de forma $(\lambda v_1, \lambda v_2)$, cu $\lambda \neq 0$ este, la rândul său, un vector director al lui δ); acești vectori sunt vectori direcatori pentru orice dreaptă paralelă cu δ . Avem asocierea

$$[\delta](alegem (v_1, v_2)) \mapsto [v_1 : v_2 : 0] \in \mathcal{C}.$$

Definiția este coerentă.

Legătura dintre cele două construcții

- (i) Există o aplicație de incluziune naturală

$$\iota : \mathbb{R}^2 \hookrightarrow \mathcal{C}, \quad \iota(x_1, x_2) = [x_1 : x_2 : 1].$$

Aplicația ι este injectivă.

- (ii) Pe de altă parte, fie $[\delta]$ clasa de echivalență a unei drepte δ modulo relația de paralelism (intuitiv, o astfel de clasă de echivalență 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) un vector director al lui δ (atunci orice vector de forma $(\lambda v_1, \lambda v_2)$, cu $\lambda \neq 0$ este, la rândul său, un vector director al lui δ); acești vectori sunt vectori direcționali pentru orice dreaptă paralelă cu δ . Avem asocierea

$$[\delta](\text{alegem } (v_1, v_2)) \mapsto [v_1 : v_2 : 0] \in \mathcal{C}.$$

Definiția este coerentă.

- **Concluzie:** Există o corespondență naturală 1:1 între planul proiectiv real $\mathbb{P}^2\mathbb{R}$ (construit geometric) și mulțimea \mathcal{C} a claselor de echivalență (construită algebraic). Aceasta în seamnă că în $\mathbb{P}^2\mathbb{R}$ sunt folosite coordonate omogene.

Sinteză

În concluzie, planul proiectiv real $\mathbb{P}^2\mathbb{R}$ conține două tipuri de elemente:

Sinteză

În concluzie, planul proiectiv real $\mathbb{P}^2\mathbb{R}$ conține două tipuri de elemente:

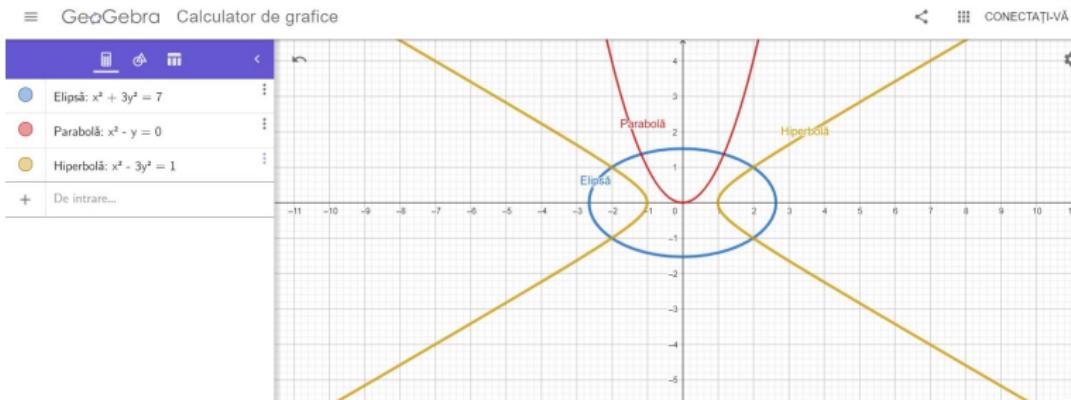
- ▶ **"punkte reale"**: elemente de forma $P = [X_1 : X_2 : X_0]$ cu $X_0 \neq 0$; P îi corespunde punctului $\left(\frac{X_1}{X_0}, \frac{X_2}{X_0}\right)$ din spațiul geometric \mathbb{R}^2 (deoarece $[X_1 : X_2 : X_0] = [\frac{X_1}{X_0} : \frac{X_2}{X_0} : 1]$),

Sinteză

În concluzie, planul proiectiv real $\mathbb{P}^2\mathbb{R}$ conține două tipuri de elemente:

- ▶ **"punkte reale"**: elemente de forma $P = [X_1 : X_2 : X_0]$ cu $X_0 \neq 0$; P îi corespunde punctului $\left(\frac{X_1}{X_0}, \frac{X_2}{X_0}\right)$ din spațiul geometric \mathbb{R}^2 (deoarece $[X_1 : X_2 : X_0] = [\frac{X_1}{X_0} : \frac{X_2}{X_0} : 1]$),
- ▶ **"punkte de la infinit"**: elemente de forma $Q = [X_1 : X_2 : 0]$. Mulțimea punctelor de la infinit formează o dreaptă, numită **dreapta de la infinit**, având ecuația $X_0 = 0$.

Câte puncte au la infinit elipsa, parabola, hiperbola?



Varianta algebrică

- ▶ Coordonate omogene: se utilizează polinoame omogene (toate monoamele au același grad).

Varianta algebrică

- ▶ Coordonate omogene: se utilizează polinoame omogene (toate monoamele au același grad).
- ▶ Oricărui loc geometric din \mathbb{R}^2 descris printr-o ecuație polinomială implicită îi corespunde un loc geometric din planul $\mathbb{P}\mathbb{R}^2$, descris printr-o ecuație ce se obține “omogenizând” ecuația inițială.

► Exemplul 1:

- Cercul $x_1^2 + x_2^2 - 1 = 0$.

► Exemplul 1:

- Cercul $x_1^2 + x_2^2 - 1 = 0$.
- Omogenizare: $x_1 = \frac{x_1}{X_0}$, $x_2 = \frac{x_2}{X_0}$.

► Exemplul 1:

- Cercul $x_1^2 + x_2^2 - 1 = 0$.
- Omogenizare: $x_1 = \frac{x_1}{X_0}$, $x_2 = \frac{x_2}{X_0}$.
- Înlocuire + calcule: $X_1^2 + X_2^2 - X_0^2 = 0$.

► **Exemplul 1:**

- Cercul $x_1^2 + x_2^2 - 1 = 0$.
- Omogenizare: $x_1 = \frac{x_1}{x_0}$, $x_2 = \frac{x_2}{x_0}$.
- Înlocuire + calcule: $X_1^2 + X_2^2 - X_0^2 = 0$.
- Puncte de la infinit?

$$\begin{cases} X_1^2 + X_2^2 - X_0^2 = 0 \\ X_0 = 0 \end{cases} \Rightarrow X_1^2 + X_2^2 = 0 \Rightarrow X_1 = 0, X_2 = 0.$$

Nu este posibil ca $X_0 = X_1 = X_2 = 0$ (așa am definit relația de echivalență), deci **nu avem puncte la infinit** în acest caz.

► Exemplul 1:

- Cercul $x_1^2 + x_2^2 - 1 = 0$.
- Omogenizare: $x_1 = \frac{x_1}{x_0}$, $x_2 = \frac{x_2}{x_0}$.
- Înlocuire + calcule: $X_1^2 + X_2^2 - X_0^2 = 0$.
- Puncte de la infinit?

$$\begin{cases} X_1^2 + X_2^2 - X_0^2 = 0 \\ X_0 = 0 \end{cases} \Rightarrow X_1^2 + X_2^2 = 0 \Rightarrow X_1 = 0, X_2 = 0.$$

Nu este posibil ca $X_0 = X_1 = X_2 = 0$ (așa am definit relația de echivalență), deci **nu avem puncte la infinit** în acest caz.

► Exemplul 2:

- Hiperbola $x_1^2 - x_2^2 - 1 = 0$.

- omogenizare: $x_1 = \frac{x_1}{x_0}$ și $x_2 = \frac{x_2}{x_0}$

- înlocuire $\rightarrow \frac{x_1^2 - x_2^2}{x_0^2} - 1 = 0 \Rightarrow x_1^2 - x_2^2 - x_0^2 = 0$

- puncte la infinit? $\begin{cases} x_1^2 - x_2^2 - x_0^2 = 0 \\ x_0 = 0 \end{cases} \Rightarrow x_1^2 - x_2^2 = 0$

$$\Rightarrow x_1 = x_2 \text{ sau } x_1 = -x_2 \Rightarrow \begin{cases} x_1 = x_2 = t \\ x_1 = -x_2 = t \end{cases} \Rightarrow \begin{cases} [t:t:0] = [1:1:0] \\ [t:-t:0] = [1:-1:0] \end{cases}$$

} pct.

Puncte de la infinit: concluzii+exemplu important

- ▶ Oricărui loc geometric L din \mathbb{R}^2 i se poate asocia un loc geometric \bar{L} din $\mathbb{P}^2\mathbb{R}$, prin **completarea cu puncte de la infinit**. Dacă L este descris prin ecuații implice, \bar{L} este dat prin **omogenizarea** ecuațiilor lui L .

Puncte de la infinit: concluzii+exemplu important

- ▶ Oricărui loc geometric L din \mathbb{R}^2 i se poate asocia un loc geometric \bar{L} din $\mathbb{P}^2\mathbb{R}$, prin **completarea cu puncte de la infinit**. Dacă L este descris prin ecuații implicate, \bar{L} este dat prin **omogenizarea** ecuațiilor lui L .
- ▶ Astfel, dacă L este dat prin ecuația

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

omogenizarea ecuației lui L se obține astfel: se notează $x_i = \frac{x_i}{x_0}$ ($i = 1, 2$) și se efectuează înlocuirile în ecuația lui L , obținând, după eliminarea numitorului, ecuația omogenă a lui \bar{L}

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

Puncte de la infinit: concluzii+exemplu important

- ▶ Oricărui loc geometric L din \mathbb{R}^2 i se poate asocia un loc geometric \bar{L} din $\mathbb{P}^2\mathbb{R}$, prin **completarea cu puncte de la infinit**. Dacă L este descris prin ecuații implicate, \bar{L} este dat prin **omogenizarea** ecuațiilor lui L .
- ▶ Astfel, dacă L este dat prin ecuația

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

omogenizarea ecuației lui L se obține astfel: se notează $x_i = \frac{x_i}{x_0}$ ($i = 1, 2$) și se efectuează înlocuirile în ecuația lui L , obținând, după eliminarea numitorului, ecuația omogenă a lui \bar{L}

- ▶ Pentru locul geometric

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{10} = 0 \\ a_{21}x_1 + a_{22}x_2 + a_{20} = 0 \end{cases}$$

prin omogenizare se obține locul geometric

$$\begin{cases} a_{11}X_1 + a_{12}X_2 + a_{10}X_0 = 0 \\ a_{21}X_1 + a_{22}X_2 + a_{20}X_0 = 0 \end{cases}$$

sau matriceal

$$\begin{pmatrix} a_{11} & a_{12} & a_{10} \\ a_{21} & a_{22} & a_{20} \end{pmatrix} \cdot \begin{pmatrix} X_1 \\ X_2 \\ X_0 \end{pmatrix} = 0 \begin{pmatrix} a_{11}x_1 + a_{12}x_2 + a_{10} = 0 \\ a_{21}x_1 + a_{22}x_2 + a_{20} = 0 \end{pmatrix}$$



Pentru aplicații

- Fie $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ aplicația dată de

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} a_{10} \\ a_{20} \end{pmatrix} = \begin{pmatrix} a_{11}x_1 + a_{12}x_2 + a_{10} \\ a_{21}x_1 + a_{22}x_2 + a_{20} \end{pmatrix}$$

Pentru aplicații

- Fie $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ aplicația dată de

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} a_{10} \\ a_{20} \end{pmatrix} = \begin{pmatrix} a_{11}x_1 + a_{12}x_2 + a_{10} \\ a_{21}x_1 + a_{22}x_2 + a_{20} \end{pmatrix}$$

- În coordonate omogene (după omogenizare), această transformare se scrie

$$\begin{pmatrix} X_1 \\ X_2 \\ X_0 \end{pmatrix} \mapsto \begin{pmatrix} a_{11}X_1 + a_{12}X_2 + a_{10}X_0 \\ a_{21}X_1 + a_{22}X_2 + a_{20}X_0 \\ X_0 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{10} \\ a_{21} & a_{22} & a_{20} \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} X_1 \\ X_2 \\ X_0 \end{pmatrix}$$

Pentru aplicații

- Fie $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ aplicația dată de

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} a_{10} \\ a_{20} \end{pmatrix} = \begin{pmatrix} a_{11}x_1 + a_{12}x_2 + a_{10} \\ a_{21}x_1 + a_{22}x_2 + a_{20} \end{pmatrix}$$

- În coordonate omogene (după omogenizare), această transformare se scrie

$$\begin{pmatrix} X_1 \\ X_2 \\ X_0 \end{pmatrix} \mapsto \begin{pmatrix} a_{11}X_1 + a_{12}X_2 + a_{10}X_0 \\ a_{21}X_1 + a_{22}X_2 + a_{20}X_0 \\ X_0 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{10} \\ a_{21} & a_{22} & a_{20} \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} X_1 \\ X_2 \\ X_0 \end{pmatrix}$$

- Analog, pentru $f : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ se obține o matrice 4×4 .

Pentru aplicații

- Fie $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ aplicația dată de

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} a_{10} \\ a_{20} \end{pmatrix} = \begin{pmatrix} a_{11}x_1 + a_{12}x_2 + a_{10} \\ a_{21}x_1 + a_{22}x_2 + a_{20} \end{pmatrix}$$

- În coordonate omogene (după omogenizare), această transformare se scrie

$$\begin{pmatrix} X_1 \\ X_2 \\ X_0 \end{pmatrix} \mapsto \begin{pmatrix} a_{11}X_1 + a_{12}X_2 + a_{10}X_0 \\ a_{21}X_1 + a_{22}X_2 + a_{20}X_0 \\ X_0 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{10} \\ a_{21} & a_{22} & a_{20} \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} X_1 \\ X_2 \\ X_0 \end{pmatrix}$$

- Analog, pentru $f : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ se obține o matrice 4×4 .
- Faptul că pe ultima linie este $(0, 0, 1)$ sau $(0, 0, 0, 1)$ se interpretează prin faptul că “nu se schimbă poziția dreptei de la infinit”.

De reținut: vârfuri și direcții - rolul celei de-a 4 coordonate

- Unui **vârf** de coordonate (x_1, x_2, x_3) , notate și (x, y, z) , i se asociază **coordonatele omogene**

$$[x_1 : x_2 : x_3 : 1] \text{ (sau } [x : y : z : 1]\text{)}.$$

De reținut: vârfuri și direcții - rolul celei de-a 4 coordonate

- Unui **vârf** de coordonate (x_1, x_2, x_3) , notate și (x, y, z) , i se asociază **coordonatele omogene**

$$[x_1 : x_2 : x_3 : 1] \text{ (sau } [x : y : z : 1]).$$

- Unei **direcții** date de vectorul (v_1, v_2, v_3) , i se asociază **coordonatele omogene**

$$[v_1 : v_2 : v_3 : 0].$$

De reținut: transformări – reprezentare matriceală

- Fie $f : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ o aplicație afină arbitrară a lui \mathbb{R}^3 , dată prin

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \xrightarrow{f} \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}. \quad (6)$$

(a da o aplicație afină f revine la a da matricele $(a_{ij})_{i,j}$ și $(b_i)_i$).

- Lui f îi corespunde o matrice 4×4

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

- **Principiu:** Dat un vârf / o direcție având coordonate omogene reprezentate de un vector coloană

$$\xi = \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_0 \end{pmatrix},$$

aplicarea transformării f generează un nou vector coloană, și anume

$$\mathcal{M}_f \cdot \xi.$$

Transformări (III). Reperul de vizualizare (poziția camerei)

Mihai-Sorin Stupariu

Sem. I, 2022 - 2023

Coordinate de modelare. Coordinate de vizualizare

Reperul de vizualizare

Transformarea de vizualizare

Coordonate de modelare și coordonate de vizualizare

► **Coordonatele de modelare**

Coordonate de modelare și coordonate de vizualizare

► **Coordonatele de modelare**

- originea O

Coordonate de modelare și coordonate de vizualizare

► **Coordonatele de modelare**

- originea O
- axe de coordonate Ox, Oy, Oz cu versorii e_1, e_2, e_3

Coordonate de modelare și coordonate de vizualizare

► **Coordonatele de modelare**

- originea O
- axe de coordonate Ox, Oy, Oz cu vesorii e_1, e_2, e_3
- implicit, obiectele/primitivele (vârfurile) sunt indicate în raport cu acest sistem de coordonate

Coordonate de modelare și coordonate de vizualizare

► Coordonatele de modelare

- originea O
- axe de coordonate Ox, Oy, Oz cu vesorii e_1, e_2, e_3
- implicit, obiectele/primitivele (vârfurile) sunt indicate în raport cu acest sistem de coordonate

► Apelarea funcției `glm::lookAt()`; are ca efect (implicit) generarea unui nou reper / sistem de coordonate, numite **reper de vizualizare / coordonate de vizualizare**

Coordonate de modelare și coordonate de vizualizare

► Coordonatele de modelare

- originea O
- axe de coordonate Ox, Oy, Oz cu vesorii e_1, e_2, e_3
- implicit, obiectele/primitivele (vârfurile) sunt indicate în raport cu acest sistem de coordonate

► Apelarea funcției `glm::lookAt()`; are ca efect (implicit) generarea unui nou reper / sistem de coordonate, numite **reper de vizualizare / coordonate de vizualizare**

- originea: P_0 (poziția observatorului)

Coordonate de modelare și coordonate de vizualizare

► Coordonatele de modelare

- originea O
- axe de coordonate Ox, Oy, Oz cu vesorii e_1, e_2, e_3
- implicit, obiectele/primitivele (vârfurile) sunt indicate în raport cu acest sistem de coordonate

► Apelarea funcției `glm::lookAt()`; are ca efect (implicit) generarea unui nou reper / sistem de coordonate, numite **reper de vizualizare / coordonate de vizualizare**

- originea: P_0 (poziția observatorului)
- axe: date de vesorii u, v, n (construiți în continuare)

Functia glm::lookAt()

- ▶ Pentru a înțelege funcția `glm::lookAt()`; care sunt elementele geometrice relevante atunci când vorbim despre observarea unei scene 3D? (de exemplu vederea umană sau folosirea unui aparat fotografic / telefon mobil).

Functia glm::lookAt()

- ▶ Pentru a înțelege funcția `glm::lookAt()`; care sunt elementele geometrice relevante atunci când vorbim despre observarea unei scene 3D? (de exemplu vederea umană sau folosirea unui aparat fotografic / telefon mobil).
 - Poziția (coordonatele) observatorului

Functia glm::lookAt()

- ▶ Pentru a înțelege funcția `glm::lookAt()`; care sunt elementele geometrice relevante atunci când vorbim despre observarea unei scene 3D? (de exemplu vederea umană sau folosirea unui aparat fotografic / telefon mobil).
 - Poziția (coordonatele) observatorului
 - Direcția / Punctul de referință (spre care este îndreptată privirea sau dispozitivul)

Functia glm::lookAt()

- ▶ Pentru a înțelege funcția `glm::lookAt()`; care sunt elementele geometrice relevante atunci când vorbim despre observarea unei scene 3D? (de exemplu vederea umană sau folosirea unui aparat fotografic / telefon mobil).
 - Poziția (coordonatele) observatorului
 - Direcția / Punctul de referință (spre care este îndreptată privirea sau dispozitivul)
 - Orientarea

Funcția `glm::lookAt()`

- ▶ Pentru a înțelege funcția `glm::lookAt()`; care sunt elementele geometrice relevante atunci când vorbim despre observarea unei scene 3D? (de exemplu vederea umană sau folosirea unui aparat fotografic / telefon mobil).
 - Poziția (coordonatele) observatorului
 - Direcția / Punctul de referință (spre care este îndreptată privirea sau dispozitivul)
 - Orientarea
- ▶ Funcția `glm::lookAt`
`glm::lookAt (x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);`

Funcția `glm::lookAt()`

- ▶ Pentru a înțelege funcția `glm::lookAt()`; care sunt elementele geometrice relevante atunci când vorbim despre observarea unei scene 3D? (de exemplu vederea umană sau folosirea unui aparat fotografic / telefon mobil).
 - Poziția (coordonatele) observatorului
 - Direcția / Punctul de referință (spre care este îndreptată privirea sau dispozitivul)
 - Orientarea
- ▶ Funcția `glm::lookAt`
`glm::lookAt (x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);`
 - (x_0, y_0, z_0) : coordonatele observatorului P_0 în reperul de modelare;

Funcția `glm::lookAt()`

- ▶ Pentru a înțelege funcția `glm::lookAt()`; care sunt elementele geometrice relevante atunci când vorbim despre observarea unei scene 3D? (de exemplu vederea umană sau folosirea unui aparat fotografic / telefon mobil).
 - Poziția (coordonatele) observatorului
 - Direcția / Punctul de referință (spre care este îndreptată privirea sau dispozitivul)
 - Orientarea
- ▶ Funcția `glm::lookAt`
`glm::lookAt (x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);`
 - (x_0, y_0, z_0) : coordonatele observatorului P_0 în reperul de modelare;
 - $(x_{ref}, y_{ref}, z_{ref})$: coordonatele unui punct de referință P_{ref} spre care se uită observatorul;

Funcția `glm::lookAt()`

- ▶ Pentru a înțelege funcția `glm::lookAt()`; care sunt elementele geometrice relevante atunci când vorbim despre observarea unei scene 3D? (de exemplu vederea umană sau folosirea unui aparat fotografic / telefon mobil).
 - Poziția (coordonatele) observatorului
 - Direcția / Punctul de referință (spre care este îndreptată privirea sau dispozitivul)
 - Orientarea
- ▶ Funcția `glm::lookAt`
`glm::lookAt (x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);`
 - (x_0, y_0, z_0) : coordonatele observatorului P_0 în reperul de modelare;
 - $(x_{ref}, y_{ref}, z_{ref})$: coordonatele unui punct de referință P_{ref} spre care se uită observatorul;
 - (V_x, V_y, V_z) : vector care indică verticala din planul de vizualizare

Funcția `glm::lookAt()`

- ▶ Pentru a înțelege funcția `glm::lookAt()`; care sunt elementele geometrice relevante atunci când vorbim despre observarea unei scene 3D? (de exemplu vederea umană sau folosirea unui aparat fotografic / telefon mobil).
 - Poziția (coordonatele) observatorului
 - Direcția / Punctul de referință (spre care este îndreptată privirea sau dispozitivul)
 - Orientarea
- ▶ Funcția `glm::lookAt`
`glm::lookAt (x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);`
 - (x_0, y_0, z_0) : coordonatele observatorului P_0 în reperul de modelare;
 - $(x_{ref}, y_{ref}, z_{ref})$: coordonatele unui punct de referință P_{ref} spre care se uită observatorul;
 - (V_x, V_y, V_z) : vector care indică verticala din planul de vizualizare
- ▶ Implicit: $P_0 = (0, 0, 0)$, $P_{ref} = (0, 0 - 1)$, $V = (0, 1, 0)$

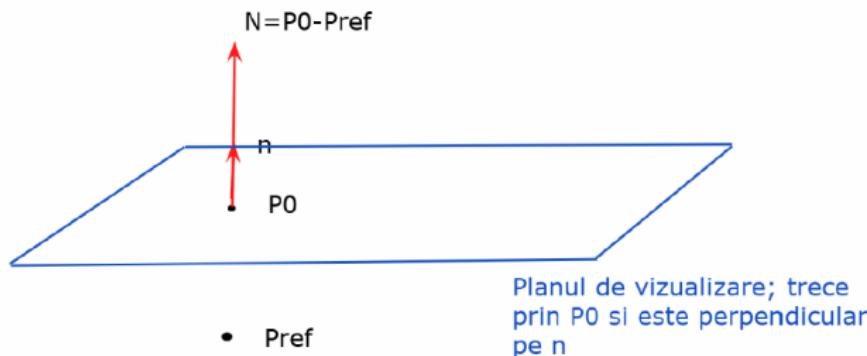
Funcția `glm::lookAt()`

- ▶ Pentru a înțelege funcția `glm::lookAt()`; care sunt elementele geometrice relevante atunci când vorbim despre observarea unei scene 3D? (de exemplu vederea umană sau folosirea unui aparat fotografic / telefon mobil).
 - Poziția (coordonatele) observatorului
 - Direcția / Punctul de referință (spre care este îndreptată privirea sau dispozitivul)
 - Orientarea
- ▶ Funcția `glm::lookAt`
`glm::lookAt (x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);`
 - (x_0, y_0, z_0) : coordonatele observatorului P_0 în reperul de modelare;
 - $(x_{ref}, y_{ref}, z_{ref})$: coordonatele unui punct de referință P_{ref} spre care se uită observatorul;
 - (V_x, V_y, V_z) : vector care indică verticala din planul de vizualizare
- ▶ Implicit: $P_0 = (0, 0, 0)$, $P_{ref} = (0, 0 - 1)$, $V = (0, 1, 0)$
- ▶ În continuare: construirea reperului de vizualizare pornind de la argumentele funcției `glm::lookAt()`;

Funcția `glm::lookAt()`

- ▶ Pentru a înțelege funcția `glm::lookAt()`; care sunt elementele geometrice relevante atunci când vorbim despre observarea unei scene 3D? (de exemplu vederea umană sau folosirea unui aparat fotografic / telefon mobil).
 - Poziția (coordonatele) observatorului
 - Direcția / Punctul de referință (spre care este îndreptată privirea sau dispozitivul)
 - Orientarea
 - ▶ Funcția `glm::lookAt`
- `glm::lookAt (x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);`
- (x_0, y_0, z_0) : coordonatele observatorului P_0 în reperul de modelare;
 - $(x_{ref}, y_{ref}, z_{ref})$: coordonatele unui punct de referință P_{ref} spre care se uită observatorul;
 - (V_x, V_y, V_z) : vector care indică verticala din planul de vizualizare
- ▶ Implicit: $P_0 = (0, 0, 0)$, $P_{ref} = (0, 0 - 1)$, $V = (0, 1, 0)$
 - ▶ În continuare: construirea reperului de vizualizare pornind de la argumentele funcției `glm::lookAt()`;
 - ▶ Originea reperului: $P_0 = (x_0, y_0, z_0)$; axele date de u, v, n

Reperul de vizualizare - vectorul n



$$N = \overrightarrow{P_{\text{ref}} P_0} = P_0 - P_{\text{ref}}$$

de ce $P_0 - \text{Pref}$ și nu
 $\text{Pref} - P_0$?

$n = \frac{N}{\|N\|}$

de ce se
împarte la
 $\|N\|$?

Reperul de vizualizare - vectorii v , u

- În planul de vizualizare sunt definiți doi vectori u și v care sunt vectorii primelor două axe ale reperului de vizualizare (“orizontală” și “verticală” din planul de vizualizare).

Reperul de vizualizare - vectorii v, u

- ▶ În planul de vizualizare sunt definiți doi vectori u și v care sunt vectorii primelor două axe ale reperului de vizualizare (“orizontală” și “verticală” din planul de vizualizare).
- ▶ **primul versor u** - direcționează orizontală din planul de vizualizare: este perpendicular pe vectorul n (ca să fie inclus în planul de vizualizare) și este perpendicular pe vectorul V indicat în gluLookAt

$$u = \frac{V \times n}{\|V\|}$$

Reperul de vizualizare - vectorii v , u

- ▶ În planul de vizualizare sunt definiți doi vectori u și v care sunt vectorii primelor două axe ale reperului de vizualizare ("orizontală" și "verticală" din planul de vizualizare).
- ▶ **primul versor u** - direcționează orizontală din planul de vizualizare: este perpendicular pe vectorul n (ca să fie inclus în planul de vizualizare) și este perpendicular pe vectorul V indicat în gluLookAt

$$u = \frac{V \times n}{\|V\|}$$

- ▶ **al doilea versor v** - verticală "reală" din planul de vizualizare

$$v = n \times u$$

Legătura dintre vectorii V și v

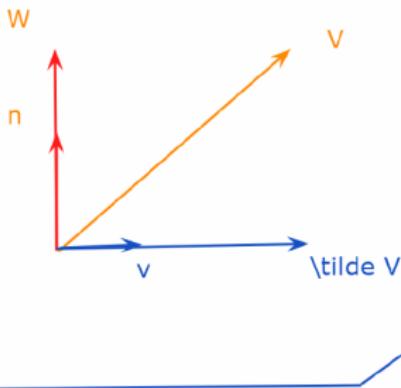
Comentariu/Întrebare: ce legătură există între vectorul V , indicat ca “verticală” în funcția `glm::lookAt()`; și vectorul v , calculat ca fiind al doilea versor al reperului de vizualizare?

Legătura dintre vectorii V și v

Comentariu/Întrebare: ce legătură există între vectorul V, indicat ca "verticală" în funcția `glm::lookAt()`; și vectorul v, calculat ca fiind al doilea versor al reperului de vizualizare?

R: Vectorul V se descompune ca suma dintre un vector \tilde{V} (=proiecția lui V pe planul de vizualizare) și un vector W , perpendicular pe planul de vizualizare (coliniar cu n). Are loc relația $v = \frac{\tilde{v}}{\|\tilde{v}\|}$.

Dacă modificăm vectorul V , adăugând multiplii ai lui N sau n , rezultă un alt vector v și se modifică.



$$V = \tilde{V} + W$$

$W \rightarrow$ volumar
symm.

$W \rightarrow$ whinian
cen

cu n

an n

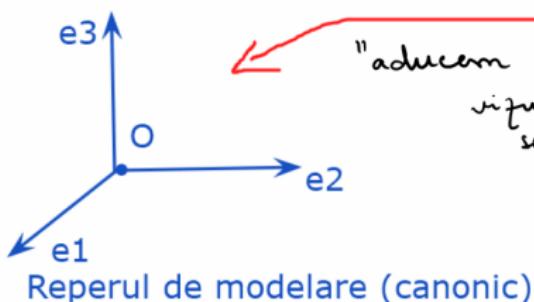
1

3

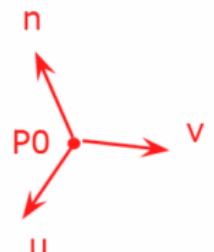
Mudan

၁၃၁

Schimbarea reperului ca transformare



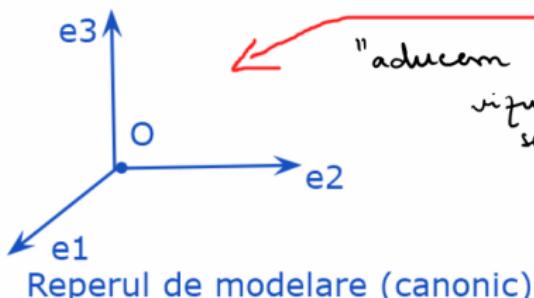
"aducem reperul de vizualizare ca să se suprapună pe reperul de modelare"



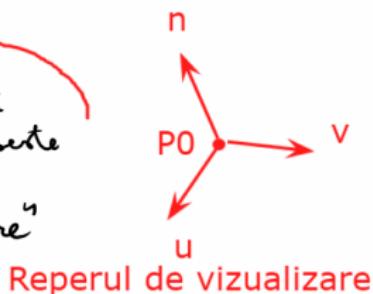
Descrierea transformărilor:

- translatăm a.î. P_0 să devină originea, adică aplicăm $T_{(-x_0, -y_0, -z_0)}$ vectorii sunt $\perp 2 \times 2$, cu normă 1
- roatare 3D a.î. reperul ortonormalat (u, v, n) să se suprapună cu reperul ON (e_1, e_2, e_3)

Schimbarea reperului ca transformare



"aducem reperul de vizualizare ca să se suprapună pe reperul de modelare"



Descrierea transformărilor:

- translatăm a.î. P_0 să devină originea, adică aplicăm $T_{(-x_0, -y_0, -z_0)}$ vectorii sunt $\perp 2 \times 2$, cu normă 1
- roatare 3D a.î. reperul ortonormalat (u, v, n) să se suprapună cu reperul $ON (e_1, e_2, e_3)$

Care este matricea asociată?

Matricea asociată schimbării de reper

Matricele asociate celor două transformări:

$$\textcircled{1} \quad \mathbb{T}_{(-x_0, -y_0, -z_0)} \rightarrow M_{\mathbb{T}_{(-x_0, -y_0, -z_0)}} = \begin{pmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

\textcircled{2} rotatia ? Pentru început: matricea 3×3 care transformă (e_1, e_2, e_3) în reperul ON (u, v, n).

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

Coloanele acestei matrice sunt componentele lui u, v, n în reperul canonic.

Matricea asociată schimbării de reper

Matricea A^{-1} este cea care transformă reperul $(u, v, n)^T$ în reperul canonic.

Obs. $A^t \cdot A = \begin{pmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ n_x & n_y & n_z \end{pmatrix} \cdot \begin{pmatrix} u_x & v_x & n_x \\ u_y & v_y & n_y \\ u_z & v_z & n_z \end{pmatrix} =$

$$= \mathbb{I}_3 \text{ (deoarece } (u, v, n) \text{ este reper ON)}$$

Deri: Întrucât (u, v, n) este reper ON, matricea A verifică relația $A^t \cdot A = \mathbb{I}_3$ (A s.n. ortogonală) și matricea inversă este $A^{-1} = A^t$.

Matricea asociată schimbării de reper

În final, matricea 4×4 asociată este:

$$M = A^+ \cdot M_{\text{II}} = \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \Rightarrow$$

$$\Rightarrow M = \begin{pmatrix} u_x & u_y & u_z & -\langle u, P_0 \rangle \\ v_x & v_y & v_z & -\langle v, P_0 \rangle \\ n_x & n_y & n_z & -\langle n, P_0 \rangle \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

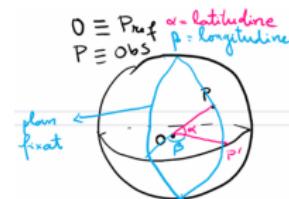
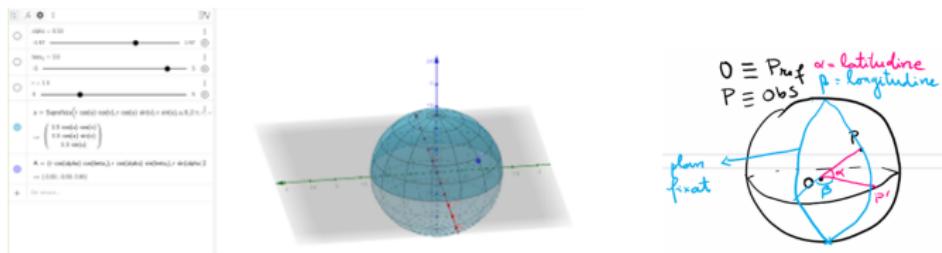
Survolarea unui obiect - codul 07_02_survolare_cub.cpp

- ▶ La ce revine a survola un obiect?

Survolarea unui obiect - codul 07_02_survolare_cub.cpp

- ▶ La ce revine a survola un obiect?
- ▶ Reprezentarea sferei de centru C și rază r

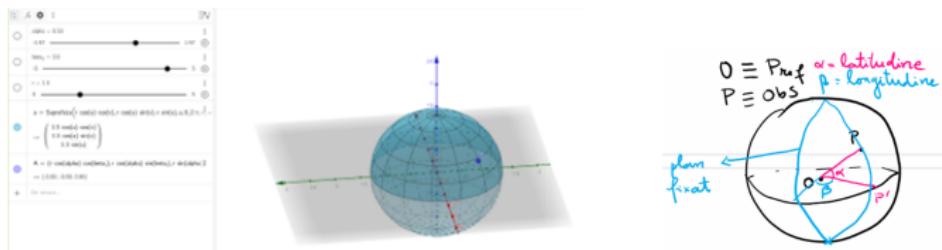
$$\begin{cases} x = C_x + r \cos(\alpha) \cos(\beta) \\ y = C_y + r \cos(\alpha) \sin(\beta) \\ z = C_z + r \sin(\alpha) \end{cases} \quad \alpha \in [-\frac{\pi}{2}, \frac{\pi}{2}], \beta \in [0, 2\pi]$$



Survolarea unui obiect - codul 07_02_survolare_cub.cpp

- ▶ La ce revine a survola un obiect?
- ▶ Reprezentarea sferei de centru C și rază r

$$\begin{cases} x = C_x + r \cos(\alpha) \cos(\beta) \\ y = C_y + r \cos(\alpha) \sin(\beta) \\ z = C_z + r \sin(\alpha) \end{cases} \quad \alpha \in [-\frac{\pi}{2}, \frac{\pi}{2}], \beta \in [0, 2\pi]$$



- ▶ Pentru a implementa survolarea, observatorul Obs se deplasează pe o sferă cu centrul în punctul de referință Ref și cu raza dist. În cod dist, alpha, beta sunt variabile.

```
//pozitia observatorului - se deplaseaza pe sfera
Obsx = Refx + dist * cos(alpha) * cos(beta);
Obsy = Refy + dist * cos(alpha) * sin(beta);
Obsz = Refz + dist * sin(alpha);
```

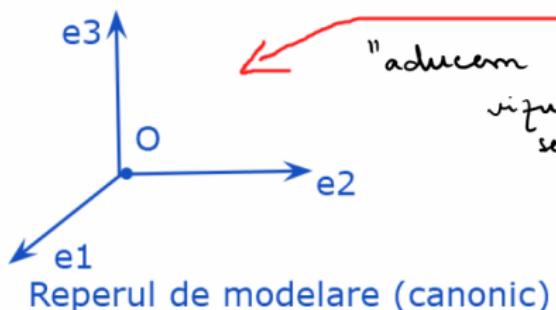
Transformări (IV). Proiecții

Mihai-Sorin Stupariu

Sem. I, 2022 - 2023

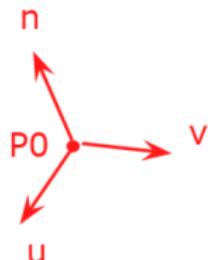
Schimbarea reperului ca transformare

Schimbarea de reper \leftrightarrow Efectuarea unei transformări



Reperul de modelare (canonic)

"aducem reperul de vizualizare ca să se suprapună peste reperul de modelare"



Reperul de vizualizare

Descrierea transformărilor:

- translatăm a.î. P_0 să devină originea, adică aplicăm $T_{(-x_0, -y_0, -z_0)}$ vectorii sunt $1 \times 2 \times 2$, cu normă 1
- roatare 3D a.î. reperul ortonormal (u, v, n) să se suprapună cu reperul $ON (e_1, e_2, e_3)$

- Rolul transformărilor de proiecție: de a permite reprezentarea unei lumi 3D, teoretic nemărginite, pe un monitor 2D mărginit.

- ▶ Rolul transformărilor de proiecție: de a permite reprezentarea unei lumi 3D, teoretic nemărginite, pe un monitor 2D mărginit.
- ▶ Despre aplicarea proiecțiilor:

- ▶ Rolul transformărilor de proiecție: de a permite reprezentarea unei lumi 3D, teoretic nemărginite, pe un monitor 2D mărginit.
- ▶ Despre aplicarea proiecțiilor:
 - dacă nu a fost efectuată nicio transformare de modelare, proiecția este aplicată în raport cu reperul de modelare, fiind decupat pătratul "standard" $[-1, 1] \times [-1, 1]$,

- ▶ **Rolul transformărilor de proiecție:** de a permite reprezentarea unei lumi 3D, teoretic nemărginite, pe un monitor 2D mărginit.
- ▶ Despre aplicarea proiecțiilor:
 - dacă nu a fost efectuată nicio transformare de modelare, proiecția este aplicată în raport cu reperul de modelare, fiind decupat pătratul "standard" $[-1, 1] \times [-1, 1]$,
 - dacă a fost efectuată o transformare de vizualizare (observatorul a fost "adus" în origine, axele au fost "aliniate", etc.): din punctul de vedere al logicii imaginii, decuparea / proiecția sunt realizate în raport cu observatorul și reperul de vizualizare. Altfel spus, proiecția este aplicată după transformarea de vizualizare.

- ▶ Rolul transformărilor de proiecție: de a permite reprezentarea unei lumi 3D, teoretic nemărginite, pe un monitor 2D mărginit.
- ▶ Despre aplicarea proiecțiilor:
 - dacă nu a fost efectuată nicio transformare de modelare, proiecția este aplicată în raport cu reperul de modelare, fiind decupat pătratul "standard" $[-1, 1] \times [-1, 1]$,
 - dacă a fost efectuată o transformare de vizualizare (observatorul a fost "adus" în origine, axele au fost "aliniate", etc.): din punctul de vedere al logicii imaginii, decuparea / proiecția sunt realizate în raport cu observatorul și reperul de vizualizare. Altfel spus, proiecția este aplicată după transformarea de vizualizare.
- ▶ O proiecție este o transformare care implică (i) decuparea, (ii) proiecția propriu-zisă, fiind necesară o matrice 4×4 adecvată. Din punct de vedere al implementării: dacă `matVis` este matricea de vizualizare (dată de `glm::lookAt()`) și `matPr` este matricea de proiecție, atunci în codul sursă trebuie să apară `matPr * matVis`.

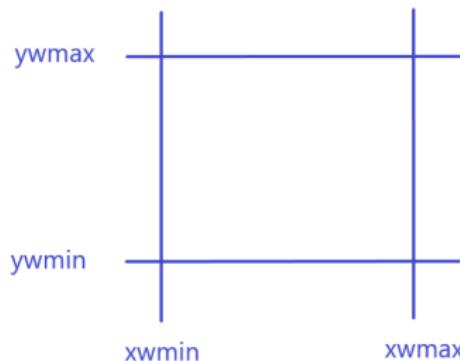
Cazul 2D

► `glm::ortho (xwmin, xwmax, ywmin, ywmax);`

Cazul 2D

- ▶ `glm::ortho (xwmin, xwmax, ywmin, ywmax);`
- ▶ Efectul: este decupat un dreptunghi \mathcal{D} din planul orizontal Oxy (se presupune că nu au fost aplicate alte transformări). Dreptunghiul \mathcal{D} are laturile paralele cu axele de coordonate, fiind delimitat de dreptele

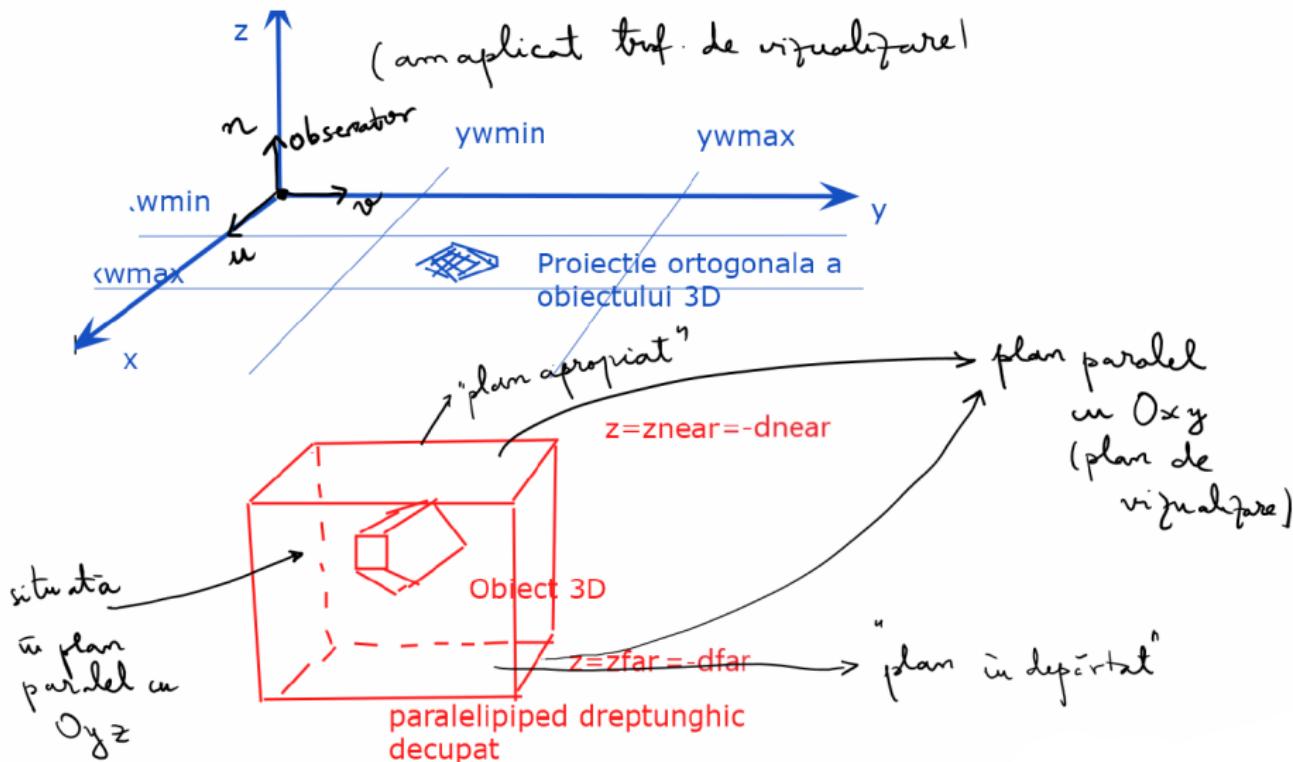
$$x = xwmin, \quad x = xwmax, \quad y = ywmin, \quad y = ywmax.$$



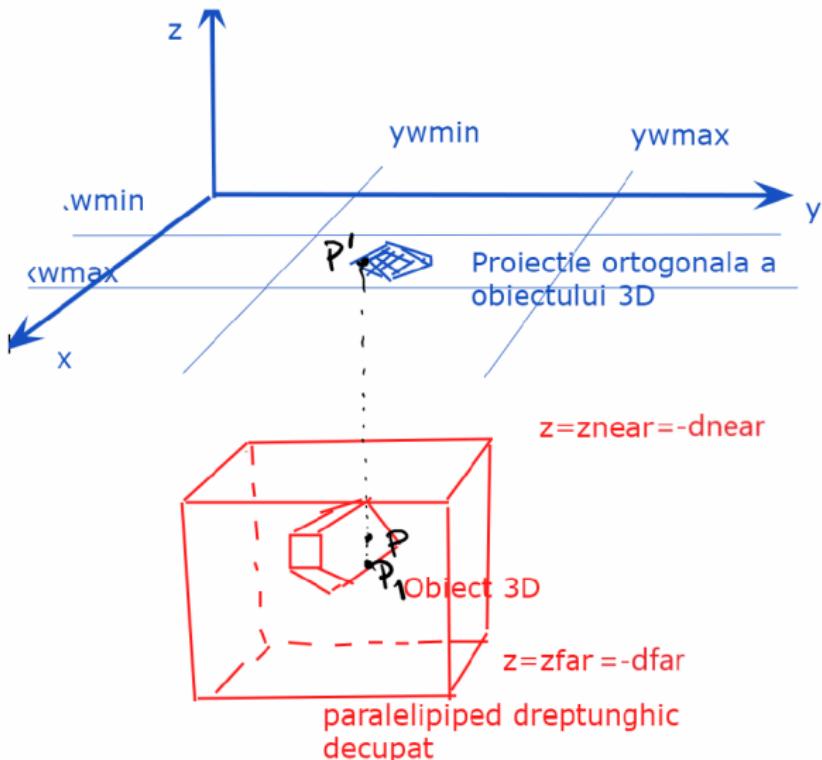
Apoi este realizată o transformare a dreptunghiului \mathcal{D} în pătratul "standard" $[-1, 1] \times [-1, 1]$. Matricea 4×4 asociată transformării poate fi determinată explicit.

Cazul 3D - proiecții ortogonale

`glm::ortho (xwmin, xwmax, ywmin, ywmax, dnear, dfar);`



Ce este o proiecție ortogonală?



P' este în pr.
ortogonală lui P_1 .

P' = proiecția
ortogonală
a lui P
(prin P ducem
• dreptă
pe P' pe Oxy,
apoi intersecția
cu Oxy)

Cazul 3D - proiecții ortogonale

Matricea 4×4 asociată este

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

Această matrice are rolul de a transforma un **paralelipiped dreptunghic** decupat în paralelipipedul “standard” $[-1, 1] \times [-1, 1] \times [-1, 1]$, apoi, în mod implicit, sunt reținute primele două coordonate.

Cazul 3D - proiecții perspective

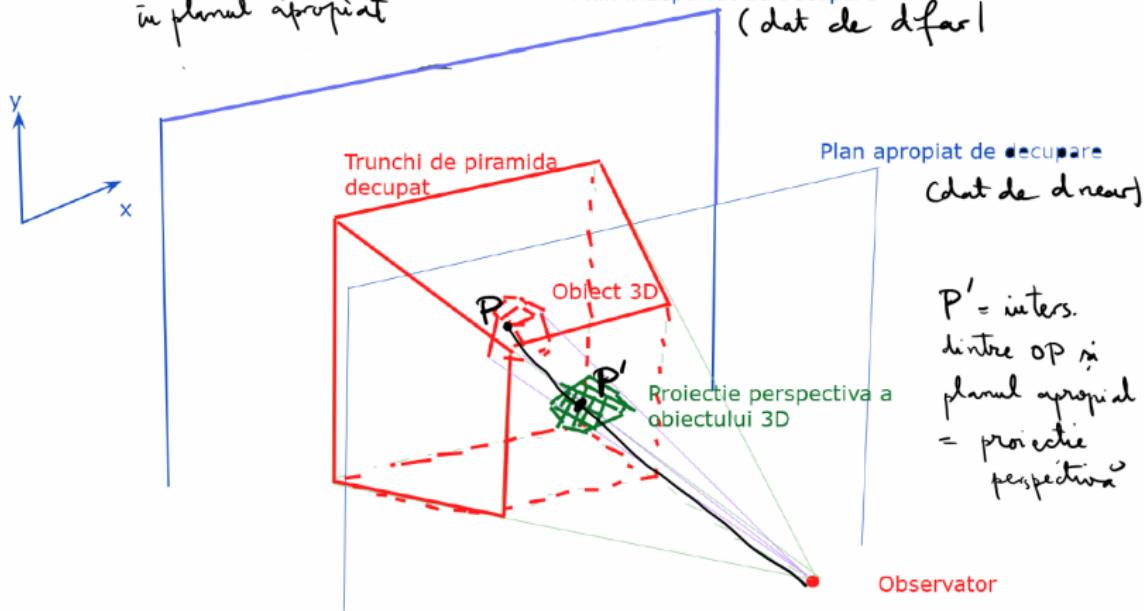
`glm::frustum()` - este decupat un trunchi de piramidă.

```
glm::frustum(xwmin, xwmax, ywmin, ywmax, dnear, dfar);
```

in planet apropiat

Plan îndepărtat de decupare

(dat de dfar)



Cazul 3D - proiecții perspective. Valoarea *dnear* și efectul asupra desenului în cazul *glFrustum*

Presupunem că nu modificăm valoarea *xwmin*, *xwmax*, *ywmin*, *ywmax*. Dacă *dnear* este mic, atunci trunchiul de piramidă decupat are “deschidere mai mare” și obiectele vor părea mai mici (stânga). Dacă *dnear* este mare, atunci trunchiul de piramidă decupat este “mai îngust” și obiectele vor părea mai mari (dreapta).

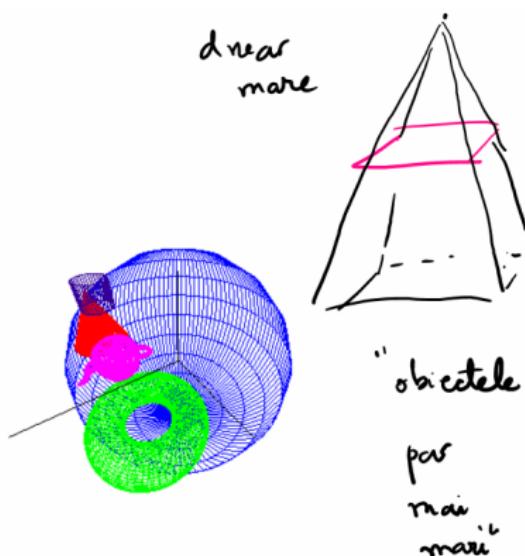
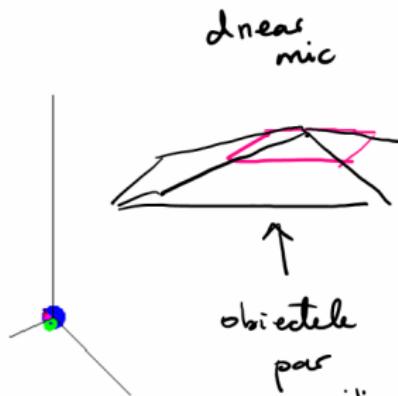
Desenarea unui cub folosind testul de adâncime



Desenarea unui cub folosind testul de adâncime

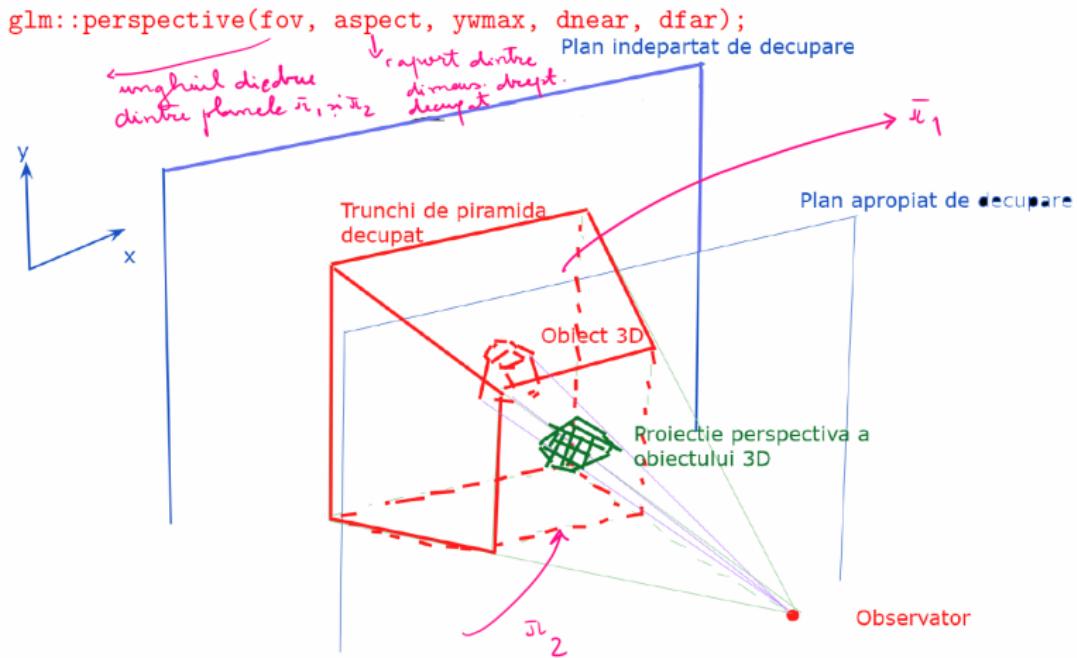


Cazul 3D - proiecții perspective. Valoarea d_{near} și efectul asupra desenului în cazul glFrustum - alt exemplu

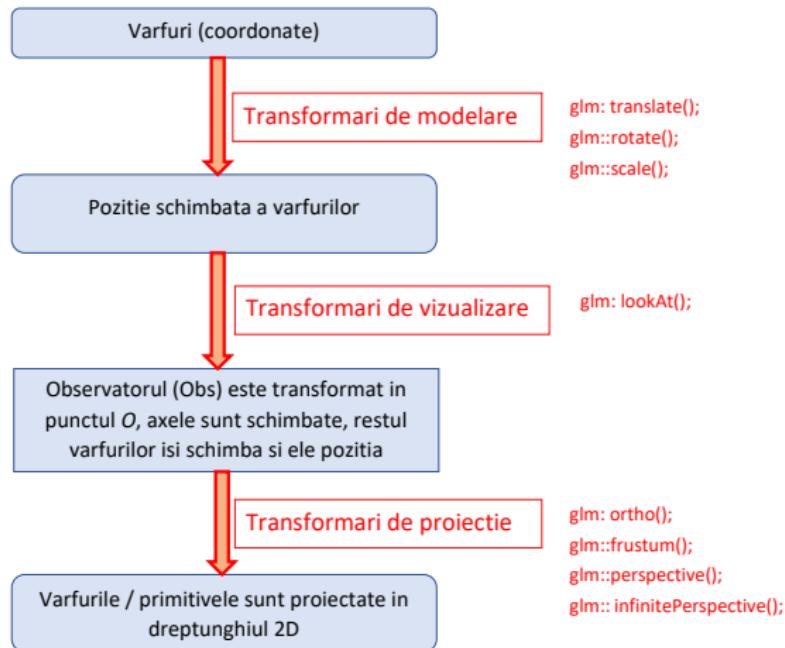


Cazul 3D - proiecții perspective

`glm::perspective()` - este decupat un trunchi de piramidă decupat dintr-o piramidă în care înălțimea dusă din vârful piramidei inițiale (observator) cade în centrul dreptunghiului. "Deschiderea" piramidei este dată de `fov`.



Concluzie - fluxul transformărilor

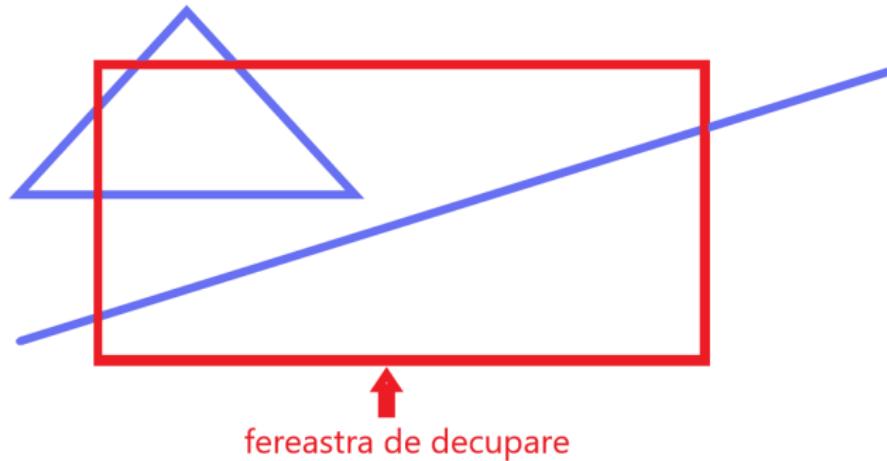


Atenție la ordinea înmulțirii matricelor corespunzătoare din shader:

matrProiectie * matrVizualizare * matrModelare.

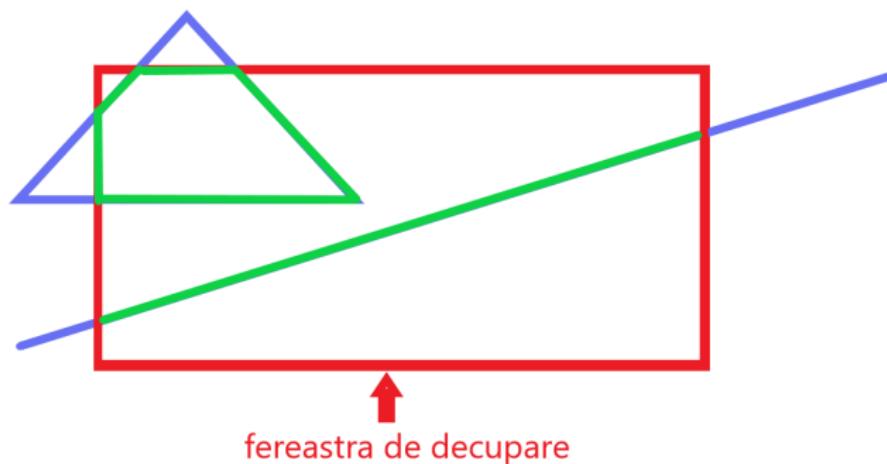
Algoritmi de decupare - motivație

În cazul primitivelor din desen, doar o parte a acestora intersectează fereastra de decupare și urmează să fie randate.



Algoritmi de decupare - motivație

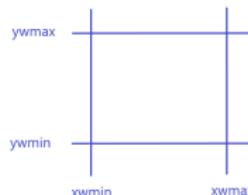
Pentru segment va fi randată doar o porțiune. În cazul triunghiului, va fi decupată o porțiune a sa, fiind randat un pentagon.



Algoritmi de decupare - observații generale

- ▶ Trebuie stabilit dacă o primitivă geometrică intersectează sau nu fereastra de decupare. În continuare presupunem că suntem în cazul 2D, fereastra de decupare fiind delimitată de dreptele

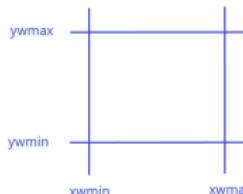
$$x = xwmin, \quad x = xwmax, \quad y = ywmin, \quad y = ywmax.$$



Algoritmi de decupare - observații generale

- ▶ Trebuie stabilit dacă o primitivă geometrică intersectează sau nu fereastra de decupare. În continuare presupunem că suntem în cazul 2D, fereastra de decupare fiind delimitată de dreptele

$$x = xwmin, \quad x = xwmax, \quad y = ywmin, \quad y = ywmax.$$

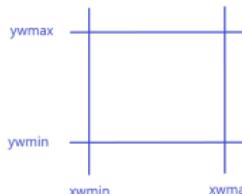


- ▶ Algoritmii de decupare depind de tipul primitivei. În cazul punctelor: se bazează pe inegalități.

Algoritmi de decupare - observații generale

- ▶ Trebuie stabilit dacă o primitivă geometrică intersectează sau nu fereastra de decupare. În continuare presupunem că suntem în cazul 2D, fereastra de decupare fiind delimitată de dreptele

$$x = xwmin, \quad x = xwmax, \quad y = ywmin, \quad y = ywmax.$$

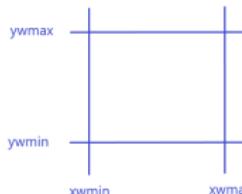


- ▶ Algoritmii de decupare depind de tipul primitivei. În cazul punctelor: se bazează pe inegalități.
- ▶ În continuare: algoritmi pentru segmente de dreaptă (pornind de la aceștia se poate ajunge la algoritmi pentru poligoane). Sunt descriși doi algoritmi: (i) Cohen-Sutherland (calitativ), (ii) **Liang-Barski** (cantitativ).

Algoritmi de decupare - observații generale

- ▶ Trebuie stabilit dacă o primitivă geometrică intersectează sau nu fereastra de decupare. În continuare presupunem că suntem în cazul 2D, fereastra de decupare fiind delimitată de dreptele

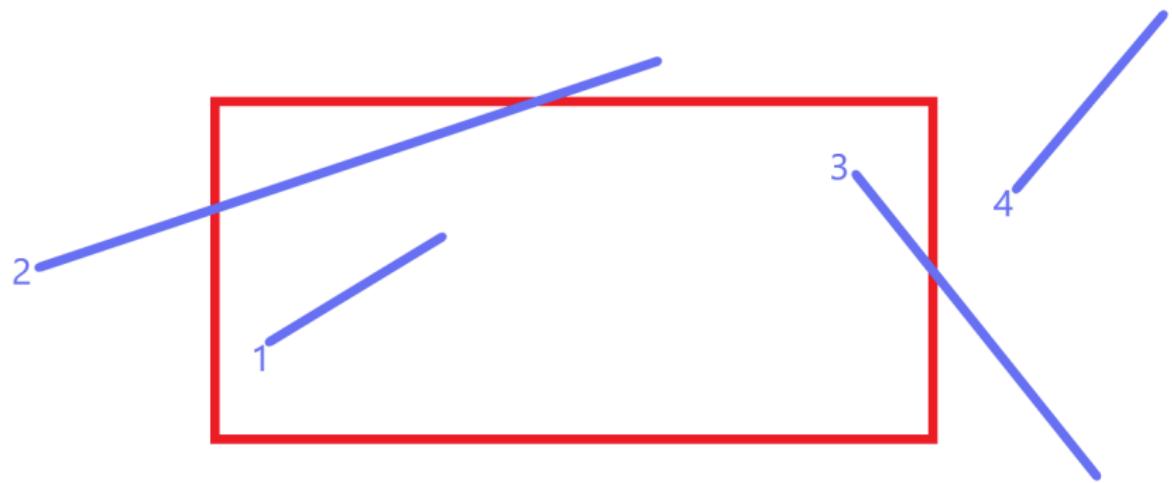
$$x = xwmin, \quad x = xwmax, \quad y = ywmin, \quad y = ywmax.$$



- ▶ Algoritmii de decupare depind de tipul primitivei. În cazul punctelor: se bazează pe inegalități.
- ▶ În continuare: algoritmi pentru segmente de dreaptă (pornind de la aceștia se poate ajunge la algoritmi pentru poligoane). Sunt descriși doi algoritmi: (i) Cohen-Sutherland (calitativ), (ii) Liang-Barski (cantitativ).
- ▶ Alte metode / referințe se găsesc în [Line Clipping in 2D: Overview, Techniques and Algorithms, 2022](#)

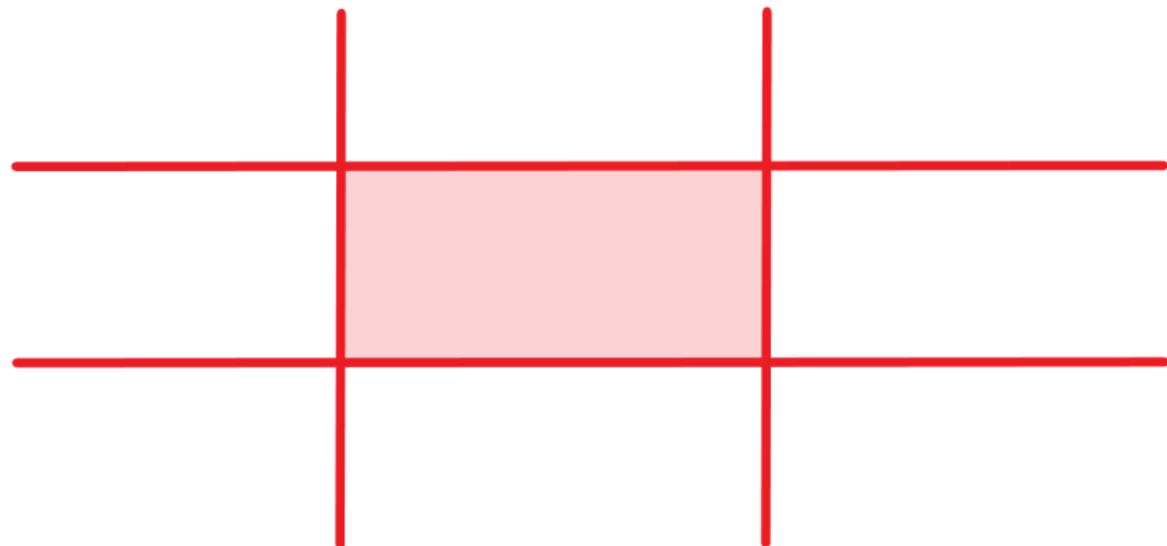
Abordare calitativă. Algoritmul Cohen-Sutherland

Diverse configurații ale segmentelor față de fereastra de decupare.



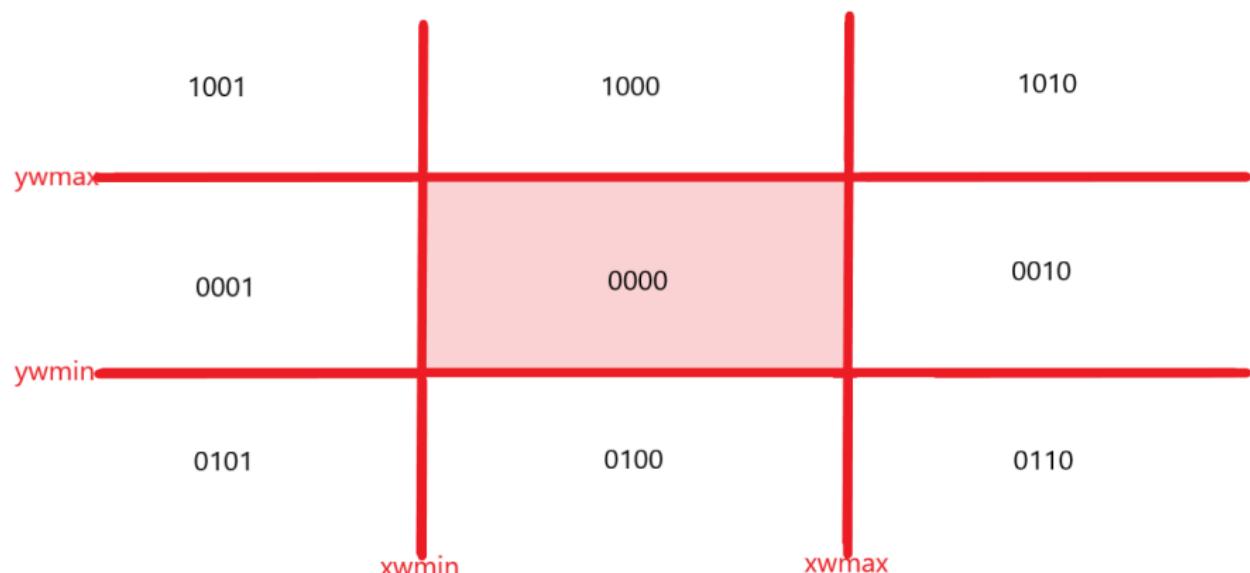
Algoritmul Cohen-Sutherland - împărțirea planului

Planul este împărțit în 9 regiuni, fiecareia dintre ele îi este asociat un cod pe 4 biți (**TBRL – Top Bottom Right Left**).



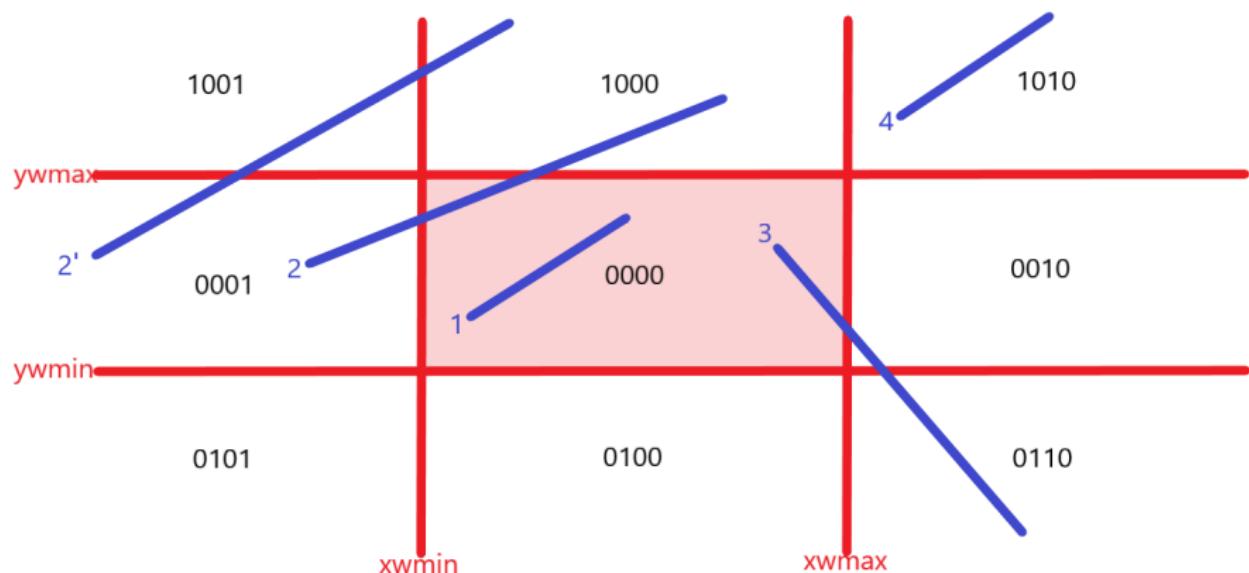
Algoritmul Cohen-Sutherland - codurile regiunilor

Planul este împărțit în 9 regiuni, fiecareia dintre ele îi este asociat un cod pe 4 biți (TBRL – Top Bottom Right Left).



Algoritmul Cohen-Sutherland - coduri și segmente

Dat un segment s , codurile asociate extremităților sale dă o primă informație despre poziția segmentului s față de fereastra de decupare.



Algoritmul Cohen-Sutherland - sinteză

- ▶ Pentru un punct $M = (x_M, y_M)$ se stabilește codul regiunii:
 $x < xwmin \Rightarrow L = 1, R = 0$, deci **01
 $x > xwmax \Rightarrow L = 0, R = 1$, deci **10, etc

Algoritmul Cohen-Sutherland - sinteză

- ▶ Pentru un punct $M = (x_M, y_M)$ se stabilește codul regiunii:
 $x < xwmin \Rightarrow L = 1, R = 0$, deci **01
 $x > xwmax \Rightarrow L = 0, R = 1$, deci **10, etc
- ▶ Date două puncte, P și Q , cele două coduri asociate dău o primă informație referitoare la poziția segmentului $[PQ]$ față de fereastra de decupare.

Algoritmul Cohen-Sutherland - sinteză

- ▶ Pentru un punct $M = (x_M, y_M)$ se stabilește codul regiunii:
 $x < xwmin \Rightarrow L = 1, R = 0$, deci **01
 $x > xwmax \Rightarrow L = 0, R = 1$, deci **10, etc
- ▶ Date două puncte, P și Q , cele două coduri asociate dău o primă informație referitoare la poziția segmentului $[PQ]$ față de fereastra de decupare.
- ▶ Există cazuri în care folosind codurile se poate lua o decizie referitoare la poziția relativă a segmentului față de fereastră (de exemplu două coduri de tipul **10, ambele coduri 0000, etc.).
- ▶ Există cazuri în care folosind codurile nu se poate lua o decizie (de exemplu 0001 și 1000), fiind necesari alți algoritmi, cantitativi (se determină explicit coordonatele intersecțiilor dintre segment și dreptele suport ale ferestrei de decupare, se stabilește dacă aceste puncte sunt pe laturile dreptunghiului, etc.).

Abordare cantitativă. Algoritmul Liang-Barsky

- ▶ **Principiu:** Utilizează **reprezentarea parametrică** a unei drepte.
Idea centrală: fiecărui punct de pe dreaptă îi corespunde exact un număr real (parametru) și reciproc.

Abordare cantitativă. Algoritmul Liang-Barsky

- ▶ **Principiu:** Utilizează **reprezentarea parametrică** a unei drepte.
Idea centrală: fiecărui punct de pe dreaptă îi corespunde exact un număr real (parametru) și reciproc.
- ▶ **Explicit:** Fie $P_0 = (x_0, y_0), P_1 = (x_1, y_1) \in \mathbb{R}^2$, presupunem (fără a restrânge generalitatea) că $x_0 < x_1, y_0 < y_1$.
Notăm $\Delta x = x_1 - x_0, \Delta y = y_1 - y_0$.
Dreapta P_0P_1 are reprezentarea parametrică

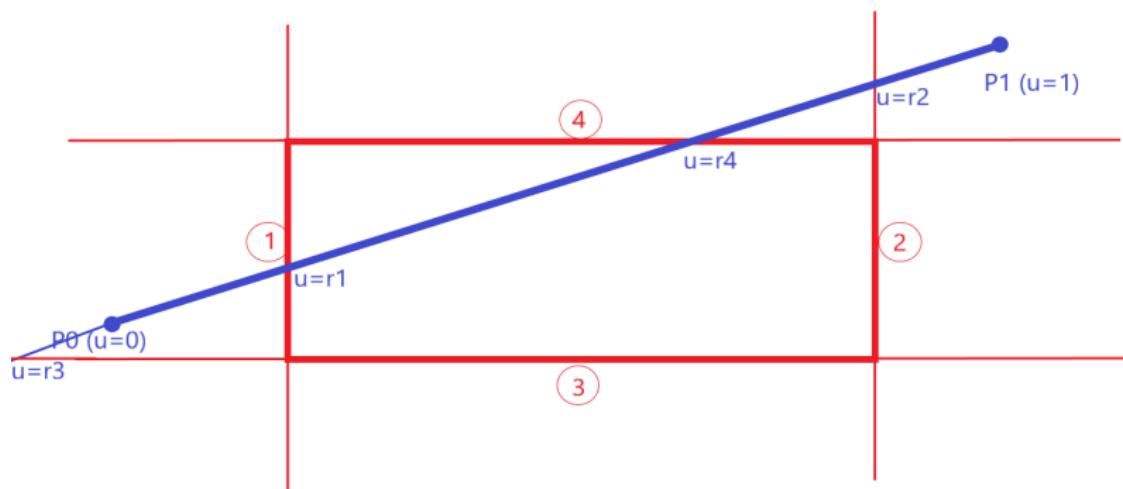
$$\begin{cases} x = x_0 + u \cdot \Delta x \\ y = y_0 + u \cdot \Delta y \end{cases}, u \in \mathbb{R}.$$

A da un punct de pe dreapta P_0P_1 revine la a indica o valoare a lui u și reciproc.

Algoritmul Liang-Barsky - intersecții cu fereastra

Intersecții cu fereastra de decupare: Dreapta P_0P_1 intersectează dreptele suport ale laturilor dreptunghiului (notate 1, 2, 3, 4) în puncte care corespund unor valori r_1, r_2, r_3, r_4 ale parametrului (calculabile explicit!). De exemplu, r_1 se determină din condiția

$$xwmin = x_0 + r_1 \cdot \Delta x, \text{ deci } r_1 = \frac{xwmin - x_0}{\Delta x}, \text{ etc.}$$



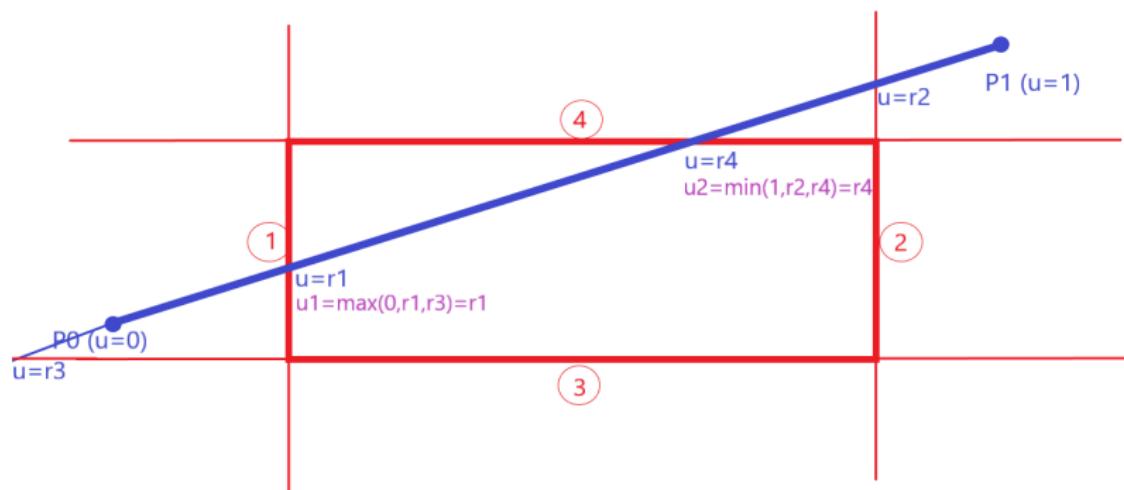
Algoritmul Liang-Barsky - valori importante ale parametrului

"Intrare" în dreptunghi: $u_1 = \max(0, r_1, r_3)$

(start segment, fâșia verticală, fâșia orizontală)

"Ieșire" din dreptunghi: $u_2 = \min(1, r_2, r_4)$

(stop segment, fâșia verticală, fâșia orizontală)



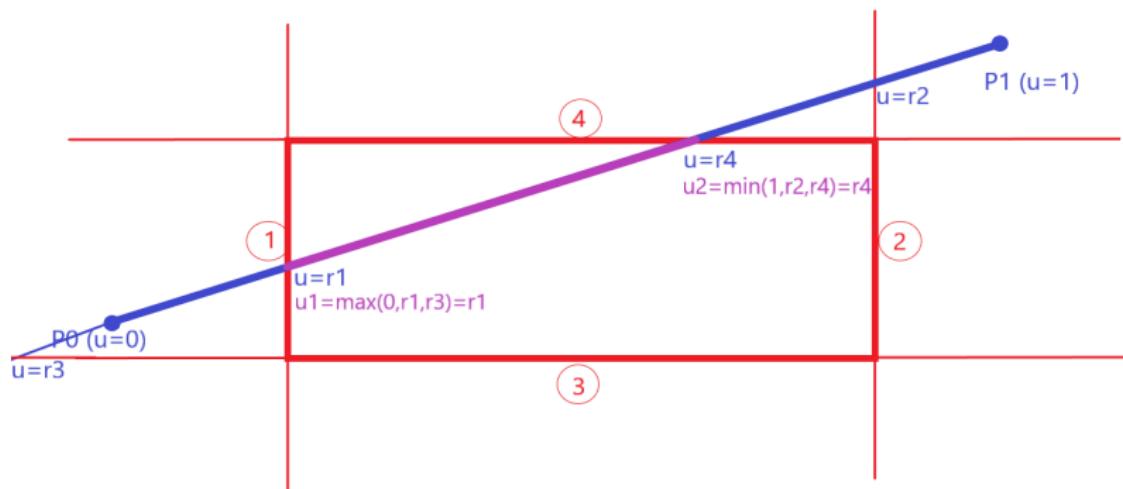
Algoritmul Liang-Barsky - condiția de intersecție

Condiția ca **segmentul** $[P_0P_1]$ să intersecteze dreptunghiul de decupare:

$$u_1 < u_2$$

(intuitiv: "intrăm în dreptunghiul de decupare înainte de a ieși").

Mai mult, segmentul care urmează să fie decupat are extremitățile corespunzătoare parametrilor u_1 și u_2 .



Grafică 3D

Mihai-Sorin Stupariu

Sem. I, 2022 - 2023

Obiecte 3D - scurte repere istorice și resurse

Poliedre

Suprafețe implicate

Curbe parametrizate

Suprafețe parametrizate

Suprafețe parametrizate - suprafețe de rotație

Scurt istoric (i)

“The Utah teapot” - 1975. Consultați și schița ceainicului realizată de Martin Newell. În biblioteca glut sunt implementate (și pot fi folosite în OpenGL “vechi”) funcții dedicate (e.g. glutWireTeapot)



Sursa: Wikipedia, imagine încărcată de Marshall Astor (<http://www.marshallastor.com/>)

Scurt istoric, resurse (ii)

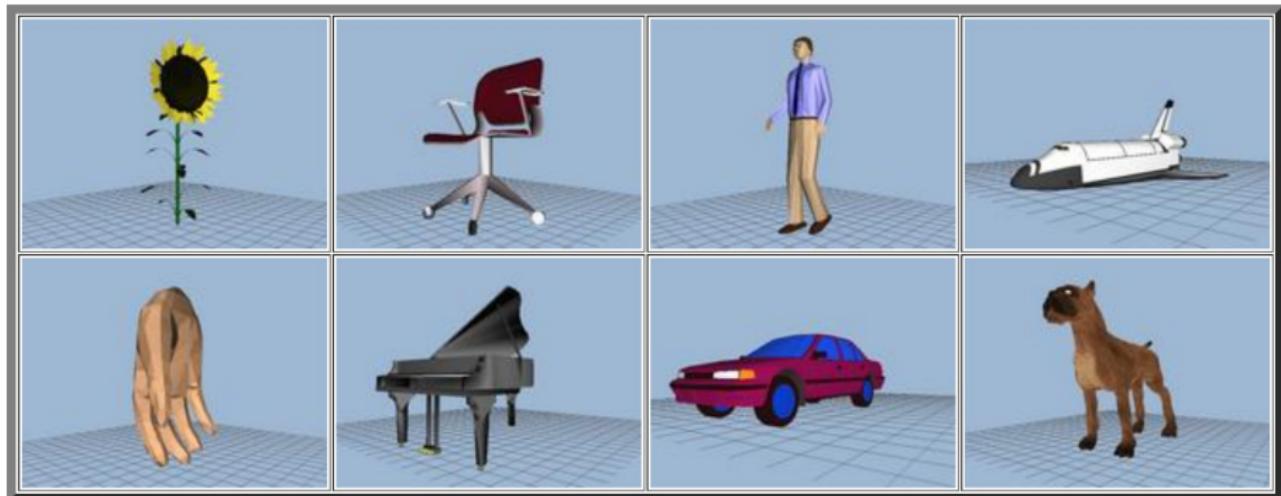
“The Stanford bunny” - 1994.



Sursa: [Stanford scanning repository](#)

Scurt istoric, resurse (iii)

“The Princeton shape benchmark” - ~ 2005



Sursa: *Princeton shape benchmark*

Scurt istoric, resurse (iv)

ModelNet - 2015

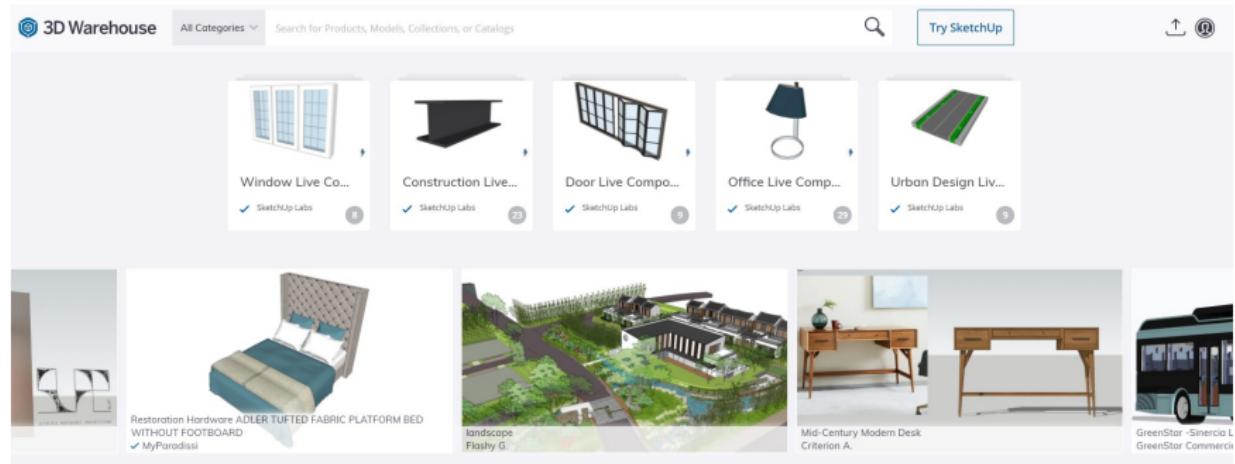


Figure 5: **ModelNet Dataset.** Left: word cloud visualization of the ModelNet dataset based on the number of 3D models in each category. Larger font size indicates more instances in the category. Right: Examples of 3D chair models.

Sursa: [Wu et al., 2015]

Scurt istoric, resurse (v)

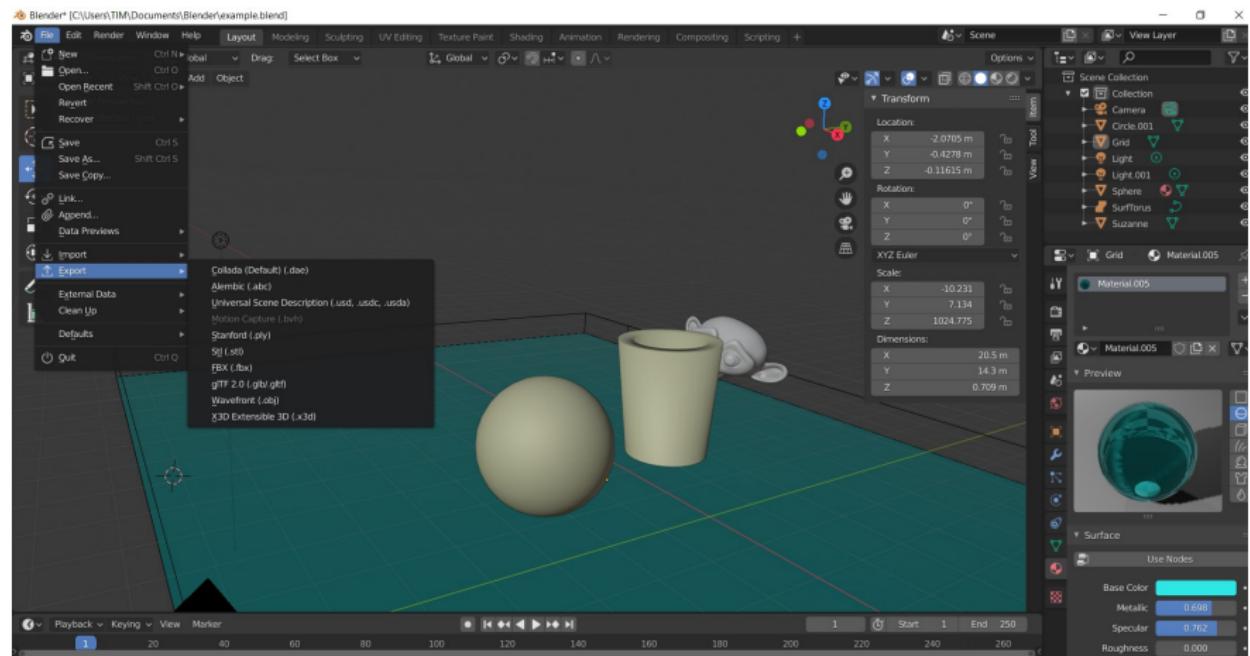
... în prezent există multe resurse disponibile!



Sursa: [3D warehouse](#)

Resurse (vi)

... modele 3D pot fi create cu aplicații dedicate!



Exemplu: .PLY

Formatul PLY (altă referință)

bun_zipper.ply - 3D Viewer

File Edit Tools View Help



bun_zipper - Notepad

File Edit Format View Help

ply
format ascii 1.0
comment zipper output
element vertex 35947
property float x
property float y
property float z
property float confidence
property float intensity
element face 69451
property list uchar int vertex_indices
end_header
-0.0378297 0.12794 0.00447467 0.850855 0.5
-0.0447794 0.128887 0.00190497 0.900159 0.5
-0.0680095 0.151244 0.0371953 0.398443 0.5
-0.0922874 0.13015 0.0232201 0.85268 0.5
-0.0226054 0.126675 0.00715587 0.675938 0.5
-0.0251078 0.125921 0.00624226 0.711533 0.5

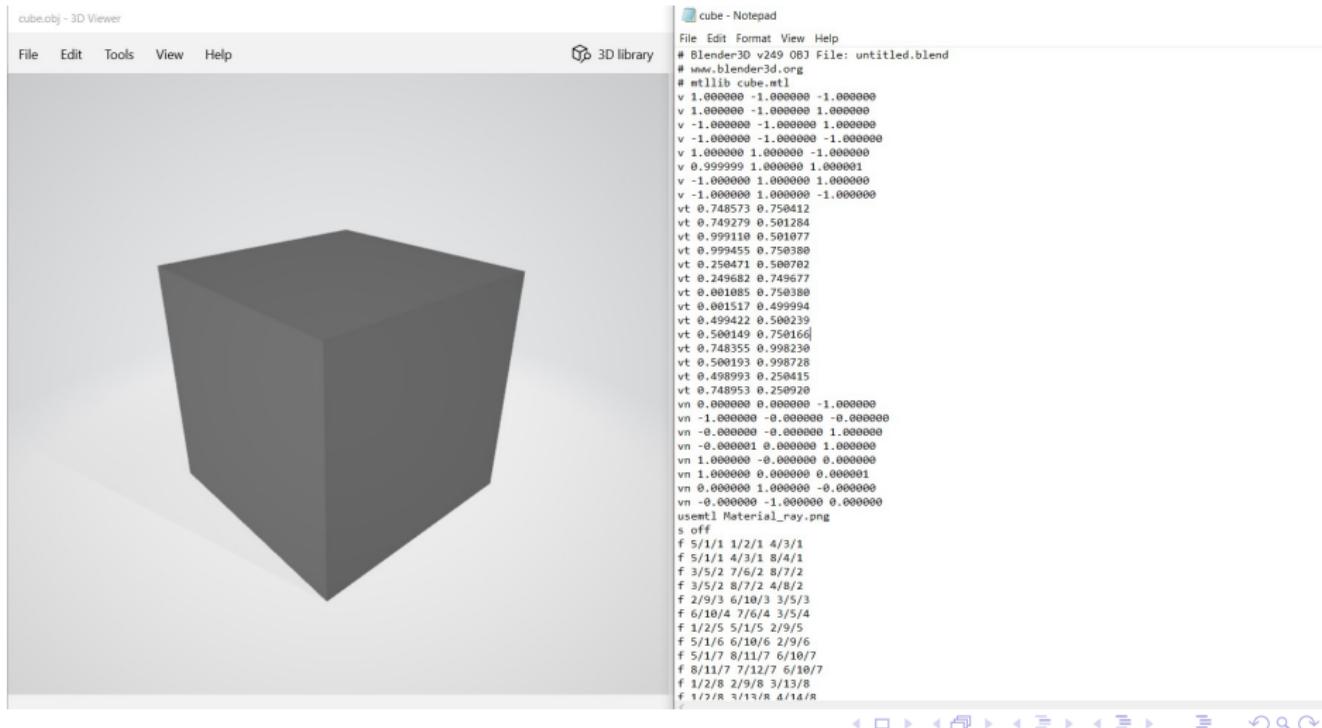
bun_zipper - Notepad

File Edit Format View Help

-0.0678418 0.143634 -0.0139334 0.718635 0.5
-0.0677458 0.149084 -0.0355035 0.386188 0.5
-0.0221568 0.157426 -0.00647102 0.226161 0.5
-0.0708454 0.150585 -0.0434585 0.224465 0.337855
-0.0310262 0.153728 -0.00354608 0.167698 0.5
-0.0408442 0.15362 -0.00816685 0.734503 0.5
3 21216 21215 20399
3 9186 9280 14838
3 16020 13433 5187
3 16021 16020 5187
3 20919 20920 21003
3 23418 15239 23127
3 30553 27378 30502
3 7291 7293 21464
3 12883 12714 13083
3 12777 4682 16928
3 22066 22936 21048
3 21134 22066 21048

Exemplu: .OBJ

Formatul .OBJ (altă referință)



Cadru

► Grafica 3D:

- ce reprezentăm? (aspecte teoretice)
- cum reprezentăm? (funcționalități OpenGL)

Cadru

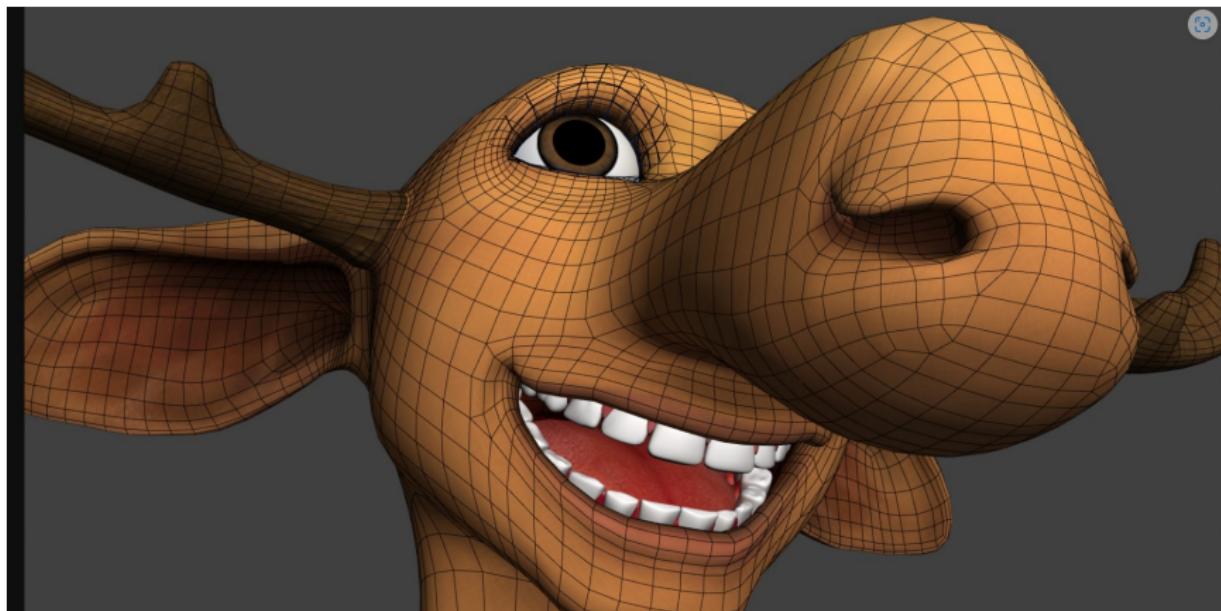
- ▶ Grafica 3D:
 - ce reprezentăm? (aspecte teoretice)
 - cum reprezentăm? (funcționalități OpenGL)
- ▶ Pentru a reprezenta cât mai realist un obiect 3D, pe lângă **coordonate, culori, coordonate de texturare**, un element cheie sunt **vectorii normali** asociați vârfurilor (un atribut al vârfurilor, indicați în funcția de tip creare a VBO, apoi transferați în shader).

Cadru

- ▶ Grafica 3D:
 - ce reprezentăm? (aspecte teoretice)
 - cum reprezentăm? (funcționalități OpenGL)
- ▶ Pentru a reprezenta cât mai realist un obiect 3D, pe lângă **coordonate, culori, coordonate de texturare**, un element cheie sunt **vectorii normali** asociați vârfurilor (un atribut al vârfurilor, indicați în funcția de tip creare a VBO, apoi transferați în shader).
- ▶ Formatele standard pentru obiectele 3D includ astfel de informații. Extrăgând informațiile din fișier, astfel de modele 3D pot fi utilizate în OpenGL.

1. Poliedre / alte obiecte

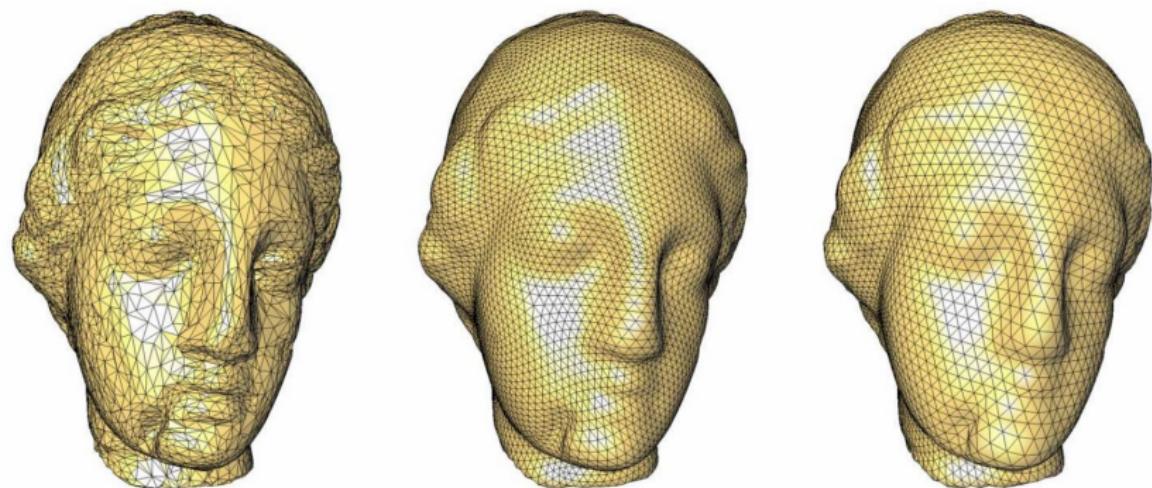
Sunt legate de rețele de poligoane/triunghiuri (*polygon/triangle meshes*)



Sursa: https://talkingmoose.ca/wp-content/uploads/2012/01/Moose_11112_10.jpg

1. Poliedre / alte obiecte

Sunt legate de rețele de poligoane/triunghiuri (*polygon/triangle meshes*)



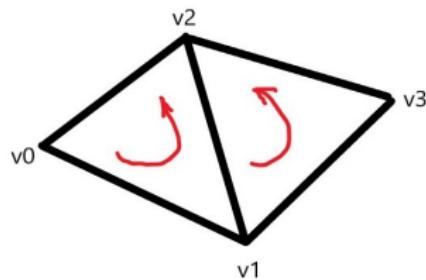
Sursa: <https://hal.inria.fr/file/index/docid/186820/filename/modeling-course.pdf>

1. Poliedre: vectori normali pentru triunghiuri adiacente

- Un element esențial este indexarea vârfurilor.

1. Poliedre: vectori normali pentru triunghiuri adiacente

- ▶ Un element esențial este indexarea vârfurilor.
- ▶ Legat de modul de calcul pentru normale / indexarea vârfurilor:



Fețele din imagine sunt indexate a.î. să fie același sens de parcursare:

0 1 2
2 1 3

1. Poliedre: vectori normali

a) Pentru triunghiuri/poligoane convexe: descriere în cursurile anterioare.

1. Poliedre: vectori normali

- a) Pentru triunghiuri/poligoane convexe: descriere în cursurile anterioare.
- b) Pentru rețele de triunghiuri:

1. Poliedre: vectori normali

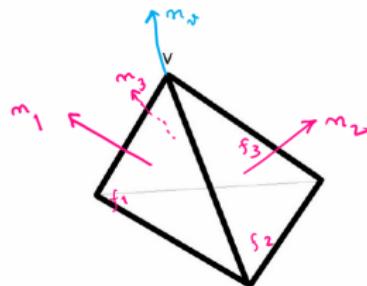
- a) Pentru triunghiuri/poligoane convexe: descriere în cursurile anterioare.
- b) Pentru rețele de triunghiuri:
 - (i) se lucrează la nivel de vârfuri, pentru fiecare vârf se calculează normala (! normala este atribut al vârfurilor), folosind normalele fețelor adiacente.

1. Poliedre: vectori normali

- a) Pentru triunghiuri/poligoane convexe: descriere în cursurile anterioare.
- b) Pentru rețele de triunghiuri:
 - (i) se lucrează la nivel de vârfuri, pentru fiecare vârf se calculează normala (! normala este atribut al vârfurilor), folosind normalele fețelor adiacente.
 - (ii) se folosește pentru fiecare față normala, aşa cum a fost calculată (! atenție la implementare: normala este atribut al vârfurilor).

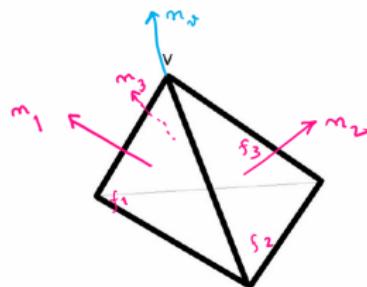
1. Poliedre: vectori normali - detaliere

- Fie v un vârf incident cu fețele f_1, f_2, \dots, f_q , având vectorii normali n_1, n_2, \dots, n_q (respectiv).



1. Poliedre: vectori normali - detaliere

- Fie v un vârf incident cu fețele f_1, f_2, \dots, f_q , având vectorii normali n_1, n_2, \dots, n_q (respectiv).

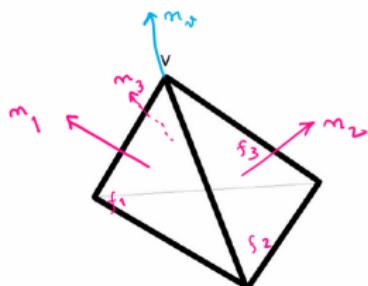


- Vectorul normal în v poate fi definit ca

$$\mathbf{n}_v = \frac{\sum \mathbf{n}_i}{\| \sum \mathbf{n}_i \|}.$$

1. Poliedre: vectori normali - detaliere

- Fie v un vârf incident cu fețele f_1, f_2, \dots, f_q , având vectorii normali n_1, n_2, \dots, n_q (respectiv).



- Vectorul normal în v poate fi definit ca

$$n_v = \frac{\sum n_i}{\|\sum n_i\|}.$$

- (Variantă mai generală) Se pot considera ponderi $\lambda_1, \dots, \lambda_q$ (au suma $\sum \lambda_i = 1$) asociate fețelor (de exemplu date de arii), apoi n_v se definește ca

$$n_v = \frac{\sum \lambda_i n_i}{\|\sum \lambda_i n_i\|}.$$

2. Suprafețe implice

Forma generală: $F(x, y, z) = 0$ (practic - dificil de implementat)

Exemple (suprafețe date de ecuații de gradul II - cuadrice)

- $x^2 + y^2 + z^2 = 1$
- $x^2 + y^2 - z^2 = 1$
- $x^2 - y^2 - z^2 = 1$
- $x^2 + y^2 - z^2 = 0$
- $x^2 + y^2 = 1$
- $x^2 - y^2 = 2z$

2. Suprafețe implice

Forma generală: $F(x, y, z) = 0$ (practic - dificil de implementat)

Exemple (suprafețe date de ecuații de gradul II - cuadrice)

- $x^2 + y^2 + z^2 = 1$ sferă
- $x^2 + y^2 - z^2 = 1$ hiperboloid cu 1 pânză
- $x^2 - y^2 - z^2 = 1$ hiperboloid cu 2 pânze
- $x^2 + y^2 - z^2 = 0$ con circular drept
- $x^2 + y^2 = 1$ cilindru circular drept
- $x^2 - y^2 = 2z$ paraboloid hiperbolic

2. Suprafețe implice: vectori normali

- **Vectori normali.** Fie (x_0, y_0, z_0) un punct al suprafeței. Normala exterioară la suprafață este dată de vectorul

$$\nabla F(x_0, y_0, z_0) = \left(\frac{\partial F}{\partial x}(x_0, y_0, z_0), \frac{\partial F}{\partial y}(x_0, y_0, z_0), \frac{\partial F}{\partial z}(x_0, y_0, z_0) \right).$$

2. Suprafețe implicate: vectori normali

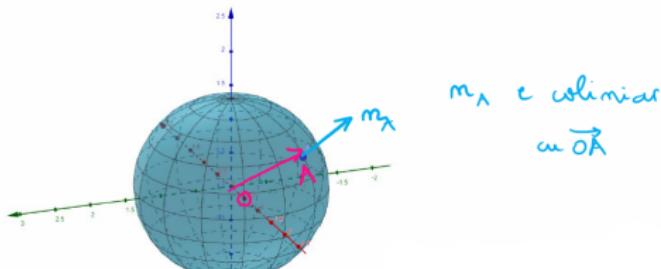
- ▶ **Vectori normali.** Fie (x_0, y_0, z_0) un punct al suprafeței. Normala exterioară la suprafață este dată de vectorul

$$\nabla F(x_0, y_0, z_0) = \left(\frac{\partial F}{\partial x}(x_0, y_0, z_0), \frac{\partial F}{\partial y}(x_0, y_0, z_0), \frac{\partial F}{\partial z}(x_0, y_0, z_0) \right).$$

- **Exemplu.** Sferă de ecuație $x^2 + y^2 + z^2 = 1$, deci $F(x, y, z) = x^2 + y^2 + z^2 - 1$. Fixăm un punct (x_0, y_0, z_0) pe sferă:

$$\frac{\partial F}{\partial x}(x_0, y_0, z_0) = 2x_0, \quad \frac{\partial F}{\partial y}(x_0, y_0, z_0) = 2y_0, \quad \frac{\partial F}{\partial z}(x_0, y_0, z_0) = 2z_0.$$

Vectorul $\nabla(x_0, y_0, z_0)$ este coliniar cu vectorul de poziție (x_0, y_0, z_0) .



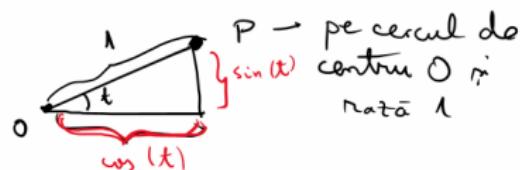
3. Reprezentări parametrice ale suprafețelor. Preliminarii - reprezentarea curbelor

Obs. Jie $c: \mathbb{R} \rightarrow \mathbb{R}^2$; $c(t) = (\cos(t), \sin(t))$, deci

$$\begin{cases} x = \cos(t) \\ y = \sin(t) \end{cases}, t \in \mathbb{R}$$

↓ cercul de centru O

în rază 1; t: parametru \rightarrow setarează



○ curbă

cercul de centru C și rază r :

$$\begin{cases} x = c_x + r \cos(t) \\ y = c_y + r \sin(t) \end{cases}, t \in \mathbb{R}$$

- se aleg valori pt. t
- se calculează pct. correspunzătoare
- se unesc și se obține o aproximare a curbei

3. Reprezentări parametrice ale suprafețelor. Exemple

1)

$$\begin{cases} x = r \cos u \cos v \\ y = r \cos u \sin v \\ z = r \sin u \end{cases} \quad r > 0 \text{ fixat, } u, v \in \mathbb{R} \text{ parametri}$$

este sfera cu centrul $O = (0, 0, 0)$ și raza r , are ecuația
 $x^2 + y^2 + z^2 = r^2$.

3. Reprezentări parametrice ale suprafețelor. Exemple

1)

$$\begin{cases} x = r \cos u \cos v \\ y = r \cos u \sin v \\ z = r \sin u \end{cases} \quad r > 0 \text{ fixat, } u, v \in \mathbb{R} \text{ parametri}$$

este **sferă cu centrul** $O = (0, 0, 0)$ și **raza** r , are ecuația
 $x^2 + y^2 + z^2 = r^2$.

Observații:

- (i) Această reprezentare a fost utilizată pentru “survolarea scenelor” (observatorul se deplasează, practic, pe o sferă).
- (ii) Desenarea sferei folosind reprezentarea parametrică este implementată în codul sursă 09_03_sfера.cpp.
- (iii) Stabiliți care este reprezentarea unei sfere de centru oarecare $C = (x_C, y_C, z_C)$.

3. Reprezentări parametrice ale suprafețelor. Exemple

2)

$$\begin{cases} x = r \cos u \\ y = r \sin u \\ z = v \end{cases} \quad r > 0 \text{ fixat, } u, v \in \mathbb{R} \text{ parametri}$$

3. Reprezentări parametrice ale suprafețelor. Exemple

2)

$$\begin{cases} x = r \cos u \\ y = r \sin u \\ z = v \end{cases} \quad r > 0 \text{ fixat, } u, v \in \mathbb{R} \text{ parametri}$$

este un **cilindru circular drept**, are ecuația $x^2 + y^2 = r^2$.

3. Reprezentări parametrice ale suprafețelor. Exemple

2)

$$\begin{cases} x = r \cos u \\ y = r \sin u \\ z = v \end{cases} \quad r > 0 \text{ fixat, } u, v, \in \mathbb{R} \text{ parametri}$$

este un **cilindru circular drept**, are ecuația $x^2 + y^2 = r^2$.

3)

$$\begin{cases} x = v \cos u \\ y = v \sin u \\ z = v \end{cases} \quad u, v, \in \mathbb{R} \text{ parametri}$$

3. Reprezentări parametrice ale suprafețelor. Exemple

2)

$$\begin{cases} x = r \cos u \\ y = r \sin u \\ z = v \end{cases} \quad r > 0 \text{ fixat, } u, v, \in \mathbb{R} \text{ parametri}$$

este un **cilindru circular drept**, are ecuația $x^2 + y^2 = r^2$.

3)

$$\begin{cases} x = v \cos u \\ y = v \sin u \\ z = v \end{cases} \quad u, v, \in \mathbb{R} \text{ parametri}$$

este un **con circular drept**, are ecuația $x^2 + y^2 - z^2 = 0$.

3. Reprezentări parametrice ale suprafețelor. Definiția generală

În general, o suprafață parametrizată este dată de o funcție

$$f : U \times V \rightarrow \mathbb{R}^3, \quad U, V \subset \mathbb{R} \text{ intervale}$$

Elemente de forma $u \in U, v \in V$ se numesc **parametri**. Pentru diverse valori ale parametrilor se obțin diferite puncte ale suprafeței. **Pentru a desena o suprafață sunt selectate anumite valori ale parametrilor și, practic, sunt eșantionate puncte pe respectiva suprafață.**

3. Reprezentări parametrice ale suprafețelor.

Implementare: vârfuri

Pentru implementare: se consideră

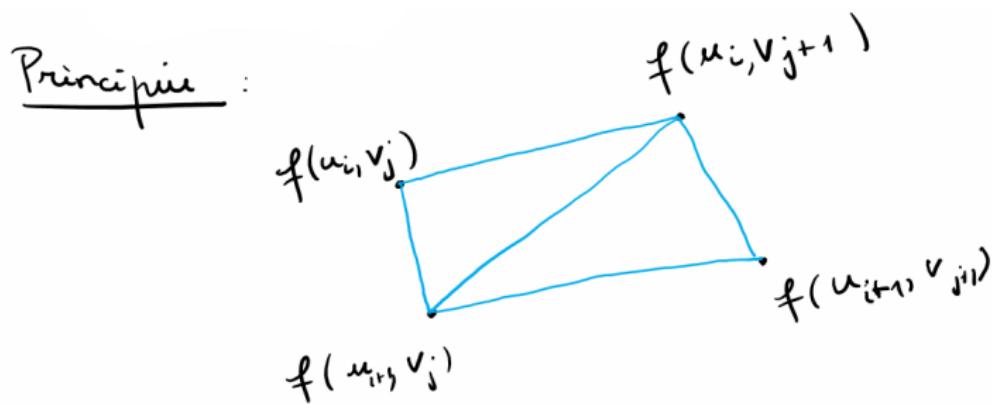
$$\begin{aligned}\tilde{U} &= \{u_1, u_2, \dots, u_m\} \subset U, \\ \tilde{V} &= \{v_1, v_2, \dots, v_n\} \subset V.\end{aligned}$$

Pentru fiecare pereche (i, j) calculăm $f(u_i, v_j)$, acestea reprezentând coordonatele unui vârf, corespunzător unui punct de pe suprafață. În plus, fiecărei astfel de perechi îi este asociat un index corespunzător vârfului.

3. Reprezentări parametrice ale suprafețelor.

Implementare: fețe

Se generează folosind triunghiuri sau patrulatere și “legând” în mod adecvat vârfurile învecinate.



Sunt desenate două triunghiuri care aproximează, local, suprafața respectivă. Se utilizează indexarea vârfurilor.

3. Reprezentări parametrice ale suprafețelor.

Implementare: vectori normali

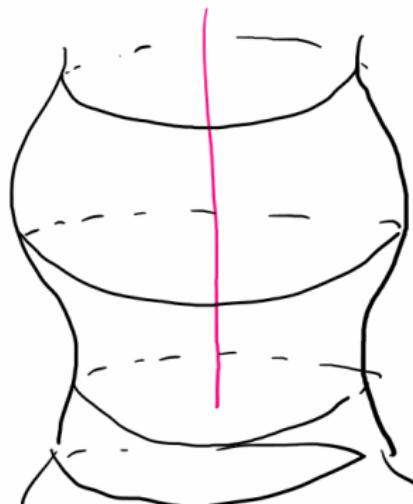
Pentru normale: două variante:

- (i) se aplică principiile pentru rețele de triunghiuri;
- (ii) fixând (u, v) și punctul corespunzător de pe suprafață, un vector său normal la suprafață în punctul respectiv se poate obține calculând $\frac{\partial f}{\partial u}(u, v) \times \frac{\partial f}{\partial v}(u, v)$, apoi normalizând:

$$\mathbf{s} = \frac{\frac{\partial f}{\partial u}(u, v) \times \frac{\partial f}{\partial v}(u, v)}{\left\| \frac{\partial f}{\partial u}(u, v) \times \frac{\partial f}{\partial v}(u, v) \right\|}.$$

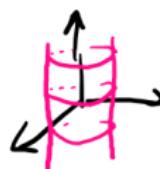
4. Un exemplu: suprafețe de rotație. Illustrare

O suprafață de rotație se obține dacă o curbă este rotită în jurul unei drepte (axa de rotație) pe care nu o intersectează. De fapt: fiecare punct al curbei descrie un cerc având centrul pe axa de rotație (punctul de intersecție dintre axă și planul perpendicular pe axă care trece prin punctul respectiv).



4. Un exemplu: suprafețe de rotație. Cazuri particulare

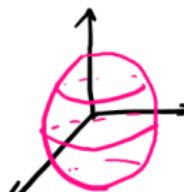
- (i) segment paralel cu axa de rotație \Rightarrow cilindru circular drept (sau o porțiune a acestuia)



- (ii) segment coplanar cu axa de rotație, dar care nu este paralel cu axa de rotație și nu o intersectează \Rightarrow trunchi de con



- (iii) semicerc \Rightarrow sferă (eventual fără poli)



4. Un exemplu: suprafețe de rotație. Reprezentare parametrică

- ▶ Practic: se aleg două funcții $\varphi, \psi : I \rightarrow \mathbb{R}$, unde $I \subset \mathbb{R}$ este un interval și $\varphi(v) > 0, \forall v \in I$. Fie

$$f : [0, 2\pi] \times I \rightarrow \mathbb{R}^3,$$

$$f(u, v) = (\varphi(v) \cos u, \varphi(v) \sin u, \psi(v))$$

(uneori suprafețele pot fi definite pe $\mathbb{R} \times I$ sau cu ordinea parametrilor inversată).

4. Un exemplu: suprafețe de rotație. Reprezentare parametrică

- ▶ Practic: se aleg două funcții $\varphi, \psi : I \rightarrow \mathbb{R}$, unde $I \subset \mathbb{R}$ este un interval și $\varphi(v) > 0, \forall v \in I$. Fie

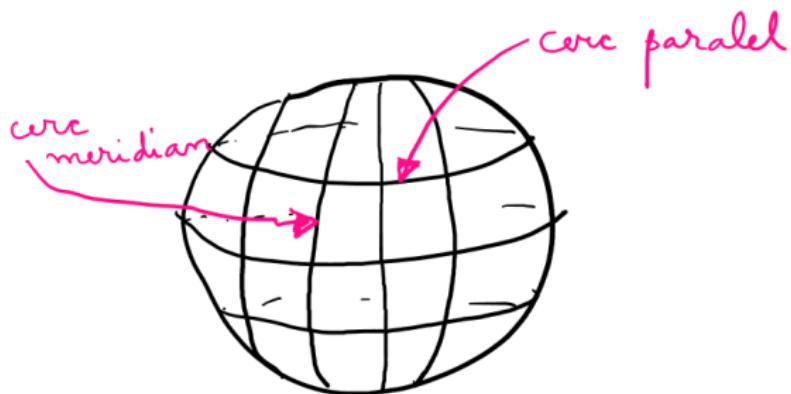
$$f : [0, 2\pi] \times I \rightarrow \mathbb{R}^3,$$

$$f(u, v) = (\varphi(v) \cos u, \varphi(v) \sin u, \psi(v))$$

(uneori suprafețele pot fi definite pe $\mathbb{R} \times I$ sau cu ordinea parametrilor inversată).

- ▶ Reprezentarea de mai sus corespunde suprafeței obținute prin rotirea curbei $v \mapsto (\varphi(v), 0, \psi(v))$ în jurul axei Oz . Condiția $\varphi(v) > 0, \forall v \in I$ corespunde faptului că această curbă nu intersectează axa de rotație (dreapta Oz).

4. Un exemplu: suprafețe de rotație. Comentarii



În general, pe o suprafață de rotație, curbele $u = \text{const}$ și $v = \text{const}$ au denumiri speciale.

Fie $(u_0, v_0) \in [0, 2\pi] \times I$, deci $f(u_0, v_0)$ este un punct de pe suprafață.

- (i) $v = v_0$, u =variabil, $u \mapsto f(u, v_0)$ cerc paralel
- (ii) $u = u_0$, v =variabil, $v \mapsto f(u_0, v)$ cerc meridian

Illuminarea scenelor

Mihai-Sorin Stupariu

Sem. I, 2022 - 2023

Modele de iluminare - generalități

Un model de iluminare face referire la:

- (a) elemente luate în considerare
- (b) parametri corespunzători elementelor de la (a)
- (c) modul în care sunt “aggregate” elementele de la (a)

Modelul Phong de iluminare

color = \uparrow

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

(1)

termenul legat de "emisie"
 termen ambiental
 independent de surse de lumină
 unei surse de lumină
 N termeni, fiecare coresp.
 unei surse de lumină

unde N este numărul surselor de lumină.

Ecuția (1) este implementată în shader (de vârfuri sau de fragment).

Termenul de emisie și termenul ambiental

- ▶ *Emission:* este ceea ce “emite” vârful respectiv (util pentru surse de lumină).

Termenul de emisie și termenul ambiental

- ▶ *Emission*: este ceea ce “emite” vârful respectiv (util pentru surse de lumină).
- ▶ *Ambiental*: nu există surse de lumină, este doar efectul unei luminozități de fond.

Termenul de emisie și termenul ambiental

- ▶ *Emission*: este ceea ce “emite” vârful respectiv (util pentru surse de lumină).
- ▶ *Ambiental*: nu există surse de lumină, este doar efectul unei luminozități de fond.
- ▶ $\text{ambient}_{\text{light model}} * \text{ambient}_{\text{material}}$. **Operația * este dată de înmulțirea pe componente.**
- ▶ Exemplu:

$$\begin{aligned}(0.2, 0.4, 0.3) &\leftarrow (0.1, 0.8, 0.6) = \\&= (0.02, 0.32, 0.18)\end{aligned}$$

Pentru o sursă de lumină i

(i)

(ii)

attenuation factor $_i$ · spotlight effect $_i$

(ambient term + diffuse term + specular term) $_i$

(iii)

(iv)

(v)

(i) Componenta ambientală

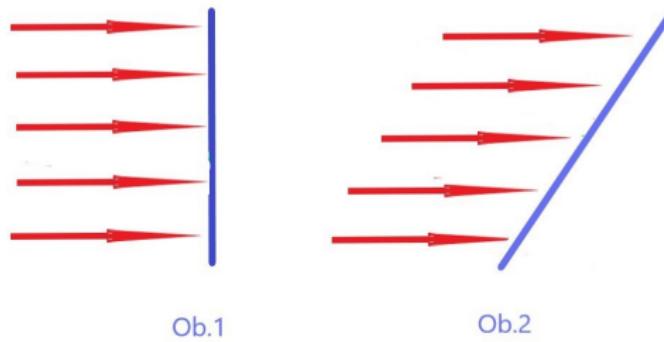
Termenul ambiental corespunzător unei surse de lumină este

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

Teoretic, $\text{ambient}_{\text{light}}$, $\text{ambient}_{\text{material}}$ sunt coduri RGB(A). Practic, este posibil ca acestea să fie și scalari.

(ii) Reflexia difuză (diffuse term)

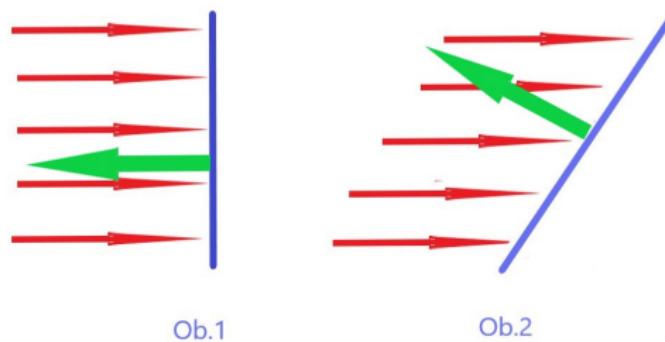
Are legătură cu geometria scenei, lumina reflectată depinde și de incidența luminii asupra obiectelor.



Relevant: unghiul dintre direcția incidentă a luminii și suprafață, de fapt dintre direcția incidentă a luminii și **normala** (în fiecare punct) la suprafață.

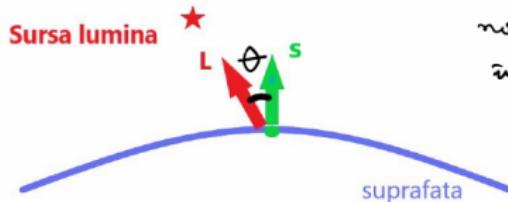
(ii) Reflexia difuză (diffuse term)

Are legătură cu geometria scenei, lumina reflectată depinde și de incidența luminii asupra obiectelor.



Relevant: unghiul dintre direcția incidentă a luminii și suprafață, de fapt dintre direcția incidentă a luminii și **normala** (în fiecare punct) la suprafață.

(ii) Reflexia difuză (diffuse term)



θ = unghiul dintre
normale la suprafață
în vîrful v (notată s)

și

vectorul L =
versor al vectorului
"către sursa de
lumină".

Lumina reflectată este legată de $\cos \theta$

$$\cos \theta = \frac{\langle L, s \rangle}{\|L\| \cdot \|s\|} = \langle L, s \rangle = L \cdot s$$

(produs scalar)

(versori, deci une normale)

...apei înmulțire cu diffuse

(ii) Reflexia difuză (diffuse term)

Reflexia difuză pentru o sursă de lumină este descrisă de

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

unde L este vectorul unitar orientat de la vârf la sursa de lumină (în cazul surselor direcționale este opusul direcției acesteia, normat), iar s este normala la suprafață în vârful considerat. Cazul $L \cdot s < 0$ corespunde situației în care sursa de lumină este în spatele obiectului.

(ii) Reflexia difuză (diffuse term)

- ▶ Comentariu legat de “vector spre sursa de lumină”. **Sursele de lumină:**

(ii) Reflexia difuză (diffuse term)

- ▶ Comentariu legat de “vector spre sursa de lumină”. **Sursele de lumină:**
 - punctuale (bec, lanternă, etc.),

(ii) Reflexia difuză (diffuse term)

- ▶ Comentariu legat de “vector spre sursa de lumină”. **Sursele de lumină:**
 - punctuale (bec, lanternă, etc.),
 - direcționale (Soare, alte corpuri cerești) - de fapt orice sursă de lumină situată la o distanță foarte mare de scenă, în raport cu proporțiile scenei.

(ii) Reflexia difuză (diffuse term)

- ▶ Comentariu legat de “vector spre sursa de lumină”. **Sursele de lumină:**
 - punctuale (bec, lanternă, etc.),
 - direcționale (Soare, alte corpuri cerești) - de fapt orice sursă de lumină situată la o distanță foarte mare de scenă, în raport cu proporțiile scenei.
- ▶ Dacă se lucrează cu `vec4`, atunci distincția se poate face la nivelul celei de-a patra componente:

(ii) Reflexia difuză (diffuse term)

- ▶ Comentariu legat de “vector spre sursa de lumină”. **Sursele de lumină:**
 - punctuale (bec, lanternă, etc.),
 - direcționale (Soare, alte corpuri cerești) - de fapt orice sursă de lumină situată la o distanță foarte mare de scenă, în raport cu proporțiile scenei.
- ▶ Dacă se lucrează cu `vec4`, atunci distincția se poate face la nivelul celei de-a patra componente:
 - 1.0 pentru surse punctuale;

(ii) Reflexia difuză (diffuse term)

- ▶ Comentariu legat de “vector spre sursa de lumină”. **Sursele de lumină:**
 - punctuale (bec, lanternă, etc.),
 - direcționale (Soare, alte corpuri cerești) - de fapt orice sursă de lumină situată la o distanță foarte mare de scenă, în raport cu proporțiile scenei.
- ▶ Dacă se lucrează cu `vec4`, atunci distincția se poate face la nivelul celei de-a patra componente:
 - 1.0 pentru surse punctuale;
 - 0.0 pentru surse direcționale.

(ii) Reflexia difuză (diffuse term)

Dacă se lucrează cu vec3:

- sursă punctuală:

pozitia în \mathbb{R}^3

vec3 LightPos



$$L = \frac{\text{Light Pos} - \text{Frag Pos}}{\|\text{Light Pos} - \text{Frag Pos}\|}$$

- sursă directională:



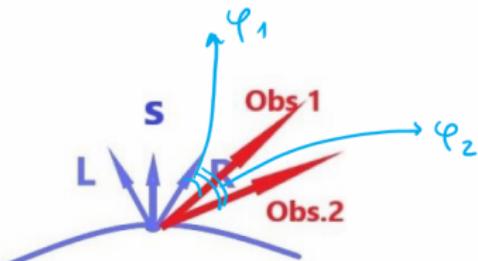
direcție : vec3 d

$$L = \frac{-d}{\|-d\|}$$

(iii) Reflexia speculară



Suprafata mata



Suprafata stralucitoare

R = versor p̄tr. directia "ideală"
(de reflexie a luminii)

φ = unghiul dintre directia ideală
de reflexie și observator

$$\varphi_1 < \varphi_2$$

factor de reflexie proporțional cu

$$\cos^{\text{shininess}} \varphi$$

(iii) Reflexia speculară

Astfel, în unele implementări, reflexia speculară este dată de

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

unde $H = \frac{L + Obs}{\|L + Obs\|}$, iar Obs este vesorul determinat de vârful considerat și poziția observatorului.

(iv) Coeficientul de atenuare

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 distanța de la sursa de lumină la vârful considerat ($d=\text{dist}(\text{FragPos}, \text{LightPos})$).

(v) Efectul de tip spot

- Efectul de tip spot **pentru o sursă punctuală S** este cuantificat de factorul

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

(v) Efectul de tip spot

- Efectul de tip spot **pentru o sursă punctuală S** este cuantificat de factorul

$$\text{spotlight effect} = \begin{cases} 1, & \text{dacă } \theta_I = 180^0 \\ 0, & \text{dacă } v_{\text{obj}} \cdot v_{\text{light}} < \cos \theta_I, \\ (v_{\text{obj}} \cdot v_{\text{light}})^{\alpha_I}, & \text{în celelalte cazuri.} \end{cases}$$

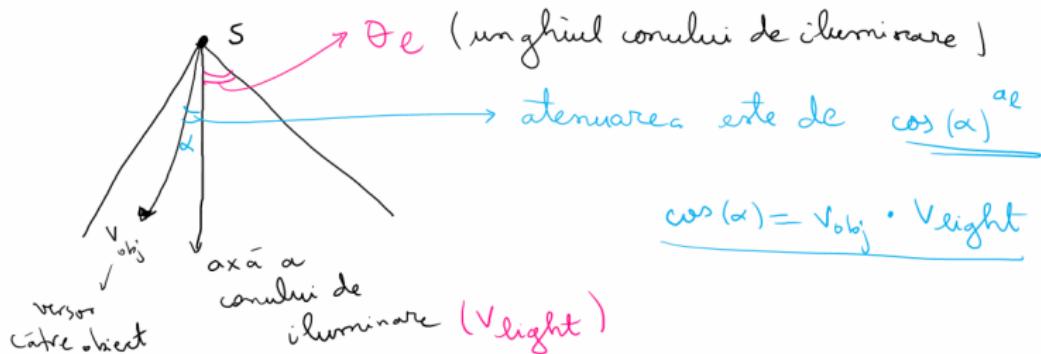
- Cu v_{obj} este vectorul unitar orientat de la sursa de lumină la obiectul iluminat.

(v) Efectul de tip spot

- Efectul de tip spot pentru o sursă punctuală **S** este cuantificat de factorul

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

- Cu v_{obj} este vectorul unitar orientat de la sursa de lumină la obiectul iluminat.
- Elementele definitorii: (i) v_{light} este un versor al axei conului de iluminare; (ii) θ_l este unghiul care definește conul de iluminare (deschiderea acestuia); (iii) α_l este coeficientul de atenuare, care caracterizează cum descrește intensitatea luminii prin îndepărțarea de axa conului.



Coduri sursă

- ▶ 10_01_iluminare.cpp: aplicarea iluminării în cazul unui cub, modelul de iluminare este implementat în shader-ul de fragment.

Coduri sursă

- ▶ 10_01_iluminare.cpp: aplicarea iluminării în cazul unui cub, modelul de iluminare este implementat în shader-ul de fragment.
- ▶ 10_02_model_iluminare.cpp: aplicarea iluminării în cazul unui cub, (i) modelul de iluminare este implementat atât în shader-ul de vârfuri cât și în shader-ul de fragment, (ii) normalele pot fi calculate atât la nivel de vârfuri, cât și la nivelul fețelor.

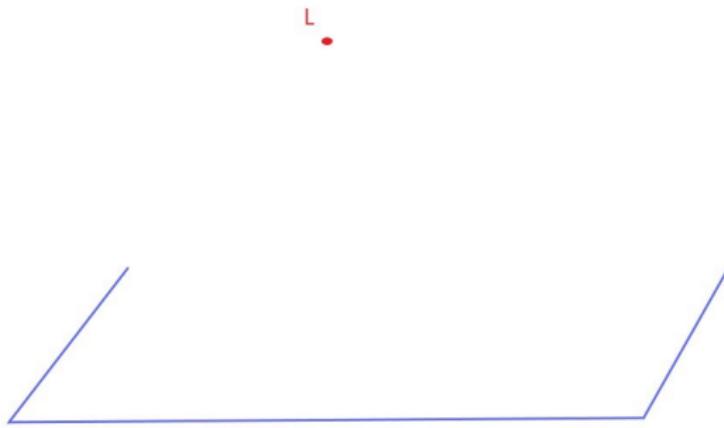
Coduri sursă

- ▶ 10_01_iluminare.cpp: aplicarea iluminării în cazul unui cub, modelul de iluminare este implementat în shader-ul de fragment.
- ▶ 10_02_model_iluminare.cpp: aplicarea iluminării în cazul unui cub, (i) modelul de iluminare este implementat atât în shader-ul de vârfuri cât și în shader-ul de fragment, (ii) normalele pot fi calculate atât la nivel de vârfuri, cât și la nivelul fețelor.
- ▶ 10_03_iluminare_sfera.cpp: aplicarea iluminării pentru sferă

Coduri sursă

- ▶ 10_01_iluminare.cpp: aplicarea iluminării în cazul unui cub, modelul de iluminare este implementat în shader-ul de fragment.
- ▶ 10_02_model_iluminare.cpp: aplicarea iluminării în cazul unui cub, (i) modelul de iluminare este implementat atât în shader-ul de vârfuri cât și în shader-ul de fragment, (ii) normalele pot fi calculate atât la nivel de vârfuri, cât și la nivelul fețelor.
- ▶ 10_03_iluminare_sfera.cpp: aplicarea iluminării pentru sferă
- ▶ Detalii despre codurile sursă se găsesc [pe pagina Moodle, secțiunea Resurse laborator, fișierul Explicatii / detalii laborator](#)

Umbre - problematizare: elementele relevante

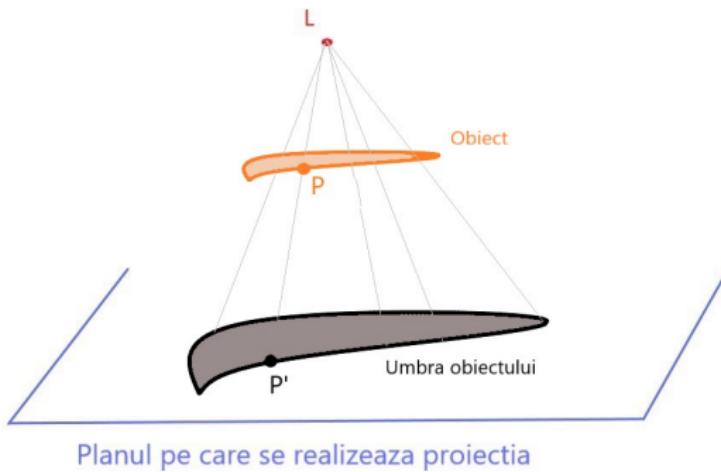


Planul pe care se realizeaza proiectia

Cadru:

- surse de lumina L (în continuare: pp punctuala)
$$L = (x_L, y_L, z_L)$$
- planul pe care se realizeaza proiectie $Ax + By + Cz + D = 0$

Problematizare - elementele relevante



Cadru:

- sursa de lumină L (în continuare: pt. punctuala)

$$L = (x_L, y_L, z_L)$$

- planul pe care se realizează proiecție $Ax + By + Cz + D = 0$

Pt. P al obiectului $\xrightarrow[\text{Irisf.}]{\text{Umbra}} P'$ (pt. al umbrei)

Ce este umbra? Etape pentru calcul

- ▶ **Umbra unui obiect \mathcal{O} :** imaginea lui \mathcal{O} printr-o aplicație (transformare) v . **Scop:** determinarea aplicației v , de fapt a matricei 4×4 asociate, M_v (de explicat modul în care sunt transformate punctele).

Ce este umbra? Etape pentru calcul

- ▶ **Umbra unui obiect \mathcal{O} :** imaginea lui \mathcal{O} printr-o aplicație (transformare) v . **Scop:** determinarea aplicației v , de fapt a matricei 4×4 asociate, M_v (de explicat modul în care sunt transformate punctele).
- ▶ Fie $P = (x_P, y_P, z_P)$ un vârf (punct) al obiectului. Sunt determinate coordonatele lui P' (proiecția perspectivă / centrală a lui P pe plan), în funcție de coordonatele lui P . Acest punct este dat de intersecția dintre dreapta PL și planul π (se presupune că există, dacă LP este paralelă cu planul, nu există umbra...). **Etape:**

Ce este umbra? Etape pentru calcul

- ▶ **Umbra unui obiect \mathcal{O} :** imaginea lui \mathcal{O} printr-o aplicație (transformare) v . **Scop:** determinarea aplicației v , de fapt a matricei 4×4 asociate, M_v (de explicat modul în care sunt transformate punctele).
- ▶ Fie $P = (x_P, y_P, z_P)$ un vârf (punct) al obiectului. Sunt determinate coordonatele lui P' (proiecția perspectivă / centrală a lui P pe plan), în funcție de coordonatele lui P . Acest punct este dat de intersecția dintre dreapta PL și planul π (se presupune că există, dacă LP este paralelă cu planul, nu există umbra...). **Etape:**
 - ▶ Reprezentarea dreptei PL

Ce este umbra? Etape pentru calcul

- ▶ **Umbra unui obiect \mathcal{O} :** imaginea lui \mathcal{O} printr-o aplicație (transformare) v . **Scop:** determinarea aplicației v , de fapt a matricei 4×4 asociate, M_v (de explicat modul în care sunt transformate punctele).
- ▶ Fie $P = (x_P, y_P, z_P)$ un vârf (punct) al obiectului. Sunt determinate coordonatele lui P' (proiecția perspectivă / centrală a lui P pe plan), în funcție de coordonatele lui P . Acest punct este dat de intersecția dintre dreapta PL și planul π (se presupune că există, dacă LP este paralelă cu planul, nu există umbra...). **Etape:**
 - ▶ Reprezentarea dreptei PL
 - ▶ Determinarea coordonatelor punctului de intersecție

Ce este umbra? Etape pentru calcul

- ▶ **Umbra unui obiect \mathcal{O} :** imaginea lui \mathcal{O} printr-o aplicație (transformare) v . **Scop:** determinarea aplicației v , de fapt a matricei 4×4 asociate, M_v (de explicat modul în care sunt transformate punctele).
- ▶ Fie $P = (x_P, y_P, z_P)$ un vârf (punct) al obiectului. Sunt determinate coordonatele lui P' (proiecția perspectivă / centrală a lui P pe plan), în funcție de coordonatele lui P . Acest punct este dat de intersecția dintre dreapta PL și planul π (se presupune că există, dacă LP este paralelă cu planul, nu există umbra...). **Etape:**
 - ▶ Reprezentarea dreptei PL
 - ▶ Determinarea coordonatelor punctului de intersecție
 - ▶ Trecerea la coordonate omogene și scrierea în coordonate omogene

Ce este umbra? Etape pentru calcul

- ▶ **Umbra unui obiect \mathcal{O} :** imaginea lui \mathcal{O} printr-o aplicație (transformare) v . **Scop:** determinarea aplicației v , de fapt a matricei 4×4 asociate, M_v (de explicat modul în care sunt transformate punctele).
- ▶ Fie $P = (x_P, y_P, z_P)$ un vârf (punct) al obiectului. Sunt determinate coordonatele lui P' (proiecția perspectivă / centrală a lui P pe plan), în funcție de coordonatele lui P . Acest punct este dat de intersecția dintre dreapta PL și planul π (se presupune că există, dacă LP este paralelă cu planul, nu există umbra...). **Etape:**
 - ▶ Reprezentarea dreptei PL
 - ▶ Determinarea coordonatelor punctului de intersecție
 - ▶ Trecerea la coordonate omogene și scrierea în coordonate omogene
 - ▶ Determinarea matricei 4×4

Reprezentarea dreptei PL

Ecuațiile dreptei PL

$$\frac{x - x_L}{x_P - x_L} = \frac{y - y_L}{y_P - y_L} = \frac{z - z_L}{z_P - z_L} \stackrel{NOT}{=} \theta \quad \Leftrightarrow$$

Reprezentarea dreptei PL

Ecuațiile dreptei PL

$$\frac{x - x_L}{x_P - x_L} = \frac{y - y_L}{y_P - y_L} = \frac{z - z_L}{z_P - z_L} \stackrel{NOT}{=} \theta \quad \Leftrightarrow$$

$$\begin{cases} x = x_L + \theta(x_P - x_L) \\ y = y_L + \theta(y_P - y_L) \\ z = z_L + \theta(z_P - z_L) \end{cases}, \quad \theta \in \mathbb{R}$$

Reprezentarea dreptei PL

Ecuațiile dreptei PL

$$\frac{x - x_L}{x_P - x_L} = \frac{y - y_L}{y_P - y_L} = \frac{z - z_L}{z_P - z_L} \stackrel{NOT}{=} \theta \quad \Leftrightarrow$$

$$\begin{cases} x = x_L + \theta(x_P - x_L) \\ y = y_L + \theta(y_P - y_L) \\ z = z_L + \theta(z_P - z_L) \end{cases}, \quad \theta \in \mathbb{R}$$

Semnificație: a da un punct de pe dreapta PL este echivalent cu a da o valoare θ

Determinarea coordonatelor punctului de intersecție

Ecuăția planului este $Ax + By + Cz + D = 0$. Pentru a determina intersecția dintre dreaptă și plan (presupunem că există!) determinăm valoarea θ_0 pentru care este verificată ecuația planului, altfel spus pentru care avem

$$0 = A[x_L + \theta_0(x_P - x_L)] + B[y_L + \theta_0(y_P - y_L)] + C[z_L + \theta_0(z_P - z_L)] + D$$

Determinarea coordonatelor punctului de intersecție

Ecuăția planului este $Ax + By + Cz + D = 0$. Pentru a determina intersecția dintre dreaptă și plan (presupunem că există!) determinăm valoarea θ_0 pentru care este verificată ecuația planului, altfel spus pentru care avem

$$0 = A[x_L + \theta_0(x_P - x_L)] + B[y_L + \theta_0(y_P - y_L)] + C[z_L + \theta_0(z_P - z_L)] + D$$

Prin calcul direct se obține

$$\theta_0 = \frac{Ax_L + By_L + Cz_L + D}{A(x_L - x_P) + B(y_L - y_P) + C(z_L - z_P)}$$

Determinarea coordonatelor punctului de intersecție

Ecuăția planului este $Ax + By + Cz + D = 0$. Pentru a determina intersecția dintre dreaptă și plan (presupunem că există!) determinăm valoarea θ_0 pentru care este verificată ecuația planului, altfel spus pentru care avem

$$0 = A[x_L + \theta_0(x_P - x_L)] + B[y_L + \theta_0(y_P - y_L)] + C[z_L + \theta_0(z_P - z_L)] + D$$

Prin calcul direct se obține

$$\theta_0 = \frac{Ax_L + By_L + Cz_L + D}{A(x_L - x_P) + B(y_L - y_P) + C(z_L - z_P)}$$

Am presupus tacit că $A(x_L - x_P) + B(y_L - y_P) + C(z_L - z_P) \neq 0$. Care este interpretarea geometrică a condiției de egalitate?

Determinarea coordonatelor punctului de intersecție

Ecuăția planului este $Ax + By + Cz + D = 0$. Pentru a determina intersecția dintre dreaptă și plan (presupunem că există!) determinăm valoarea θ_0 pentru care este verificată ecuația planului, altfel spus pentru care avem

$$0 = A[x_L + \theta_0(x_P - x_L)] + B[y_L + \theta_0(y_P - y_L)] + C[z_L + \theta_0(z_P - z_L)] + D$$

Prin calcul direct se obține

$$\theta_0 = \frac{Ax_L + By_L + Cz_L + D}{A(x_L - x_P) + B(y_L - y_P) + C(z_L - z_P)}$$

Am presupus tacit că $A(x_L - x_P) + B(y_L - y_P) + C(z_L - z_P) \neq 0$. Care este interpretarea geometrică a condiției de egalitate?

Cunoscând θ_0 , prin înlocuire, se găsesc coordonatele lui P'

Coordonatele punctului de intersecție

$$\begin{aligned}x_{P'} &= x_L + \theta_0(x_P - x_L) = \\&= x_L + \frac{Ax_L + By_L + Cz_L + D}{A(x_L - x_P) + B(y_L - y_P) + C(z_L - z_P)} \cdot (x_P - x_L) =\end{aligned}$$

Coordonatele punctului de intersecție

$$\begin{aligned}x_{P'} &= x_L + \theta_0(x_P - x_L) = \\&= x_L + \frac{Ax_L + By_L + Cz_L + D}{A(x_L - x_P) + B(y_L - y_P) + C(z_L - z_P)} \cdot (x_P - x_L) = \\&\quad \dots\end{aligned}$$

Coordonatele punctului de intersecție

$$\begin{aligned}
 x_{P'} &= x_L + \theta_0(x_P - x_L) = \\
 &= x_L + \frac{Ax_L + By_L + Cz_L + D}{A(x_L - x_P) + B(y_L - y_P) + C(z_L - z_P)} \cdot (x_P - x_L) = \\
 &\quad \dots \\
 &= \frac{x_P(By_L + Cz_L + D) - y_P Bx_L - z_P Cx_L - Dx_L}{(Ax_L + By_L + Cz_L) - (Ax_P + By_P + Cz_P)}
 \end{aligned}$$

Analog

$$\begin{aligned}
 y_{P'} &= \frac{-x_P A y_L + y_P (Ax_L + Cz_L + D) - z_P C y_L - Dy_L}{(Ax_L + By_L + Cz_L) - (Ax_P + By_P + Cz_P)} \\
 z_{P'} &= \frac{-x_P A z_L - y_P B z_L + z_P (Ax_L + By_L + D) - Dz_L}{(Ax_L + By_L + Cz_L) - (Ax_P + By_P + Cz_P)}
 \end{aligned}$$

Coordonatele punctului de intersecție

$$\begin{aligned}
 x_{P'} &= x_L + \theta_0(x_P - x_L) = \\
 &= x_L + \frac{Ax_L + By_L + Cz_L + D}{A(x_L - x_P) + B(y_L - y_P) + C(z_L - z_P)} \cdot (x_P - x_L) = \\
 &\quad \dots \\
 &= \frac{x_P(By_L + Cz_L + D) - y_P Bx_L - z_P Cx_L - Dx_L}{(Ax_L + By_L + Cz_L) - (Ax_P + By_P + Cz_P)}
 \end{aligned}$$

Analog

$$\begin{aligned}
 y_{P'} &= \frac{-x_P A y_L + y_P (Ax_L + Cz_L + D) - z_P C y_L - D y_L}{(Ax_L + By_L + Cz_L) - (Ax_P + By_P + Cz_P)} \\
 z_{P'} &= \frac{-x_P A z_L - y_P B z_L + z_P (Ax_L + By_L + D) - D z_L}{(Ax_L + By_L + Cz_L) - (Ax_P + By_P + Cz_P)}
 \end{aligned}$$

Observați că x_P, y_P, z_P apar la numitor, deci aplicația $P \mapsto P'$ nu este una liniară/afină. Pe de altă parte, numitorul este același. Atât numitorul, cât și numărătorii sunt liniari în x_P, y_P, z_P .

Trecerea la coordonate omogene

Putem scrie, folosind toate cele 4 coordonate:

$$\begin{bmatrix} x_{P'} \\ y_{P'} \\ z_{P'} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\text{numarator}(x_{P'})}{\text{numitorul comun}} \\ \frac{\text{numarator}(y_{P'})}{\text{numitorul comun}} \\ \frac{\text{numarator}(z_{P'})}{\text{numitorul comun}} \\ 1 \end{bmatrix} \stackrel{\text{coord. omog.}}{=} \begin{bmatrix} \text{numarator}(x_{P'}) \\ \text{numarator}(y_{P'}) \\ \text{numarator}(z_{P'}) \\ \text{numitorul comun} \end{bmatrix}$$

Trecerea la coordonate omogene

Putem scrie, folosind toate cele 4 coordonate:

$$\begin{bmatrix} x_{P'} \\ y_{P'} \\ z_{P'} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\text{numarator}(x_{P'})}{\text{numitorul comun}} \\ \frac{\text{numarator}(y_{P'})}{\text{numitorul comun}} \\ \frac{\text{numarator}(z_{P'})}{\text{numitorul comun}} \\ 1 \end{bmatrix} \stackrel{\text{coord. omog.}}{=} \begin{bmatrix} \text{numarator}(x_{P'}) \\ \text{numarator}(y_{P'}) \\ \text{numarator}(z_{P'}) \\ \text{numitorul comun} \end{bmatrix}$$

$$= \begin{bmatrix} x_P(By_L + Cz_L + D) & -y_P Bx_L & -z_P Cx_L & -Dx_L \\ -x_P A y_L & +y_P(Ax_L + Cz_L + D) & -z_P Cy_L & -Dy_L \\ -x_P A z_L & -y_P Bz_L & +z_P(Ax_L + By_L + D) & -Dz_L \\ -x_P A & -y_P B & -z_P C & +(Ax_L + By_L + Cz_L) \end{bmatrix} = M \cdot \begin{bmatrix} x_P \\ y_P \\ z_P \\ 1 \end{bmatrix},$$

Trecerea la coordonate omogene

Concluzie:

$$\begin{bmatrix} x_{P'} \\ y_{P'} \\ z_{P'} \\ 1 \end{bmatrix} = \begin{bmatrix} x_P(By_L + Cz_L + D) & -y_P Bx_L & -z_P Cx_L & -Dx_L \\ -x_P A y_L & +y_P(Ax_L + Cz_L + D) & -z_P Cy_L & -Dy_L \\ -x_P A z_L & -y_P B z_L & +z_P(Ax_L + By_L + D) & -Dz_L \\ -x_P A & -y_P B & -z_P C & +(Ax_L + By_L + Cz_L) \end{bmatrix} = M \cdot \begin{bmatrix} x_P \\ y_P \\ z_P \\ 1 \end{bmatrix}$$

Determinarea matricei 4×4 . Exemplu: 10_04_umbra.cpp

$$M = \begin{pmatrix} By_L + Cz_L + D & -Bx_L & -Cx_L & -Dx_L \\ -Ay_L & Ax_L + Cz_L + D & -Cy_L & -Dy_L \\ -Az_L & -Bz_L & Ax_L + By_L + D & -Dz_L \\ -A & -B & -C & Ax_L + By_L + Cz_L \end{pmatrix}.$$

Ptr. plan de ecuatie $z + D = 0$ (deci $A = 0, B = 0, C = 1$)

$$M = \begin{pmatrix} z_L + D & 0 & -x_L & -Dx_L \\ 0 & z_L + D & -y_L & -Dy_L \\ 0 & 0 & D & -Dz_L \\ 0 & 0 & -1 & z_L \end{pmatrix}$$

Efecte vizuale

Mihai-Sorin Stupariu

Sem. I, 2022 - 2023

Principiul amestecării

Legat de factorul $A(\alpha)$ din codul RGBA. Implicit $A = 1.0$ (obiectele sunt opace). Este specificat în cadrul funcțiilor de culoare / material.
Elementele relevante:

- ▶ Destinația (fragmentul deja procesat), caracterizată de
 $D = (R_d, G_d, B_d, A_d)$ (cod RGBA)
 $F_D = (D_R, D_G, D_B, D_A)$ (factor destinație)

Principiul amestecării

Legat de factorul $A(\alpha)$ din codul RGBA. Implicit $A = 1.0$ (obiectele sunt opace). Este specificat în cadrul funcțiilor de culoare / material.

Elementele relevante:

- ▶ Destinația (fragmentul deja procesat), caracterizată de
 $D = (R_d, G_d, B_d, A_d)$ (cod RGBA)
 $F_D = (D_R, D_G, D_B, D_A)$ (factor destinație)
- ▶ Sursa (fragmentul care este procesat), caracterizată de
 $S = (R_s, G_s, B_s, A_s)$ (cod RGBA)
 $F_S = (S_R, S_G, S_B, S_A)$ (factor sursă)

Principiul amestecării

- factorul destinație / surșă sunt indicate în
 F_D F_S

`glBlendFunc (srcfactor , destfactor);`

F_S F_D

- combinarea se realizează după formula

$$D * F_D + S * F_S$$

menținută de GL_CLAMP.

Valori pentru factorii sursă / destinație

Constantă 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

Exemplu: codul sursă 10_05_amestecare_2D.cpp (I)

- ▶ Sunt desenate un triunghi galben și un triunghi turcoaz (cyan) pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_DST_ALPHA

Exemplu: codul sursă 10_05_amestecare_2D.cpp (I)

- ▶ Sunt desenate un triunghi galben și un triunghi turcoaz (cyan) pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_DST_ALPHA
- ▶ **Varianta 1:** Ordinea: fundal negru, apoi triunghi galben, apoi triunghi turcoaz

Exemplu: codul sursă 10_05_amestecare_2D.cpp (I)

- ▶ Sunt desenate un triunghi galben și un triunghi turcoaz (cyan) pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_DST_ALPHA
- ▶ **Varianta 1:** Ordinea: fundal negru, apoi triunghi galben, apoi triunghi turcoaz
Pas 1. Triunghiul galben (*sursa*₁) pe fundalul negru (*destinatie*₁)

Exemplu: codul sursă 10_05_amestecare_2D.cpp (I)

- ▶ Sunt desenate un triunghi galben și un triunghi turcoaz (cyan) pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_DST_ALPHA
- ▶ **Varianta 1:** Ordinea: fundal negru, apoi triunghi galben, apoi triunghi turcoaz
Pas 1. Triunghiul galben (*sursa*₁) pe fundalul negru (*destinatie*₁)

$$F_S * sursa_1 + F_D * destinatie_1 = (0.5, 0.5, 0.5, 0.5) * sursa_1 + (0.5, 0.5, 0.5, 0.5) * destinatie_1 =$$

Exemplu: codul sursă 10_05_amestecare_2D.cpp (I)

- ▶ Sunt desenate un triunghi galben și un triunghi turcoaz (cyan) pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_DST_ALPHA
- ▶ **Varianta 1:** Ordinea: fundal negru, apoi triunghi galben, apoi triunghi turcoaz
Pas 1. Triunghiul galben (*sursa*₁) pe fundalul negru (*destinatie*₁)

$$F_S * sursa_1 + F_D * destinatie_1 = (0.5, 0.5, 0.5, 0.5) * sursa_1 + (0.5, 0.5, 0.5, 0.5) * destinatie_1 = \\ = (0.5, 0.5, 0.5, 0.5) * (1.0, 1.0, 0.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) =$$

Exemplu: codul sursă 10_05_amestecare_2D.cpp (I)

- ▶ Sunt desenate un triunghi galben și un triunghi turcoaz (cyan) pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_DST_ALPHA
- ▶ **Varianta 1:** Ordinea: fundal negru, apoi triunghi galben, apoi triunghi turcoaz
Pas 1. Triunghiul galben (*sursa*₁) pe fundalul negru (*destinatie*₁)

$$\begin{aligned} F_S * sursa_1 + F_D * destinatie_1 &= (0.5, 0.5, 0.5, 0.5) * sursa_1 + (0.5, 0.5, 0.5, 0.5) * destinatie_1 = \\ &= (0.5, 0.5, 0.5, 0.5) * (1.0, 1.0, 0.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\ &= (0.5, 0.5, 0.0, 0.25) \end{aligned}$$

Exemplu: codul sursă 10_05_amestecare_2D.cpp (I)

- ▶ Sunt desenate un triunghi galben și un triunghi turcoaz (cyan) pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_DST_ALPHA
- ▶ **Varianta 1:** Ordinea: fundal negru, apoi triunghi galben, apoi triunghi turcoaz
Pas 1. Triunghiul galben (*sursa*₁) pe fundalul negru (*destinatie*₁)

$$\begin{aligned} F_S * \text{sursa}_1 + F_D * \text{destinatie}_1 &= (0.5, 0.5, 0.5, 0.5) * \text{sursa}_1 + (0.5, 0.5, 0.5, 0.5) * \text{destinatie}_1 = \\ &= (0.5, 0.5, 0.5, 0.5) * (1.0, 1.0, 0.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\ &= (0.5, 0.5, 0.0, 0.25) \end{aligned}$$

Pas 2. Triunghiul turcoaz (*sursa*₂) pe ceea ce s-a calculat mai sus (*destinatie*₂):

Exemplu: codul sursă 10_05_amestecare_2D.cpp (I)

- ▶ Sunt desenate un triunghi galben și un triunghi turcoaz (cyan) pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_SRC_ALPHA
- ▶ **Varianta 1:** Ordinea: fundal negru, apoi triunghi galben, apoi triunghi turcoaz
Pas 1. Triunghiul galben (*sursa*₁) pe fundalul negru (*destinatie*₁)

$$\begin{aligned} F_S * \text{sursa}_1 + F_D * \text{destinatie}_1 &= (0.5, 0.5, 0.5, 0.5) * \text{sursa}_1 + (0.5, 0.5, 0.5, 0.5) * \text{destinatie}_1 = \\ &= (0.5, 0.5, 0.5, 0.5) * (1.0, 1.0, 0.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\ &= (0.5, 0.5, 0.0, 0.25) \end{aligned}$$

Pas 2. Triunghiul turcoaz (*sursa*₂) pe ceea ce s-a calculat mai sus (*destinatie*₂):

$$F_S * \text{sursa}_2 + F_D * \text{destinatie}_2 = (0.5, 0.5, 0.5, 0.5) * \text{sursa}_2 + (0.5, 0.5, 0.5, 0.5) * \text{destinatie}_2 =$$

Exemplu: codul sursă 10_05_amestecare_2D.cpp (I)

- ▶ Sunt desenate un triunghi galben și un triunghi turcoaz (cyan) pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_SRC_ALPHA
- ▶ **Varianta 1:** Ordinea: fundal negru, apoi triunghi galben, apoi triunghi turcoaz
Pas 1. Triunghiul galben (*sursa*₁) pe fundalul negru (*destinatie*₁)

$$\begin{aligned} F_S * sursa_1 + F_D * destinatie_1 &= (0.5, 0.5, 0.5, 0.5) * sursa_1 + (0.5, 0.5, 0.5, 0.5) * destinatie_1 = \\ &= (0.5, 0.5, 0.5, 0.5) * (1.0, 1.0, 0.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\ &= (0.5, 0.5, 0.0, 0.25) \end{aligned}$$

Pas 2. Triunghiul turcoaz (*sursa*₂) pe ceea ce s-a calculat mai sus (*destinatie*₂):

$$\begin{aligned} F_S * sursa_2 + F_D * destinatie_2 &= (0.5, 0.5, 0.5, 0.5) * sursa_2 + (0.5, 0.5, 0.5, 0.5) * destinatie_2 = \\ &= (0.5, 0.5, 0.5, 0.5) * (0.0, 1.0, 1.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.5, 0.5, 0.0, 0.25) = \end{aligned}$$

Exemplu: codul sursă 10_05_amestecare_2D.cpp (I)

- ▶ Sunt desenate un triunghi galben și un triunghi turcoaz (cyan) pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_SRC_ALPHA
- ▶ **Varianta 1:** Ordinea: fundal negru, apoi triunghi galben, apoi triunghi turcoaz
Pas 1. Triunghiul galben (*sursa*₁) pe fundalul negru (*destinatie*₁)

$$\begin{aligned} F_S * sursa_1 + F_D * destinatie_1 &= (0.5, 0.5, 0.5, 0.5) * sursa_1 + (0.5, 0.5, 0.5, 0.5) * destinatie_1 = \\ &= (0.5, 0.5, 0.5, 0.5) * (1.0, 1.0, 0.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\ &= (0.5, 0.5, 0.0, 0.25) \end{aligned}$$

Pas 2. Triunghiul turcoaz (*sursa*₂) pe ceea ce s-a calculat mai sus (*destinatie*₂):

$$\begin{aligned} F_S * sursa_2 + F_D * destinatie_2 &= (0.5, 0.5, 0.5, 0.5) * sursa_2 + (0.5, 0.5, 0.5, 0.5) * destinatie_2 = \\ &= (0.5, 0.5, 0.5, 0.5) * (0.0, 1.0, 1.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.5, 0.5, 0.0, 0.25) = \\ &= (0.0, 0.5, 0.5, 0.25) + (0.25, 0.25, 0, 0.125) = \end{aligned}$$

Exemplu: codul sursă 10_05_amestecare_2D.cpp (I)

- ▶ Sunt desenate un triunghi galben și un triunghi turcoaz (cyan) pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_SRC_ALPHA
- ▶ **Varianta 1:** Ordinea: fundal negru, apoi triunghi galben, apoi triunghi turcoaz
Pas 1. Triunghiul galben (*sursa*₁) pe fundalul negru (*destinatie*₁)

$$\begin{aligned} F_S * sursa_1 + F_D * destinatie_1 &= (0.5, 0.5, 0.5, 0.5) * sursa_1 + (0.5, 0.5, 0.5, 0.5) * destinatie_1 = \\ &= (0.5, 0.5, 0.5, 0.5) * (1.0, 1.0, 0.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\ &= (0.5, 0.5, 0.0, 0.25) \end{aligned}$$

Pas 2. Triunghiul turcoaz (*sursa*₂) pe ceea ce s-a calculat mai sus (*destinatie*₂):

$$\begin{aligned} F_S * sursa_2 + F_D * destinatie_2 &= (0.5, 0.5, 0.5, 0.5) * sursa_2 + (0.5, 0.5, 0.5, 0.5) * destinatie_2 = \\ &= (0.5, 0.5, 0.5, 0.5) * (0.0, 1.0, 1.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.5, 0.5, 0.0, 0.25) = \\ &= (0.0, 0.5, 0.5, 0.25) + (0.25, 0.25, 0, 0.125) = \\ &= (0.25, 0.75, 0.5, 0.375) \end{aligned}$$

Exemplu: codul sursă 10_05_amestecare_2D.cpp (II)

- Sunt desenate un triunghi turcoaz (cyan) și un triunghi galben pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_SRC_ALPHA

Exemplu: codul sursă 10_05_amestecare_2D.cpp (II)

- ▶ Sunt desenate un triunghi turcoaz (cyan) și un triunghi galben pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_SRC_ALPHA
- ▶ **Varianta 2:** Ordinea: fundal negru, apoi triunghi turcoaz, apoi triunghi galben

Exemplu: codul sursă 10_05_amestecare_2D.cpp (II)

- ▶ Sunt desenate un triunghi turcoaz (cyan) și un triunghi galben pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_SRC_ALPHA
- ▶ **Varianta 2:** Ordinea: fundal negru, apoi triunghi turcoaz, apoi triunghi galben
Pas 1. Triunghiul turcoaz (*sursa₁*) pe fundalul negru (*destinatie₁*)

Exemplu: codul sursă 10_05_amestecare_2D.cpp (II)

- ▶ Sunt desenate un triunghi turcoaz (cyan) și un triunghi galben pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_SRC_ALPHA
- ▶ **Varianta 2:** Ordinea: fundal negru, apoi triunghi turcoaz, apoi triunghi galben
Pas 1. Triunghiul turcoaz (*sursa*₁) pe fundalul negru (*destinatie*₁)

$$F_S * sursa_1 + F_D * destinatie_1 = (0.5, 0.5, 0.5, 0.5) * sursa_1 + (0.5, 0.5, 0.5, 0.5) * destinatie_1 =$$

Exemplu: codul sursă 10_05_amestecare_2D.cpp (II)

- ▶ Sunt desenate un triunghi turcoaz (cyan) și un triunghi galben pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_SRC_ALPHA
- ▶ **Varianta 2:** Ordinea: fundal negru, apoi triunghi turcoaz, apoi triunghi galben
Pas 1. Triunghiul turcoaz (*sursa*₁) pe fundalul negru (*destinatie*₁)

$$F_S * sursa_1 + F_D * destinatie_1 = (0.5, 0.5, 0.5, 0.5) * sursa_1 + (0.5, 0.5, 0.5, 0.5) * destinatie_1 = \\ = (0.5, 0.5, 0.5, 0.5) * (0.0, 1.0, 1.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) =$$

Exemplu: codul sursă 10_05_amestecare_2D.cpp (II)

- ▶ Sunt desenate un triunghi turcoaz (cyan) și un triunghi galben pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_SRC_ALPHA
- ▶ **Varianta 2:** Ordinea: fundal negru, apoi triunghi turcoaz, apoi triunghi galben
Pas 1. Triunghiul turcoaz (*sursa*₁) pe fundalul negru (*destinatie*₁)

$$\begin{aligned}F_S * \text{sursa}_1 + F_D * \text{destinatie}_1 &= (0.5, 0.5, 0.5, 0.5) * \text{sursa}_1 + (0.5, 0.5, 0.5, 0.5) * \text{destinatie}_1 = \\&= (0.5, 0.5, 0.5, 0.5) * (0.0, 1.0, 1.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\&= (0.0, 0.5, 0.5, 0.25)\end{aligned}$$

Exemplu: codul sursă 10_05_amestecare_2D.cpp (II)

- ▶ Sunt desenate un triunghi turcoaz (cyan) și un triunghi galben pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_SRC_ALPHA
- ▶ **Varianta 2:** Ordinea: fundal negru, apoi triunghi turcoaz, apoi triunghi galben
Pas 1. Triunghiul turcoaz (*sursa*₁) pe fundalul negru (*destinatie*₁)

$$\begin{aligned} F_S * sursa_1 + F_D * destinatie_1 &= (0.5, 0.5, 0.5, 0.5) * sursa_1 + (0.5, 0.5, 0.5, 0.5) * destinatie_1 = \\ &= (0.5, 0.5, 0.5, 0.5) * (0.0, 1.0, 1.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\ &= (0.0, 0.5, 0.5, 0.25) \end{aligned}$$

Pas 2. Triunghiul galben (*sursa*₂) pe ceea ce s-a desenat (*destinatie*₂):

Exemplu: codul sursă 10_05_amestecare_2D.cpp (II)

- ▶ Sunt desenate un triunghi turcoaz (cyan) și un triunghi galben pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_SRC_ALPHA
- ▶ **Varianta 2:** Ordinea: fundal negru, apoi triunghi turcoaz, apoi triunghi galben
Pas 1. Triunghiul turcoaz (*sursa*₁) pe fundalul negru (*destinatie*₁)

$$\begin{aligned} F_S * sursa_1 + F_D * destinatie_1 &= (0.5, 0.5, 0.5, 0.5) * sursa_1 + (0.5, 0.5, 0.5, 0.5) * destinatie_1 = \\ &= (0.5, 0.5, 0.5, 0.5) * (0.0, 1.0, 1.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\ &= (0.0, 0.5, 0.5, 0.25) \end{aligned}$$

Pas 2. Triunghiul galben (*sursa*₂) pe ceea ce s-a desenat (*destinatie*₂):

$$F_S * sursa_2 + F_D * destinatie_2 = (0.5, 0.5, 0.5, 0.5) * sursa_2 + (0.5, 0.5, 0.5, 0.5) * destinatie_2 =$$

Exemplu: codul sursă 10_05_amestecare_2D.cpp (II)

- ▶ Sunt desenate un triunghi turcoaz (cyan) și un triunghi galben pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_SRC_ALPHA
- ▶ **Varianta 2:** Ordinea: fundal negru, apoi triunghi turcoaz, apoi triunghi galben
Pas 1. Triunghiul turcoaz (*sursa*₁) pe fundalul negru (*destinatie*₁)

$$\begin{aligned} F_S * sursa_1 + F_D * destinatie_1 &= (0.5, 0.5, 0.5, 0.5) * sursa_1 + (0.5, 0.5, 0.5, 0.5) * destinatie_1 = \\ &= (0.5, 0.5, 0.5, 0.5) * (0.0, 1.0, 1.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\ &= (0.0, 0.5, 0.5, 0.25) \end{aligned}$$

Pas 2. Triunghiul galben (*sursa*₂) pe ceea ce s-a desenat (*destinatie*₂):

$$\begin{aligned} F_S * sursa_2 + F_D * destinatie_2 &= (0.5, 0.5, 0.5, 0.5) * sursa_2 + (0.5, 0.5, 0.5, 0.5) * destinatie_2 = \\ &= (0.5, 0.5, 0.5, 0.5) * (1.0, 1.0, 0.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.5, 0.5, 0.25) = \end{aligned}$$

Exemplu: codul sursă 10_05_amestecare_2D.cpp (II)

- ▶ Sunt desenate un triunghi turcoaz (cyan) și un triunghi galben pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_SRC_ALPHA
- ▶ **Varianta 2:** Ordinea: fundal negru, apoi triunghi turcoaz, apoi triunghi galben
Pas 1. Triunghiul turcoaz (*sursa*₁) pe fundalul negru (*destinatie*₁)

$$\begin{aligned} F_S * sursa_1 + F_D * destinatie_1 &= (0.5, 0.5, 0.5, 0.5) * sursa_1 + (0.5, 0.5, 0.5, 0.5) * destinatie_1 = \\ &= (0.5, 0.5, 0.5, 0.5) * (0.0, 1.0, 1.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\ &= (0.0, 0.5, 0.5, 0.25) \end{aligned}$$

Pas 2. Triunghiul galben (*sursa*₂) pe ceea ce s-a desenat (*destinatie*₂):

$$\begin{aligned} F_S * sursa_2 + F_D * destinatie_2 &= (0.5, 0.5, 0.5, 0.5) * sursa_2 + (0.5, 0.5, 0.5, 0.5) * destinatie_2 = \\ &= (0.5, 0.5, 0.5, 0.5) * (1.0, 1.0, 0.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.5, 0.5, 0.25) = \\ &= (0.5, 0.5, 0.0, 0.25) + (0.0, 0.25, 0.25, 0.125) = \end{aligned}$$

Exemplu: codul sursă 10_05_amestecare_2D.cpp (II)

- Sunt desenate un triunghi turcoaz (cyan) și un triunghi galben pe fundal negru, cu factor destinație și factor sursă dați de GL_SRC_ALPHA și GL_SRC_ALPHA
- **Varianta 2:** Ordinea: fundal negru, apoi triunghi turcoaz, apoi triunghi galben
Pas 1. Triunghiul turcoaz (*sursa*₁) pe fundalul negru (*destinatie*₁)

$$\begin{aligned} F_S * \text{sursa}_1 + F_D * \text{destinatie}_1 &= (0.5, 0.5, 0.5, 0.5) * \text{sursa}_1 + (0.5, 0.5, 0.5, 0.5) * \text{destinatie}_1 = \\ &= (0.5, 0.5, 0.5, 0.5) * (0.0, 1.0, 1.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\ &= (0.0, 0.5, 0.5, 0.25) \end{aligned}$$

Pas 2. Triunghiul galben (*sursa*₂) pe ceea ce s-a desenat (*destinatie*₂):

$$\begin{aligned} F_S * \text{sursa}_2 + F_D * \text{destinatie}_2 &= (0.5, 0.5, 0.5, 0.5) * \text{sursa}_2 + (0.5, 0.5, 0.5, 0.5) * \text{destinatie}_2 = \\ &= (0.5, 0.5, 0.5, 0.5) * (1.0, 1.0, 0.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.5, 0.5, 0.25) = \\ &= (0.5, 0.5, 0.0, 0.25) + (0.0, 0.25, 0.25, 0.125) = \\ &\quad \color{red}{(0.5, 0.75, 0.25, 0.375) \neq (0.25, 0.75, 0.5, 0.375)} \end{aligned}$$

Functii de amestecare pentru scenele 3D: codul 10_06_amestecare_3D.cpp

- ▶ Combinante:

Functii de amestecare pentru scenele 3D: codul 10_06_amestecare_3D.cpp

- ▶ Combinări:
 - ▶ ordinea în care sunt desenate obiectele
 - ▶ testul de adâncime
 - ▶ efectele de amestecare

Functii de amestecare pentru scenele 3D: codul 10_06_amestecare_3D.cpp

- ▶ Combinate:
 - ▶ ordinea în care sunt desenate obiectele
 - ▶ testul de adâncime
 - ▶ efectele de amestecare
- ▶ desenare obiecte opace cu:

Functii de amestecare pentru scenele 3D: codul 10_06_amestecare_3D.cpp

- ▶ Combinante:
 - ▶ ordinea în care sunt desenate obiectele
 - ▶ testul de adâncime
 - ▶ efectele de amestecare
- ▶ desenare obiecte opace cu:
 - ▶ z-buffer activ
 - ▶ buffer de adâncime: normal (read/write) `glDepthMask(GL_TRUE)`

Functii de amestecare pentru scenele 3D: codul 10_06_amestecare_3D.cpp

- ▶ Combinante:
 - ▶ ordinea în care sunt desenate obiectele
 - ▶ testul de adâncime
 - ▶ efectele de amestecare
- ▶ desenare obiecte opace cu:
 - ▶ z-buffer activ
 - ▶ buffer de adâncime: normal (read/write) `glDepthMask(GL_TRUE)`
- ▶ desenare obiecte transparente cu:

Functii de amestecare pentru scenele 3D: codul 10_06_amestecare_3D.cpp

- ▶ Combinări:
 - ▶ ordinea în care sunt desenate obiectele
 - ▶ testul de adâncime
 - ▶ efectele de amestecare
- ▶ desenare obiecte opace cu:
 - ▶ z-buffer activ
 - ▶ buffer de adâncime: normal (read/write) `glDepthMask(GL_TRUE)`
- ▶ desenare obiecte transparente cu:
 - ▶ z-buffer activ
 - ▶ buffer de adâncime: read `glDepthMask(GL_FALSE)`

Efectul de ceață

- **Principiu:** (mecanismul combinațiilor affine) este variată culoarea obiectelor în funcție de distanță, pe baza unei formule de tipul

$$C = f \cdot C_o + (1 - f) \cdot C_f,$$

unde: f = factor ceață; C_o = culoarea inițială a obiectului, C_f = culoarea ceții. Pentru implementare: în shader-ul de fragment `mix`.

Efectul de ceață

- **Principiu:** (mecanismul combinațiilor afine) este variată culoarea obiectelor în funcție de distanță, pe baza unei formule de tipul

$$C = f \cdot C_o + (1 - f) \cdot C_f,$$

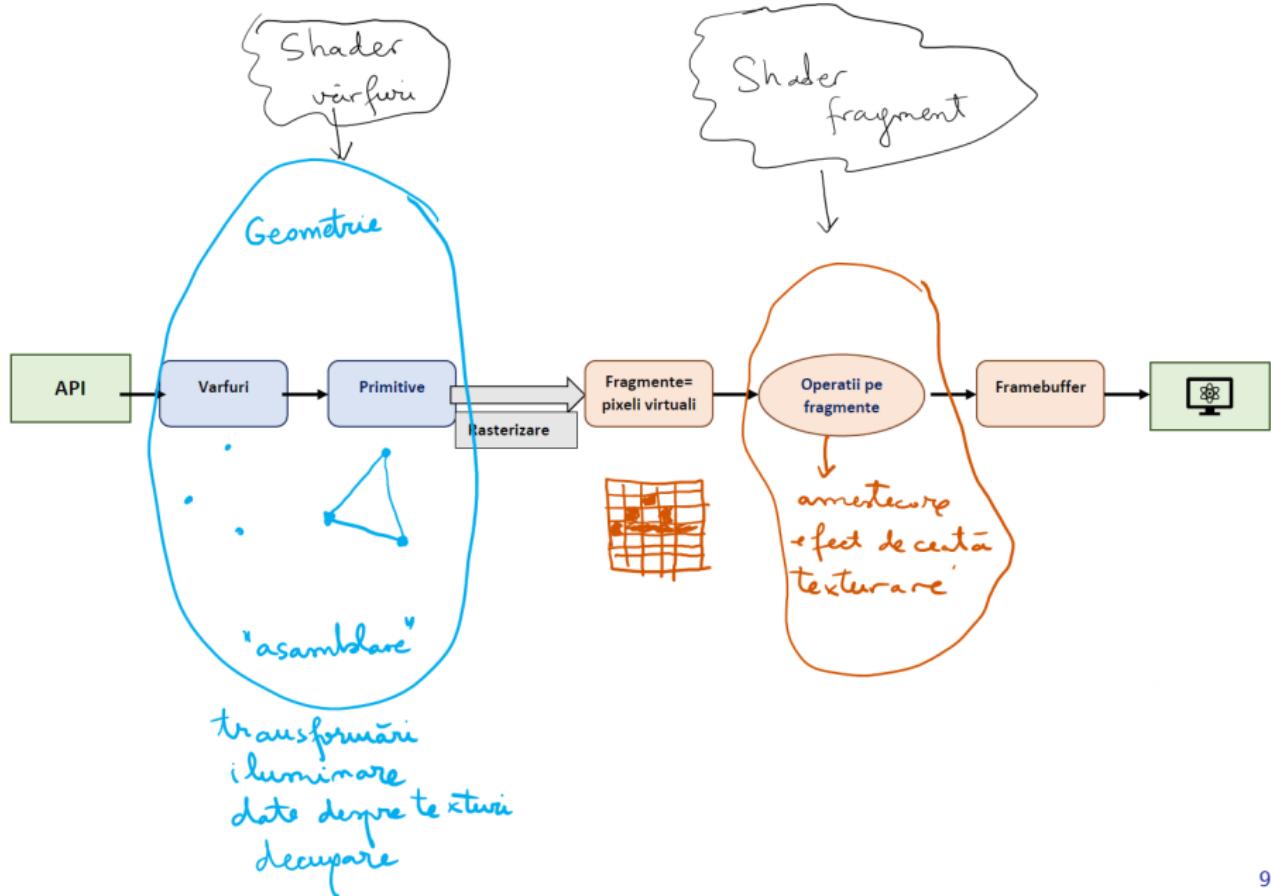
unde: f = factor ceață; C_o = culoarea inițială a obiectului, C_f = culoarea ceții. Pentru implementare: în shader-ul de fragment `mix`.

- Factorul ceață f depinde de z-adâncime (\equiv depth) față de observator, fiind o funcție de forma $f = f(z)$, descrescătoare pe $(0, \infty)$. Exemple:

$$f(z) = \begin{cases} \frac{end-z}{z-start} & (\text{liniar}) \\ e^{-\rho z} & (\text{exponențial}) \\ e^{-\rho z^2} & (\text{exponențial pătratic}) \end{cases}$$

Pentru implementare: necesară distanța z de la observator `inViewPos` la obiect `FragPos`. Parametrii necesari (de exemplu ρ - factor ceață) sunt indicați în cod.

Fluxul operațiilor



Cuaternioni

Mihai-Sorin Stupariu

Sem. I, 2022 - 2023

Problematizare - generalități

Când este considerată o clasă de transformări:

- (i) de câte informații numerice este nevoie pentru a indica o transformare?
- (ii) există o structură algebrică subiacentă?

1. Translații

1. Translații

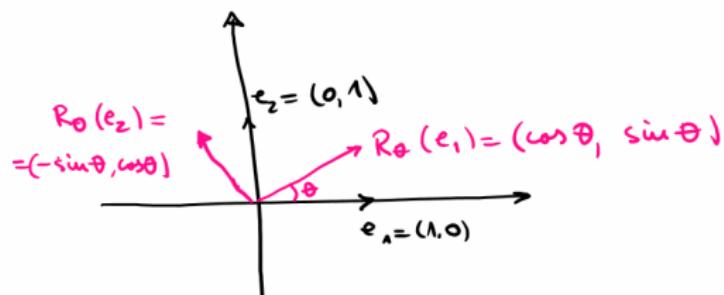
Context 2D, respectiv 3D :

- 2, respectiv 3 numere;
- $(\mathbb{R}^2, +)$, respectiv $(\mathbb{R}^3, +)$

2. Rotații 2D

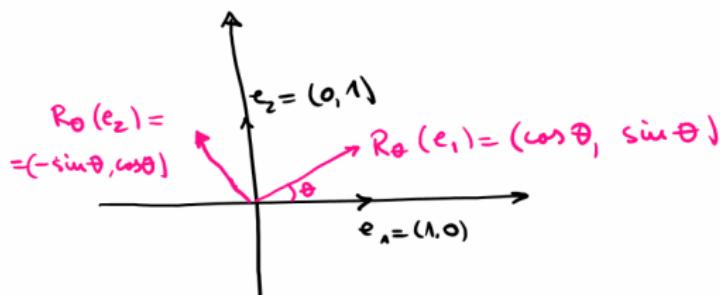
2. Rotații 2D

O rotație 2D (originea este presupusă punct fix) este complet caracterizată de **1** număr: unghiul rotației.



2. Rotații 2D

O rotație 2D (originea este presupusă punct fix) este complet caracterizată de **1** număr: unghiul rotației.



Avem

$$R_\theta(e_1) = (\cos \theta, \sin \theta) = \cos \theta \cdot e_1 + \sin \theta \cdot e_2$$

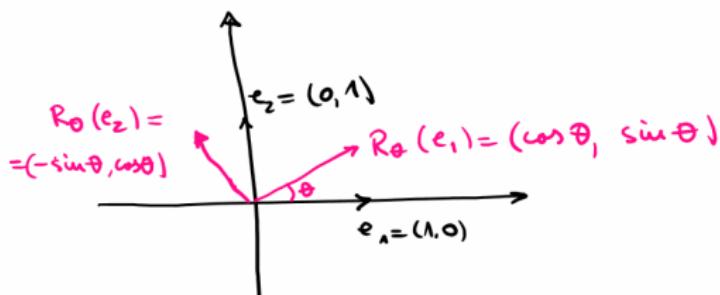
$$R_\theta(e_2) = (-\sin \theta, \cos \theta) = -\sin \theta \cdot e_1 + \cos \theta \cdot e_2$$

Așadar, R_θ este complet caracterizată de matricea

$$\begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}.$$

2. Rotații 2D

O rotație 2D (originea este presupusă punct fix) este complet caracterizată de **1** număr: unghiul rotației.



Avem

$$R_\theta(e_1) = (\cos \theta, \sin \theta) = \cos \theta \cdot e_1 + \sin \theta \cdot e_2$$

$$R_\theta(e_2) = (-\sin \theta, \cos \theta) = -\sin \theta \cdot e_1 + \cos \theta \cdot e_2$$

Așadar, R_θ este complet caracterizată de matricea

$$\begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}.$$

Matricea R_θ verifică relația $R_\theta \cdot R_\theta^T = \mathbb{I}_2$.

2. Rotații 2D. De reținut

- pentru a indica o rotație 2D este necesară / suficientă o singură informație numerică,
- a descrie o rotație \Leftrightarrow
 - \Leftrightarrow a indica modul în care este transformat un reper ortonormat în alt reper ortonormat păstrând orientarea \Leftrightarrow
 - \Leftrightarrow a indica matricea de transformare între repere, în cazul 2D aceasta este de forma
$$\begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

Definiții generale. Grupul ortogonal

Definiție (i) O matrice pătratică $A \in \mathcal{M}_n(\mathbb{R})$ se numește **ortogonală** dacă $A \cdot A^T = A^T \cdot A = \mathbb{I}_n$.

Definiții generale. Grupul ortogonal

Definiție (i) O matrice pătratică $A \in \mathcal{M}_n(\mathbb{R})$ se numește **ortogonală** dacă $A \cdot A^T = A^T \cdot A = \mathbb{I}_n$.

Definiție (ii) $O(n) = \{A \in \mathcal{M}_n(\mathbb{R}) \mid A \cdot A^T = A^T \cdot A = \mathbb{I}_n\}$.

Definiții generale. Grupul ortogonal

Definiție (i) O matrice pătratică $A \in \mathcal{M}_n(\mathbb{R})$ se numește **ortogonală** dacă $A \cdot A^T = A^T \cdot A = \mathbb{I}_n$.

Definiție (ii) $O(n) = \{A \in \mathcal{M}_n(\mathbb{R}) \mid A \cdot A^T = A^T \cdot A = \mathbb{I}_n\}$.

Observații.

Definiții generale. Grupul ortogonal

Definiție (i) O matrice pătratică $A \in \mathcal{M}_n(\mathbb{R})$ se numește **ortogonală** dacă $A \cdot A^T = A^T \cdot A = \mathbb{I}_n$.

Definiție (ii) $O(n) = \{A \in \mathcal{M}_n(\mathbb{R}) \mid A \cdot A^T = A^T \cdot A = \mathbb{I}_n\}$.

Observații.

- (a) $(O(n), \cdot)$ este grup: **grupul ortogonal de ordinul n** .
- (b) $A \in O(n) \Rightarrow \det A \in \{\pm 1\}$, după cum păstrează sau schimbă orientarea, pentru $\det A = 1$, respectiv $\det A = -1$.

Definiții generale. Grupul ortogonal

Definiție (i) O matrice pătratică $A \in \mathcal{M}_n(\mathbb{R})$ se numește **ortogonală** dacă $A \cdot A^T = A^T \cdot A = \mathbb{I}_n$.

Definiție (ii) $O(n) = \{A \in \mathcal{M}_n(\mathbb{R}) \mid A \cdot A^T = A^T \cdot A = \mathbb{I}_n\}$.

Observații.

- (a) $(O(n), \cdot)$ este grup: **grupul ortogonal de ordinul n** .
- (b) $A \in O(n) \Rightarrow \det A \in \{\pm 1\}$, după cum păstrează sau schimbă orientarea, pentru $\det A = 1$, respectiv $\det A = -1$.

Definiție (iii) $SO(n) = \{A \in O(n) \mid \det A = 1\}$. Se numește **grupul special ortogonal de ordinul n** .

Definiții generale. Grupul ortogonal

Definiție (i) O matrice pătratică $A \in \mathcal{M}_n(\mathbb{R})$ se numește **ortogonală** dacă $A \cdot A^T = A^T \cdot A = \mathbb{I}_n$.

Definiție (ii) $O(n) = \{A \in \mathcal{M}_n(\mathbb{R}) \mid A \cdot A^T = A^T \cdot A = \mathbb{I}_n\}$.

Observații.

- (a) $(O(n), \cdot)$ este grup: **grupul ortogonal de ordinul n** .
- (b) $A \in O(n) \Rightarrow \det A \in \{\pm 1\}$, după cum păstrează sau schimbă orientarea, pentru $\det A = 1$, respectiv $\det A = -1$.

Definiție (iii) $SO(n) = \{A \in O(n) \mid \det A = 1\}$. Se numește **grupul special ortogonal de ordinul n** .

Observații.

- (c) $SO(n)$ este subgrup al lui $O(n)$.

2. Rotații 2D și grupul $SO(2)$

- ▶ Am văzut că unei rotații R_θ de unghi θ îi corespunde o matrice M_{R_θ} din $SO(2)$.

2. Rotații 2D și grupul $SO(2)$

- ▶ Am văzut că unei rotații R_θ de unghi θ îi corespunde o matrice M_{R_θ} din $SO(2)$.
- ▶ Și reciproc este adevărat: se poate arăta că orice matrice $A \in SO(2)$ corespunde unei rotații de unghi convenabil.

2. Rotații 2D și grupul $SO(2)$

- ▶ Am văzut că unei rotații R_θ de unghi θ îi corespunde o matrice M_{R_θ} din $SO(2)$.
- ▶ Și reciproc este adevărat: se poate arăta că orice matrice $A \in SO(2)$ corespunde unei rotații de unghi convenabil.
- ▶ Grupul \mathcal{R}_{2D} al rotațiilor 2D este izomorf cu un grup de matrice

$$(\mathcal{R}_{2D}, \circ) \simeq (SO(2), \cdot).$$

2. Rotații 2D și numere complexe

- Rotațiile 2D pot fi interpretate cu ajutorul numerelor complexe:

$$(\cos \theta, \sin \theta) \equiv \cos \theta + i \sin \theta = e^{i\theta}.$$

2. Rotații 2D și numere complexe

- ▶ Rotațiile 2D pot fi interpretate cu ajutorul numerelor complexe:

$$(\cos \theta, \sin \theta) \equiv \cos \theta + i \sin \theta = e^{i\theta}.$$

- ▶ **Cercul / sfera 1-dimensională**

$$S^1 = \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 = 1\} = \{z \in \mathbb{C} \mid |z| = 1\}.$$

2. Rotații 2D și numere complexe

- ▶ Rotațiile 2D pot fi interpretate cu ajutorul numerelor complexe:

$$(\cos \theta, \sin \theta) \equiv \cos \theta + i \sin \theta = e^{i\theta}.$$

- ▶ **Cercul / sfera 1-dimensională**

$$S^1 = \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 = 1\} = \{z \in \mathbb{C} \mid |z| = 1\}.$$

- ▶ (S^1, \cdot) este grup.

2. Rotații 2D și numere complexe

- Rotațiile 2D pot fi interpretate cu ajutorul numerelor complexe:

$$(\cos \theta, \sin \theta) \equiv \cos \theta + i \sin \theta = e^{i\theta}.$$

- **Cercul / sfera 1-dimensională**

$$S^1 = \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 = 1\} = \{z \in \mathbb{C} \mid |z| = 1\}.$$

- (S^1, \cdot) este grup.
- Avem izomorfisme naturale

$$(\mathcal{R}_{2D}, \circ) \simeq (SO(2), \cdot) \simeq (S^1, \cdot).$$

$$R_\theta \leftrightarrow \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \leftrightarrow \cos \theta + i \sin \theta.$$

Remember: corpul \mathbb{C} al numerelor complexe

Construcție: Se consideră mulțimea \mathbb{R}^2 , înzestrată cu două operații:

$$“+”: (a, b) + (a', b') = (a + a', b + b')$$

$$“\cdot”: (a, b) \cdot (a', b') = (aa' - bb', ab' + a'b)$$

În raport cu cele două operații se obține un corp comutativ.

Remember: corpul \mathbb{C} al numerelor complexe

Construcție: Se consideră mulțimea \mathbb{R}^2 , înzestrată cu două operații:

$$“+”: (a, b) + (a', b') = (a + a', b + b')$$

$$“\cdot”: (a, b) \cdot (a', b') = (aa' - bb', ab' + a'b)$$

În raport cu cele două operații se obține un corp comutativ.

Notății:

$$1 \equiv (1, 0), \quad i \equiv (0, 1)$$

și folosind aceste notații orice pereche (a, b) se reprezintă sub forma $a + ib$.

Are loc relația fundamentală $i^2 = -1$.

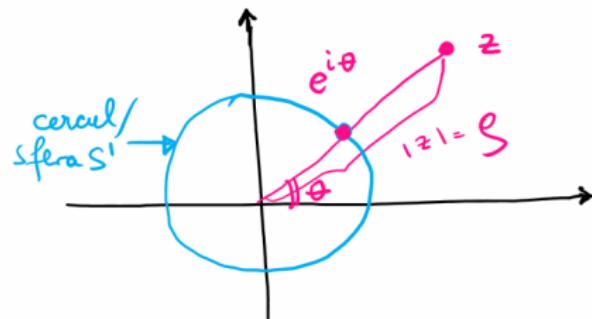
Corpul $(\mathbb{C}, +, \cdot)$.

Remember: corpul \mathbb{C} al numerelor complexe

Proprietăți și notații

- (i) Pentru $z = a + ib \in \mathbb{C}$, modulul lui z este $|z| = \sqrt{a^2 + b^2}$
- (ii) Dacă $z = a + ib \neq 0$, are loc relația $z^{-1} = \frac{\bar{z}}{|z|^2}$, unde $\bar{z} = a - ib$ este conjugatul lui z
- (iii) Orice număr complex $z \neq 0$ se scrie în mod unic sub forma

$$z = \rho(\cos \theta + i \sin \theta) = \rho e^{i\theta}, \quad \rho = |z|.$$



3. Rotații 3D - generalități

Observație 1.

A indica o rotație 3D \Leftrightarrow

\Leftrightarrow a indica o schimbare de repere ortonormate cu păstrarea orientării

\Leftrightarrow a indica o matrice din grupul $\text{SO}(3)$

De fapt

$$(\mathcal{R}_{3D}, \circ) \simeq (\text{SO}(3), \cdot).$$

3. Rotații 3D - generalități

Observație 1.

A indica o rotație 3D \Leftrightarrow

\Leftrightarrow a indica o schimbare de repere ortonormate cu păstrarea orientării

\Leftrightarrow a indica o matrice din grupul $\text{SO}(3)$

De fapt

$$(\mathcal{R}_{3D}, \circ) \simeq (\text{SO}(3), \cdot).$$

Observație 2.

Orice matrice $A \in \text{SO}(3)$ (i.e. orice rotație în context 3D) admite **o valoare proprie reală și un vector propriu (axă a rotației)**. De asemenea, rotația este caracterizată de **un unghi**, măsurat în planul perpendicular pe axă. Pentru rotația de unghi θ și axă (v_1, v_2, v_3) :

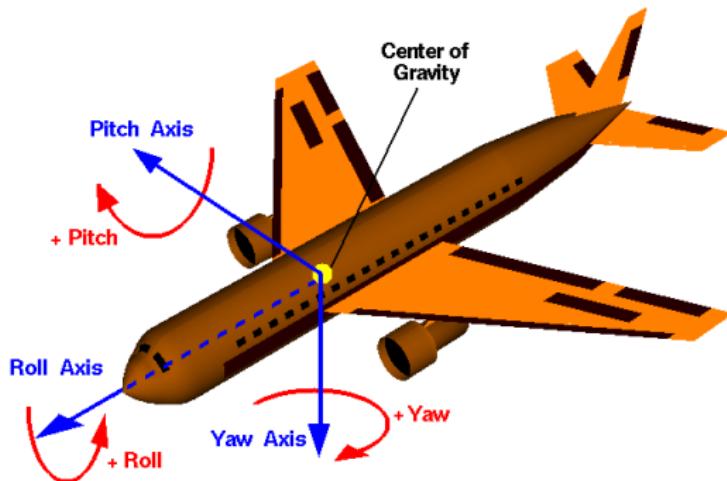
`glm :: rotate(theta, vec3(v1, v2, v3))`

3. Rotații 3D - problematizare: structura grupului $SO(3)$

Două posibilități:

- folosind unghiurile lui Euler
- folosind cuaternioni

3. Rotații 3D - unghiurile lui Euler: intuiție



Sursa: <https://upload.wikimedia.org/wikipedia/commons/7/7e/Rollpitchyawplain.png>

Altă reprezentare:

<https://upload.wikimedia.org/wikipedia/commons/8/85/Euler2a.gif>

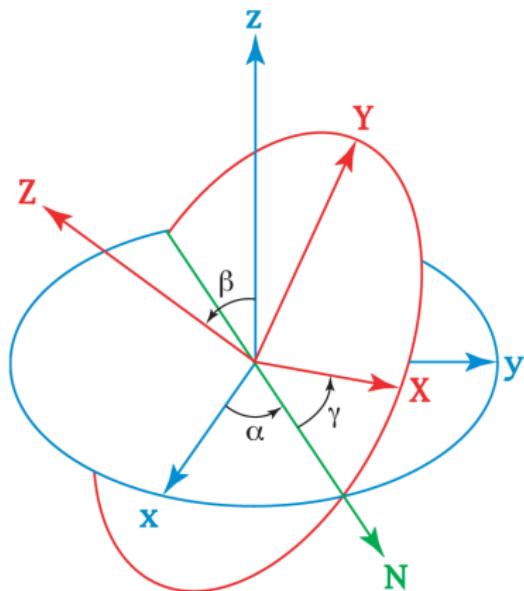
3. Rotații 3D - unghiurile lui Euler: intuiție

$$S^1 \times S^1 \times S^1 \longrightarrow SO(3)$$

se obține utilizând rotațiile în jurul
axelor de coordinate.

Fapt: orice matrice din $SO(3)$ poate fi
obținută ca produs al unor rotații (3)
în jurul axelor de coordinate,
în unghiuri alese convenabil
(unghiurile lui Euler)

3. Rotații 3D - unghiurile lui Euler: formalizare



Sursa: <https://upload.wikimedia.org/wikipedia/commons/8/82/Euler.png>

3. Rotații 3D - unghiurile lui Euler: Gimbal Lock

Ilustrare Gimbal Lock

Exemplu. $R(\alpha, (1,0,0))$ (rotație de unghi α
în axă $(1,0,0)$)

$$R\left(\frac{\pi}{2}, (0,1,0)\right)$$

$$R(\gamma, (0,0,1)) \quad (\alpha, \gamma \rightarrow \text{parametri})$$

compozitie (scriem matrice) \Rightarrow

matrice

$$\begin{pmatrix} 0 & 0 & 1 \\ \sin(\alpha + \gamma) & \cos(\alpha + \gamma) & 0 \\ -\cos(\alpha + \gamma) & \sin(\alpha + \gamma) & 0 \end{pmatrix}$$

"se pierde o
libertate
de mișcare"

(legată de proprietatea $S^1 \times S^1 \times S^1 \rightarrow SO(3)$)

3. Rotări 3D - reprezentare folosind cuaternioni

$$\mathbb{R}^4 = \{ (s, a, b, c) \mid s, a, b, c \in \mathbb{R} \}$$

???

$$\mathbb{H} = \left\{ \underbrace{s}_{\text{"parte reală"}} + \underbrace{a \cdot i + b \cdot j + c \cdot k}_{\text{"parte imaginară}} \mid s, a, b, c \in \mathbb{R} \right\}$$

i, j, k sunt definite prin

$$\begin{cases} i^2 = j^2 = k^2 = -1 \\ i \cdot j = -j \cdot i = k \\ j \cdot k = -k \cdot j = i \\ k \cdot i = -i \cdot k = j \end{cases}$$

Pe \mathbb{H} se defresc $\{ "+", "\cdot" \}$ pe componente
în dursă de regulile de mai sus.

Fapt: $(\mathbb{H}, +, \cdot)$ corp neicomutativ.

3. Rotații 3D - reprezentare folosind cuaternioni

Observație. Fie

$$\mathcal{H} = \left\{ M \in \mathcal{M}_2(\mathbb{C}) \mid M = s \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + a \begin{pmatrix} i & 0 \\ 0 & i \end{pmatrix} + b \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} + c \begin{pmatrix} 0 & i \\ i & 0 \end{pmatrix}, \quad s, a, b, c \in \mathbb{R} \right\}.$$

Au loc relațiile

$$\left(\begin{pmatrix} i & 0 \\ 0 & i \end{pmatrix} \right)^2 = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}, \quad \left(\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \right)^2 = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}, \quad \left(\begin{pmatrix} 0 & i \\ i & 0 \end{pmatrix} \right)^2 = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}.$$

De fapt, $(\mathcal{H}, +, \cdot) \cong (\mathbb{H}, +, \cdot)$.

3. Rotări 3D - reprezentare folosind cuaternioni

Notatie. Fie $q = s + ai + bj + ck = (s, \mathbf{v})$, unde $\mathbf{v} = (a, b, c)$

Convenția notării:

(i) înmulțirea este

$$q \cdot q' = (s \cdot s' - \mathbf{v} \cdot \mathbf{v}', s \cdot \mathbf{v}' + s' \cdot \mathbf{v} + \mathbf{v} \times \mathbf{v}')$$

$$(ii) |q|^2 = s^2 + \|\mathbf{v}\|^2$$

$$(iii) \text{ Pătratul } q \neq 0. \quad q^{-1} = \frac{\bar{q}}{|q|^2}, \quad \bar{q} = (s, -\mathbf{v})$$

Notatie $S^3 = \{q \in \mathbb{H} \mid |q| = 1\}$.

3. Rotări 3D - reprezentare folosind cuaternioni

Propoziție. (legătura dintre rotații 3D și cuaternioni)

(i) Fie rotația 3D având axa dată de vesorul u și unghiul θ . Fie cuaternionul $q \in S^3$ dat de

$$q = (s, v), \quad \begin{cases} s = \cos \frac{\theta}{2} \in \mathbb{R} \\ v = \sin \frac{\theta}{2} u \in \mathbb{R}^3 \end{cases}$$

Fie $P \in \mathbb{R}^3$ și P' punctul obținut aplicând rotația de unghi θ și axă u lui P , adică

$$P' = R_{u,\theta}(P).$$

Atunci în \mathbb{H} are loc relația

$$(0, P') = q \cdot (0, P) \cdot q^{-1}.$$

Altfel spus, pentru a determina P' efectuăm în \mathbb{H} calculul $q \cdot (0, P) \cdot q^{-1}$ și rezultatul ne va conduce la cuaternionul $(0, P')$.

(ii) Fie $q = s + ai + bj + cK \in S^3$ un cuaternion de normă 1. El corespunde unei rotații având matricea 3×3

$$\begin{pmatrix} s^2 + a^2 - b^2 - c^2 & 2ab - 2sc & 2ac + 2sb \\ 2ab + 2sc & s^2 - a^2 + b^2 - c^2 & 2bc - 2sa \\ 2ac - 2sb & 2bc + 2sa & s^2 - a^2 - b^2 + c^2 \end{pmatrix}.$$

3. Rotații 3D - reprezentare folosind cuaternioni

Exemplu. Considerăm rotația $R_{u,\theta}$ dată de vectorul $u = \left(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right)$ și de unghi $\theta = \frac{2\pi}{3}(120^\circ)$. De exemplu, avem $R_{u,\theta}(1, 0, 0) = (0, 1, 0)$. Interpretarea acestei relații folosind cuaternioni este următoarea.

(i) Vectorul u se scrie cu cuaternioni $u = \left(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right) \equiv \frac{1}{\sqrt{3}}i + \frac{1}{\sqrt{3}}j + \frac{1}{\sqrt{3}}k$. Conform propoziției anterioare, rotației $R_{u,\theta}$ i se asociază cuaternionul

$$q = \cos \frac{\theta}{2} + \sin \frac{\theta}{2} u = \cos \frac{2\pi}{6} + \sin \frac{2\pi}{6} \left(\frac{1}{\sqrt{3}}i + \frac{1}{\sqrt{3}}j + \frac{1}{\sqrt{3}}k \right) = \dots$$

Prin calcul, se deduce

$$q = \frac{1}{2}(1 + i + j + k), \quad q^{-1} = \frac{1}{2}(1 - i - j - k).$$

(ii) Versonii axelor Ox, Oy, Oz corespund cuaternionilor i, j, k , deci $R_{u,\theta}(1, 0, 0) = (0, 1, 0)$ se rescrie $R_{u,\theta}(i) = j$. Se poate verifica faptul că are loc relația $(0, j) = q \cdot (0, i) \cdot q^{-1}$, care este exact rescrierea din propoziția anterioară (cu $P = (1, 0, 0) \equiv i, P' = (0, 1, 0) \equiv j$).

3. Rotații 3D - reprezentare folosind cuaternioni

Alte detalii teoretice și despre implementare:

K. Shoemake, Quaternions

<https://www.cprogramming.com/tutorial/3d/quaternions.html>