

Grafica pe calculator. Preliminarii

Mihai-Sorin Stupariu

Sem. I, 2024 - 2025

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*)

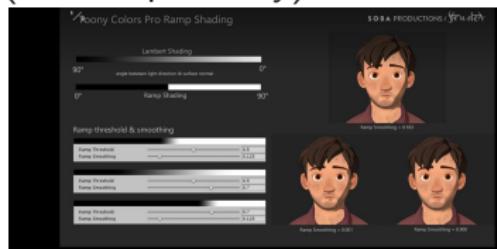


Figura: Sursa: <https://unity.com/madewith/sonder>

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, M. Baker, W. Carithers, *Computer Graphics with OpenGL* (4th edition), Pearson Education, 2015.
- J. Kessenich, G. Sellers, D. Shreiner, *The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V* (9th edition), Addison Wesley, 2016.
- 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 ed.), Addison-Wesley, 2015.
- V. Scott Gordon, J. Clevenger, *Computer Graphics Programming in OpenGL* (3rd edition), Mercury Learning and Information, 2024.

Resurse

- ▶ Site-ul oficial OpenGL
- ▶ Cărți:
 - D. Hearn, M. Baker, W. Carithers, *Computer Graphics with OpenGL* (4th edition), Pearson Education, 2015.
 - J. Kessenich, G. Sellers, D. Shreiner, *The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V* (9th edition), Addison Wesley, 2016.
 - 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 ed.), Addison-Wesley, 2015.
 - V. Scott Gordon, J. Clevenger, *Computer Graphics Programming in OpenGL* (3rd edition), Mercury Learning and Information, 2024.
- ▶ Cărți / tutoriale online:
 - <http://learnopengl.com/>
 - <http://www.opengl-tutorial.org/>
 - D. Eck, *Introduction to Computer Graphics*
 - <http://nehe.gamedev.net/>
 - A. Gerdelen - *OpenGL 4 tutorials*;
 - etc.



Visual Studio |



Mediu de dezvoltare /
limbaj de programare

API grafica - biblioteci / unelte



Placa grafica

Elemente generale

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

Elemente generale

- ▶ OpenGL este o interfață de programare (Application Programming Interface API).
- ▶ OpenGL **nu** este un limbaj de programare (există GLSL - limbaj pentru unitățile programabile de tip *shader*).

Elemente generale

- ▶ OpenGL este o interfață de programare (Application Programming Interface API).
- ▶ OpenGL **nu** este un limbaj de programare (există GLSL - limbaj pentru unitățile programabile de tip *shader*).
- ▶ OpenGL generează un buffer de cadru (*frame buffer*), apoi este realizat transferul conținutului buffer-ului pe ecran. Pentru a funcționa are nevoie de o serie de biblioteci (exemple: *GLUT*, **freeglut**, *GLFW*) - gestionarea ferestrelor. Este necesară și o bibliotecă pentru gestionarea noilor funcționalități - *extension library* (exemple: **GLEW**, *GL3W*, *GLAD*).

Elemente generale

- ▶ OpenGL este o interfață de programare (Application Programming Interface API).
- ▶ OpenGL **nu** este un limbaj de programare (există GLSL - limbaj pentru unitățile programabile de tip *shader*).
- ▶ OpenGL generează un buffer de cadru (*frame buffer*), apoi este realizat transferul conținutului buffer-ului pe ecran. Pentru a funcționa are nevoie de o serie de biblioteci (exemple: *GLUT*, **freeglut**, *GLFW*) - gestionarea ferestrelor. Este necesară și o bibliotecă pentru gestionarea noilor funcționalități - *extension library* (exemple: **GLEW**, *GL3W*, *GLAD*).
- ▶ Alte biblioteci: funcții matematice (**glm**) și texturi (**SOIL**).

Elemente generale

- ▶ OpenGL este o interfață de programare (Application Programming Interface API).
- ▶ OpenGL **nu** este un limbaj de programare (există GLSL - limbaj pentru unitățile programabile de tip *shader*).
- ▶ OpenGL generează un buffer de cadru (*frame buffer*), apoi este realizat transferul conținutului buffer-ului pe ecran. Pentru a funcționa are nevoie de o serie de biblioteci (exemple: *GLUT*, **freeglut**, *GLFW*) - gestionarea ferestrelor. Este necesară și o bibliotecă pentru gestionarea noilor funcționalități - *extension library* (exemple: **GLEW**, *GL3W*, *GLAD*).
- ▶ Alte biblioteci: funcții matematice (**glm**) și texturi (**SOIL**).
- ▶ Laborator: **template** care are integrate aceste biblioteci.

Elemente generale

- ▶ OpenGL este o interfață de programare (Application Programming Interface API).
- ▶ OpenGL **nu** este un limbaj de programare (există GLSL - limbaj pentru unitățile programabile de tip *shader*).
- ▶ OpenGL generează un buffer de cadru (*frame buffer*), apoi este realizat transferul conținutului buffer-ului pe ecran. Pentru a funcționa are nevoie de o serie de biblioteci (exemple: *GLUT*, **freeglut**, *GLFW*) - gestionarea ferestrelor. Este necesară și o bibliotecă pentru gestionarea noilor funcționalități - *extension library* (exemple: **GLEW**, *GL3W*, *GLAD*).
- ▶ Alte biblioteci: funcții matematice (**glm**) și texturi (**SOIL**).
- ▶ Laborator: **template** care are integrate aceste biblioteci.
- ▶ Arhitectura de tip *client-server* (CPU-GPU) - element cheie: “comunicarea” dintre cele două componente *hardware*.

Elemente generale

- ▶ OpenGL este o interfață de programare (Application Programming Interface API).
- ▶ OpenGL **nu** este un limbaj de programare (există GLSL - limbaj pentru unitățile programabile de tip *shader*).
- ▶ OpenGL generează un buffer de cadru (*frame buffer*), apoi este realizat transferul conținutului buffer-ului pe ecran. Pentru a funcționa are nevoie de o serie de biblioteci (exemple: *GLUT*, **freeglut**, *GLFW*) - gestionarea ferestrelor. Este necesară și o bibliotecă pentru gestionarea noilor funcționalități - *extension library* (exemple: **GLEW**, *GL3W*, *GLAD*).
- ▶ Alte biblioteci: funcții matematice (**glm**) și texturi (**SOIL**).
- ▶ Laborator: **template** care are integrate aceste biblioteci.
- ▶ Arhitectura de tip *client-server* (CPU-GPU) - element cheie: “comunicarea” dintre cele două componente *hardware*.
- ▶ Versiunea suportată depinde de placa grafică a calculatorului - se poate folosi codul sursă `01_00_test_version.cpp`.

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/freeglut 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/freeglut 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/freeglut 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_varfuri_triunghi_OLD.cpp

// Directive preprocesare

Exemplu de program care utilizează OpenGL “vechi”

Codul sursă 01_01_varfuri_triunghi_OLD.cpp

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

Exemplu de program care utilizează OpenGL “vechi”

Codul sursă 01_01_varfuri_triunghi_OLD.cpp

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

Exemplu de program care utilizează OpenGL “vechi”

Codul sursă 01_01_varfuri_triunghi_OLD.cpp

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

Exemplu de program care utilizează OpenGL “vechi”

Codul sursă 01_01_varfuri_triunghi_OLD.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_varfuri_triunghi.cpp

// Directive preprocessare

Exemplu de program care utilizează OpenGL “nou”

Codul sursă 01_02_varfuri_triunghi.cpp

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

Exemplu de program care utilizează OpenGL “nou”

Codul sursă 01_02_varfuri_triunghi.cpp

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

Exemplu de program care utilizează OpenGL “nou”

Codul sursă 01_02_varfuri_triunghi.cpp

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

Exemplu de program care utilizează OpenGL “nou”

Codul sursă 01_02_varfuri_triunghi.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_varfuri_triunghi.cpp

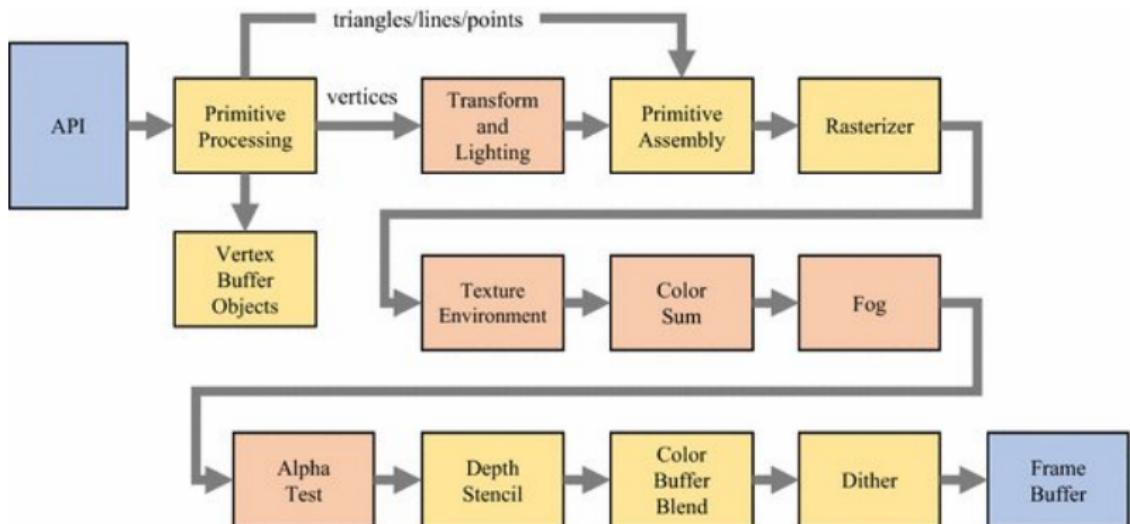
```
// Directive preprocessare
// Shader-e, date (coordonate, culori)
// Proceduri (inițializare, desen)
// Main
    // Inițializări GLUT
    // Generare fereastră
    // Apelare procedură inițializare
        // Creare VBO / VAO
        // Creare / Apelare shader-e
    // Apelare procedură desen
```

Exemplu de program care utilizează OpenGL “nou”

Codul sursă 01_02_varfuri_triunghi.cpp

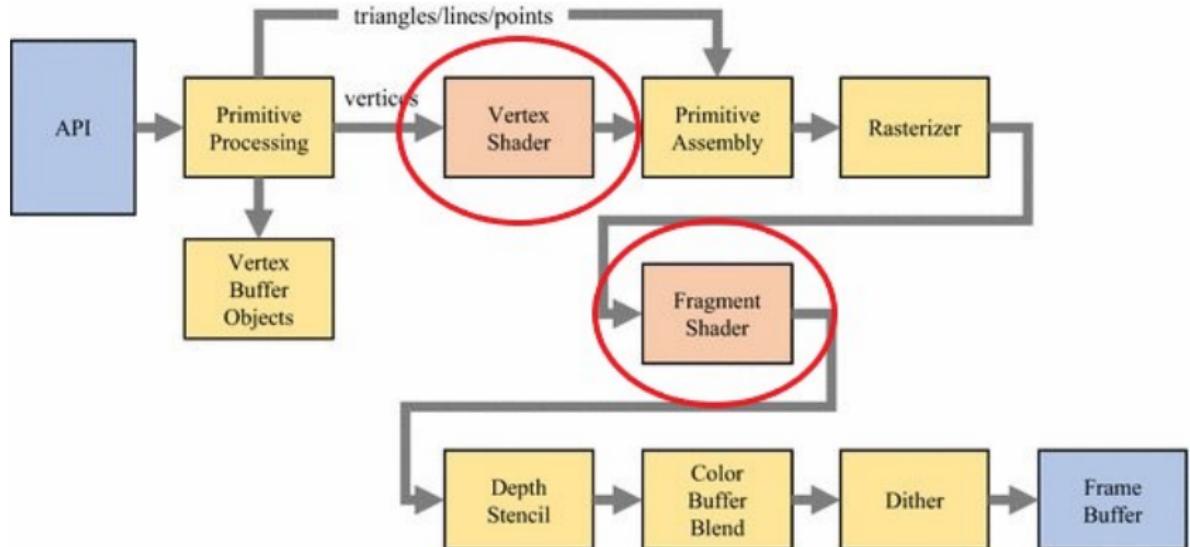
```
// Directive preprocessare
// Shader-e, date (coordonate, culori)
// Proceduri (inițializare, desen)
// Main
    // Inițializări GLUT
    // Generare fereastră
    // Apelare procedură inițializare
        // Creare VBO / VAO
        // Creare / Apelare shader-e
    // Apelare procedură desen
  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

Culori, vârfuri, primitive grafice. Program principal și *shader*-e

Mihai-Sorin Stupariu

Sem. I, 2024 - 2025

Spațiul culorilor

Programul principal: vârfuri

Programul principal: primitive grafice - generalități

Shader-e

Comunicarea dintre programul principal și shader-e

Suport teoretic: 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

- ▶ 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) ($f, d \in [0.0, 1.0]$)
- (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 trase 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)**.
- O serie întreagă de funcții asociate (exemple mai jos, v. și slide-uri):

```
// Se creeaza un buffer pentru VÂRFURI;
glGenBuffers(1, &VboId);
glBindBuffer(GL_ARRAY_BUFFER, VboId);
glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL_STATIC_DRAW);
// Se asociaza atributul (0 = coordonate) pentru shader;
glEnableVertexAttribArray(0);
 glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, 0);
```

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 Colori...),
 - ▶ 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*.

Caracteristici ale vârfurilor

- ▶ Unui vârf îi sunt asociate:
 - ▶ coordonate (fac parte din definiție),
 - ▶ o culoare (v. secțiunea Colori...),
 - ▶ 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.
- ▶ În OpenGL “modern”: pentru o anumită caracteristică, este generat un *buffer object*, la fel ca pentru coordonatele vârfurilor. Alternativ, caracteristica respectivă poate fi indicată în aceeași matrice cu coordonatele vârfurilor.

Caracteristici ale vârfurilor

- ▶ Unui vârf îi sunt asociate:
 - ▶ coordonate (fac parte din definiție),
 - ▶ o culoare (v. secțiunea Colori...),
 - ▶ 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.
- ▶ În OpenGL “modern”: pentru o anumită caracteristică, este generat un *buffer object*, la fel ca pentru coordonatele vârfurilor. Alternativ, caracteristica respectivă poate fi indicată în aceeași matrice cu coordonatele vârfurilor.
- ▶ Caracteristicile vârfurilor, indicate în programul principal, se regăsesc (sub un format adecvat) și în *shader-ul de vârfuri*.

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);

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);

- 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 (corespund lui GLenum mode)

- ▶ Puncte: GL_POINTS

Tipuri de primitive (corespund lui GLenum mode)

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

Tipuri de primitive (corespund lui GLenum mode)

- ▶ 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 (corespund lui GLenum mode)

- ▶ 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 (corespund lui GLenum mode)

- ▶ 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

Despre GLSL și obiecte-program

- **GLSL** este un limbaj de nivel înalt, cu sintaxă asemănătoare limbajului C: detalii în **specificațiile GLSL** - sunt prezentate funcțiile standard utilizate.

Despre GLSL și obiecte-program

- ▶ **GLSL** este un limbaj de nivel înalt, cu sintaxă asemănătoare limbajului C: detalii în **specificațiile GLSL** - sunt prezentate funcțiile standard utilizate.
- ▶ Avantaje: utilizare pe platforme multiple, compatibilitate cu diverse tipuri de plăci grafice.
- ▶ Motivație: de a controla banda grafică (*graphics pipeline*). Finalitate: obținerea unui obiect-program (*Program Object*) care să fie rulat pe placă grafică.

Despre GLSL și obiecte-program

- ▶ **GLSL** este un limbaj de nivel înalt, cu sintaxă asemănătoare limbajului C: detalii în **specificațiile GLSL** - sunt prezentate funcțiile standard utilizate.
- ▶ Avantaje: utilizare pe platforme multiple, compatibilitate cu diverse tipuri de plăci grafice.
- ▶ Motivație: de a controla banda grafică (*graphics pipeline*). Finalitate: obținerea unui obiect-program (*Program Object*) care să fie rulat pe placa grafică.
- ▶ Pentru a genera obiectul-program, trebuie parcurse o serie de etape prin intermediul programului principal - aplicația instalează codul GLSL pe placa grafică. Practic: v. *slide*-ul următor.

Despre GLSL și obiecte-program

- ▶ **GLSL** este un limbaj de nivel înalt, cu sintaxă asemănătoare limbajului C: detalii în **specificațiile GLSL** - sunt prezentate funcțiile standard utilizate.
- ▶ Avantaje: utilizare pe platforme multiple, compatibilitate cu diverse tipuri de plăci grafice.
- ▶ Motivație: de a controla banda grafică (*graphics pipeline*). Finalitate: obținerea unui obiect-program (*Program Object*) care să fie rulat pe placa grafică.
- ▶ Pentru a genera obiectul-program, trebuie parcurse o serie de etape prin intermediul programului principal - aplicația instalează codul GLSL pe placa grafică. Practic: v. *slide*-ul următor.
- ▶ Codul este efectiv rulat la apelarea `glDrawArrays()` sau a unei funcții de desenare similară.

Despre shader-e

- Manevrare: în *template*-ul pus la dispoziție, etapele specifice sunt incluse în funcția

```
GLuint LoadShaders(const char *vertex_file_path,const char *fragment_file_path)  
din fișierul LoadShaders.cpp
```

Ca date de intrare: calea către *shader*-ele de vârfuri, respectiv fragment. Funcția returnează un identificator al obiectului-program generat.

Despre shader-e

- Manevrare: în *template*-ul pus la dispoziție, etapele specifice sunt incluse în funcția

```
GLuint LoadShaders(const char *vertex_file_path,const char *fragment_file_path)  
din fișierul LoadShaders.cpp
```

Ca date de intrare: calea către *shader*-ele de vârfuri, respectiv fragment. Funcția returnează un identificator al obiectului-program generat.

- Fluxul:

Despre shader-e

- ▶ Manevrare: în *template*-ul pus la dispoziție, etapele specifice sunt incluse în funcția

```
GLuint LoadShaders(const char *vertex_file_path,const char *fragment_file_path)  
din fișierul LoadShaders.cpp
```

Ca date de intrare: calea către *shader*-ele de vârfuri, respectiv fragment. Funcția returnează un identificator al obiectului-program generat.

- ▶ Fluxul:
 - (dacă este cazul) obținerea codului din fișier și transformarea în *string*

Despre shader-e

- Manevrare: în *template*-ul pus la dispoziție, etapele specifice sunt incluse în funcția

```
GLuint LoadShaders(const char *vertex_file_path,const char *fragment_file_path)  
din fișierul LoadShaders.cpp
```

Ca date de intrare: calea către *shader*-ele de vârfuri, respectiv fragment. Funcția returnează un identificator al obiectului-program generat.

- Fluxul:
 - (dacă este cazul) obținerea codului din fișier și transformarea în *string*
 - crearea unui obiect *shader* `glCreateShader()`, încărcarea *string*-ului în obiectul *shader* `glShaderSource()` și compilarea `glCompileShader()` - pas realizat separat pentru fiecare tip de *shader*

Despre shader-e

- Manevrare: în *template*-ul pus la dispoziție, etapele specifice sunt incluse în funcția

```
GLuint LoadShaders(const char *vertex_file_path,const char *fragment_file_path)  
din fișierul LoadShaders.cpp
```

Ca date de intrare: calea către *shader*-ele de vârfuri, respectiv fragment. Funcția returnează un identificator al obiectului-program generat.

- Fluxul:
 - (dacă este cazul) obținerea codului din fișier și transformarea în *string*
 - crearea unui obiect *shader* `glCreateShader()`, încărcarea *string*-ului în obiectul *shader* `glShaderSource()` și compilarea `glCompileShader()` - pas realizat separat pentru fiecare tip de *shader*
 - crearea unui obiect-program `glCreateProgram()`, atașarea obiectelor *shader* compilate `glAttachShader()` și legarea programului `glLinkProgram()`

Despre shader-e

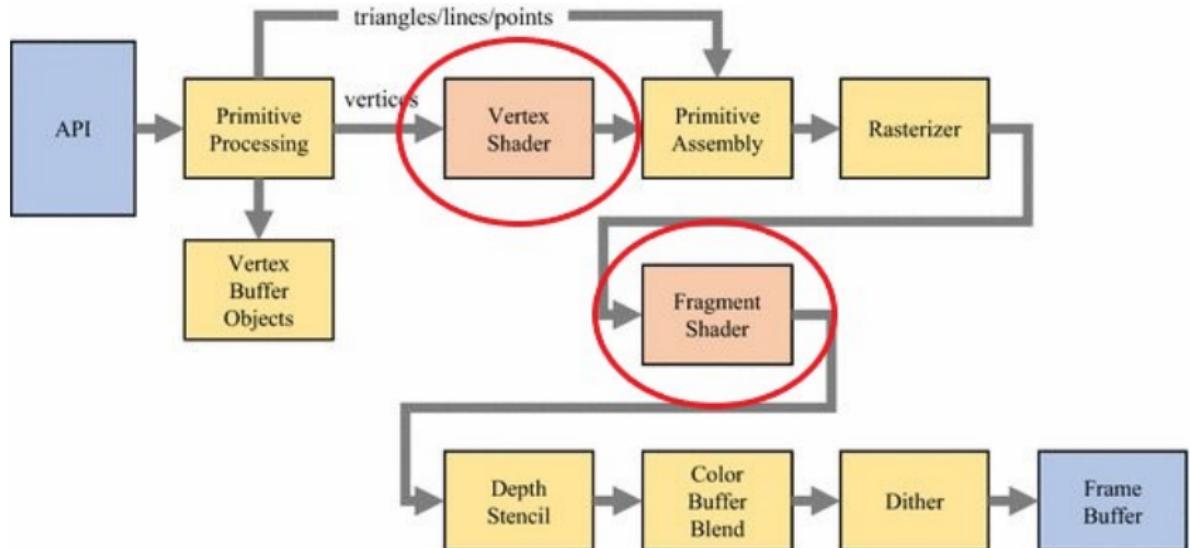
- Manevrare: în *template*-ul pus la dispoziție, etapele specifice sunt incluse în funcția

```
GLuint LoadShaders(const char *vertex_file_path,const char *fragment_file_path)  
din fișierul LoadShaders.cpp
```

Ca date de intrare: calea către *shader*-ele de vârfuri, respectiv fragment. Funcția returnează un identificator al obiectului-program generat.

- Fluxul:
 - (dacă este cazul) obținerea codului din fișier și transformarea în *string*
 - crearea unui obiect *shader* `glCreateShader()`, încărcarea *string*-ului în obiectul *shader* `glShaderSource()` și compilarea `glCompileShader()` - pas realizat separat pentru fiecare tip de *shader*
 - crearea unui obiect-program `glCreateProgram()`, atașarea obiectelor *shader* compilate `glAttachShader()` și legarea programului `glLinkProgram()`
 - utilizarea obiectului-program `glUseProgram()` - poate fi apelată inclusiv în funcția de desenare.

“Programmable pipeline”



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

Shader-ul de vârfuri (*vertex shader*)

- ▶ Rol: procesează vârfurile (și informațiile despre acestea) și le trimită mai departe în banda grafică.

Shader-ul de vârfuri (*vertex shader*)

- ▶ Rol: procesează vârfurile (și informațiile despre acestea) și le trimită mai departe în banda grafică.
- ▶ La apelarea unei funcții de desenare (de ex. `glDrawArrays()`) *shader-ul de vârfuri* procesează, unul câte unul (eventual în paralel) vârfurile din *VBO*.

Shader-ul de vârfuri (*vertex shader*)

- ▶ Rol: procesează vârfurile (și informațiile despre acestea) și le trimită mai departe în banda grafică.
- ▶ La apelarea unei funcții de desenare (de ex. `glDrawArrays()`) *shader-ul de vârfuri* procesează, unul câte unul (eventual în paralel) vârfurile din *VBO*.
- ▶ Date de intrare / ieșire: informații despre vârfuri (poziție, culoare, etc.) și o serie de alte date (de ex. matrice ale transformărilor). În particular, în *shader-ul de vârfuri* pot fi aplicate diverse transformări (inclusiv pentru mișcare) și poate fi controlată geometria scenei. De asemenea, mai există variabilele uniforme, preluate din programul principal.

Shader-ul de vârfuri (*vertex shader*)

- ▶ Rol: procesează vârfurile (și informațiile despre acestea) și le trimită mai departe în banda grafică.
- ▶ La apelarea unei funcții de desenare (de ex. `glDrawArrays()`) *shader-ul de vârfuri* procesează, unul câte unul (eventual în paralel) vârfurile din *VBO*.
- ▶ Date de intrare / ieșire: informații despre vârfuri (poziție, culoare, etc.) și o serie de alte date (de ex. matrice ale transformărilor). În particular, în *shader-ul de vârfuri* pot fi aplicate diverse transformări (inclusiv pentru mișcare) și poate fi controlată geometria scenei. De asemenea, mai există variabilele uniforme, preluate din programul principal.
- ▶ Datele furnizate de *shader-ul de vârfuri* trec printr-un proces de asamblare a primitivelor și de rasterizare, fiind transformate în fragmente (locații ale pixelilor) - proces realizat de API.

Shader-ul de vârfuri (*vertex shader*)

- ▶ Rol: procesează vârfurile (și informațiile despre acestea) și le trimită mai departe în banda grafică.
- ▶ La apelarea unei funcții de desenare (de ex. `glDrawArrays()`) *shader-ul de vârfuri* procesează, unul câte unul (eventual în paralel) vârfurile din *VBO*.
- ▶ Date de intrare / ieșire: informații despre vârfuri (poziție, culoare, etc.) și o serie de alte date (de ex. matrice ale transformărilor). În particular, în *shader-ul de vârfuri* pot fi aplicate diverse transformări (inclusiv pentru mișcare) și poate fi controlată geometria scenei. De asemenea, mai există variabilele uniforme, preluate din programul principal.
- ▶ Datele furnizate de *shader-ul de vârfuri* trec printr-un proces de asamblare a primitivelor și de rasterizare, fiind transformate în fragmente (locații ale pixelilor) - proces realizat de API.
- ▶ Mai multe detalii: ulterior (de ex. texturare, modele de iluminare, etc.)

Shader-ul de fragment (*fragment shader*)

- ▶ Rol: procesarea fragmentelor și stabilirea culorii pentru fiecare pixel din buffer-ul de cadru. Sunt realizate operații la nivel de fragment (de ex. amestecarea culorilor, combinarea cu texturile, etc.)

Shader-ul de fragment (*fragment shader*)

- ▶ Rol: procesarea fragmentelor și stabilirea culorii pentru fiecare pixel din buffer-ul de cadru. Sunt realizate operații la nivel de fragment (de ex. amestecarea culorilor, combinarea cu texturile, etc.)
- ▶ Date de intrare: provin din *shader-ul de vârfuri*. Date de ieșire: legate de culoarea fragmentelor. De asemenea, mai există variabilele uniforme, preluate din programul principal.

Shader-ul de fragment (*fragment shader*)

- ▶ Rol: procesarea fragmentelor și stabilirea culorii pentru fiecare pixel din buffer-ul de cadru. Sunt realizate operații la nivel de fragment (de ex. amestecarea culorilor, combinarea cu texturile, etc.)
- ▶ Date de intrare: provin din *shader-ul de vârfuri*. Date de ieșire: legate de culoarea fragmentelor. De asemenea, mai există variabilele uniforme, preluate din programul principal.
- ▶ Mai multe detalii: ulterior (de ex. texturare, modele de iluminare, etc.)

Elemente de principiu - comunicarea dintre progr. princ. și shader-e

Două moduri de comunicare

- ▶ Prin intermediul *buffer-elor* (vârfuri și caracteristici asociate) - C++ și al *vertex attributes - shader*.
- ▶ Folosind variabile uniforme.

Pentru vârfuri și caracteristici ale acestora - prog. princ.

- ▶ Pași realizați:

Pentru vârfuri și caracteristici ale acestora - prog. princ.

► Pași realizați:

- creare vectori cu vârfuri, caracteristici (culoare, etc.) - pot fi indicate separat sau în aceeași matrice, indici;
- generare nume ptr. buffer-objects: `glGenBuffers()`;
- activare/"legare" buffer `glBindBuffer()` și transfer de date în buffer: `glBufferData()`.

Pentru vârfuri și caracteristici ale acestora - prog. princ.

► Pași realizați:

- creare vectori cu vârfuri, caracteristici (culoare, etc.) - pot fi indicate separat sau în aceeași matrice, indici;
 - generare nume ptr. buffer-objects: `glGenBuffers()`;
 - activare/"legare" buffer `glBindBuffer()` și transfer de date în buffer: `glBufferData()`.
 - asociere cu un *vertex attribute* și indicarea locațiilor (ptr. *shader!*): `glVertexAttribPointer()` - parametrul GLuint index trebuie să se regăsească, cu aceeași valoare, și în *shader*,
 - activare a *vertex attribute*: `glEnableVertexAttribArray()` - parametrii GLuint index și GLint size trebuie să se regăsească, cu aceeași valoare, și în *shader*,
-
- Structura specifică pentru *buffer*: *VBO* (*Vertex Buffer Object*). Pot fi utilizate mai multe *VBO* (inclusiv pentru vârfuri).

Pentru vârfuri și caracteristici ale acestora - prog. princ.

► Pași realizați:

- creare vectori cu vârfuri, caracteristici (culoare, etc.) - pot fi indicate separat sau în aceeași matrice, indici;
- generare nume ptr. buffer-objects: `glGenBuffers()`;
- activare/"legare" buffer `glBindBuffer()` și transfer de date în buffer: `glBufferData()`.
- asociere cu un *vertex attribute* și indicarea locațiilor (ptr. *shader!*): `glVertexAttribPointer()` - parametrul GLuint index trebuie să se regăsească, cu aceeași valoare, și în *shader*,
- activare a *vertex attribute*: `glEnableVertexAttribArray()` - parametrii GLuint index și GLint size trebuie să se regăsească, cu aceeași valoare, și în *shader*,
- apelare funcție de desenare.

- Structura specifică pentru *buffer*: *VBO* (*Vertex Buffer Object*). Pot fi utilizate mai multe *VBO* (inclusiv pentru vârfuri).
- O structură asociată: *VAO* (*Vertex Array Object*) - are rolul de a permite gestionarea eficientă și organizarea *VBO* - necesar cel puțin un *VAO*. Funcții: `glGenVertexArrays`, `glBindVertexArray()`.

Pentru vârfuri și caracteristici ale acestora - *shader*

- În *shader-ul de vârfuri*:

Pentru vârfuri și caracteristici ale acestora - *shader*

► În *shader*-ul de vârfuri:

- Informațiile despre vârfuri sunt transferate în *shader* ca *vertex attributes*, aici devenind valori de intrare (*input*). Parametrii *index* și *size* din funcția `glEnableVertexAttribArray()` se regăsesc (cu aceleași valori) apelați în *location = index*, respectiv *vec**, unde **=size*. Exemplu:

```
layout (location = 0) in vec4 in_Position;  
layout (location = 1) in vec3 in_Color;
```

Indexul 0 corespunde unui vector cu 4 componente (coordonate).
Indexul 1 corespunde unui vector cu 3 componente (culori).

Pentru vârfuri și caracteristici ale acestora - *shader*

► În *shader*-ul de vârfuri:

- Informațiile despre vârfuri sunt transferate în *shader* ca *vertex attributes*, aici devenind valori de intrare (*input*). Parametrii *index* și *size* din funcția `glEnableVertexAttribArray()` se regăsesc (cu aceleași valori) apelați în *location = index*, respectiv *vec**, unde **=size*. Exemplu:

```
layout (location = 0) in vec4 in_Position;  
layout (location = 1) in vec3 in_Color;
```

Indexul 0 corespunde unui vector cu 4 componente (coordonate).

Indexul 1 corespunde unui vector cu 3 componente (culori).

- Trebuie să existe o corespondență perfectă între valorile din programul principal și cele din *shader*, altminteri efectul nu va fi cel dorit.

Variabile uniforme

- În programul principal:

Variabile uniforme

► În programul principal:

- obținerea unei referințe către variabila uniformă

```
glGetUniformLocation(GLuint program, const GLchar *name)
```

Variabila `program` și string-ul cu numele variabilei trebuie modificate.

Exemplu:

```
codColLocation = glGetUniformLocation(ProgramId,  
"codColShader");
```

Această funcție poate fi apelată și în `init`

Variabile uniforme

► În programul principal:

- obținerea unei referințe către variabila uniformă

```
glGetUniformLocation(GLuint program, const GLchar *name)
```

Variabila `program` și string-ul cu numele variabilei trebuie modificate.

Exemplu:

```
codColLocation = glGetUniformLocation(ProgramId,  
"codColShader");
```

Această funcție poate fi apelată și în `init`

- specificarea valorii variabilei `glUniform*`() Sufixul * indică informații despre variabilă. De asemenea, trebuie indicată locația utilizată.

Această funcție trebuie apelată la fiecare modificare a valorii variabilei.

Exemplu:

```
int codCol = 0;  
glUniform1i(codColLocation, codCol);
```

Variabile uniforme

► În programul principal:

- obținerea unei referințe către variabila uniformă

```
glGetUniformLocation(GLuint program, const GLchar *name)
```

Variabila `program` și string-ul cu numele variabilei trebuie modificate.

Exemplu:

```
codColLocation = glGetUniformLocation(ProgramId,  
"codColShader");
```

Această funcție poate fi apelată și în `init`

- specificarea valorii variabilei `glUniform*`() Sufixul * indică informații despre variabilă. De asemenea, trebuie indicată locația utilizată.

Această funcție trebuie apelată la fiecare modificare a valorii variabilei.

Exemplu:

```
int codCol = 0;  
glUniform1i(codColLocation, codCol);
```

► În `shader`:

Variabile uniforme

► În programul principal:

- obținerea unei referințe către variabila uniformă

`glGetUniformLocation(GLuint program, const GLchar *name)`

Variabila `program` și string-ul cu numele variabilei trebuie modificate.

Exemplu:

```
codColLocation = glGetUniformLocation(ProgramId,
"codColShader");
```

Această funcție poate fi apelată și în `init`

- specificarea valorii variabilei `glUniform*`() Sufixul * indică informații despre variabilă. De asemenea, trebuie indicată locația utilizată.

Această funcție trebuie apelată la fiecare modificare a valorii variabilei.

Exemplu:

```
int codCol = 0;
glUniform1i(codColLocation, codCol);
```

► În shader:

- declararea variabilei uniforme și a tipului acestora.

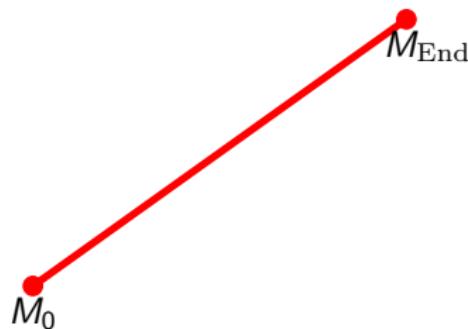
Exemplu:

```
uniform int codColShader
```

Motivație și problematizare

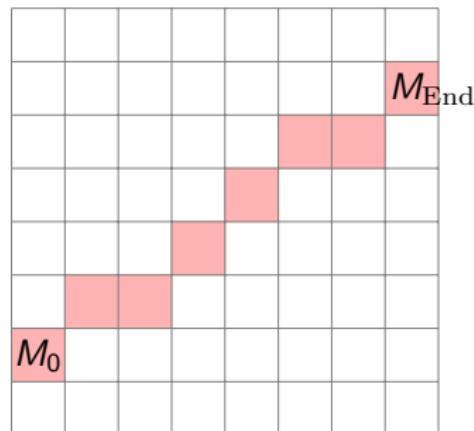
“Continuu”

“Grafica vectorială”



“Discret”

“Grafica rasterială”



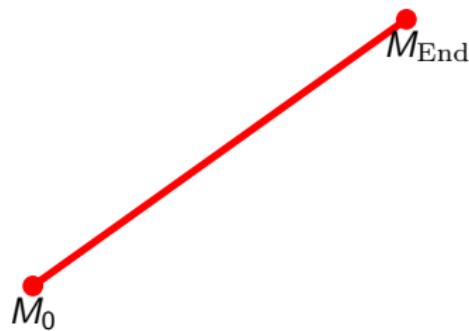
$$M_0 = (x_0, y_0), M_{\text{End}} = (x_{\text{End}}, y_{\text{End}}) \\ x_0, y_0, x_{\text{End}}, y_{\text{End}} \in \mathbb{R}$$

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

Motivație și problematizare

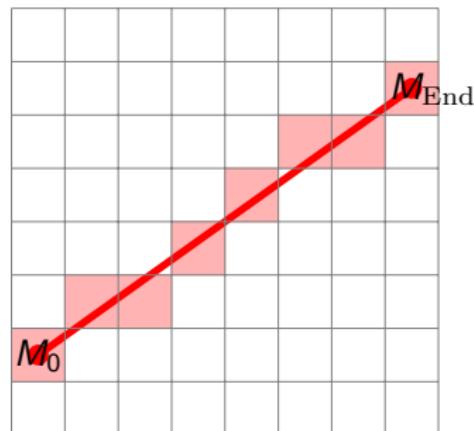
“Continuu”

“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:

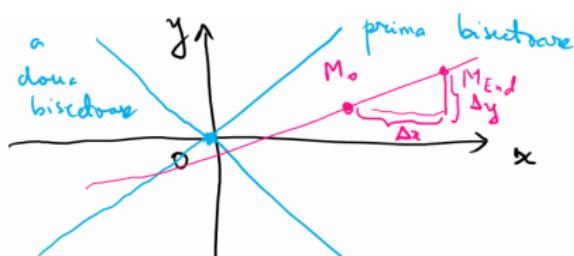
Ip. 1 Intuitiv: "deplasarea se face înspre dreapta/sus"

$$x_{\text{End}} > x_0, \quad y_{\text{End}} > y_0 \Leftrightarrow$$

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

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

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



Ecuăția dreptei

Ecuăția 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

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

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

(pct. $M_0(x_0, y_0)$ aparține dreptei)

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

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

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

(pct. $M_0(x_0, y_0)$ aparține dreptei)

Concluzie: În cazul continuu avem

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

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...

pentru $f(x_{k+1})$

Calcule...

pentru $f(x_{k+1})$

$$\begin{aligned}f(x_{k+1}) &= mx_{k+1} + n \stackrel{x_{k+1}=x_k+1}{=} m(x_k + 1) + n = \\&= mx_k + m + n = mx_k + n + m = f(x_k) + m.\end{aligned}$$

Varianta 2 - algoritmul Digital Differential Analyzer (DDA)

Algoritm

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

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

Exemplu

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

Exemplu

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

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

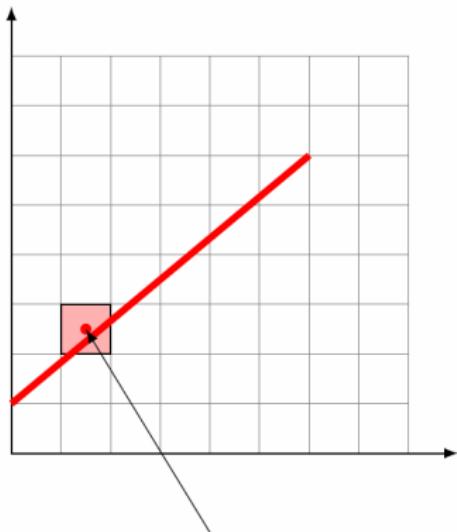
Exemplu

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

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

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

Varianta 3 - algoritmul lui Bresenham. Idei fundamentale



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

Doi "candidați" pt pixelul următor:
 $j: (x_k + 1, y_k)$ (Δ ipoteza
 $s: (x_k + 1, y_k + 1)$ privind
 punctul dreptei)

Ideea: se calculează distanța de la punctul de pe dreaptă coresp. abscisei $x_k + 1$ la centrele pixelilor candidați:

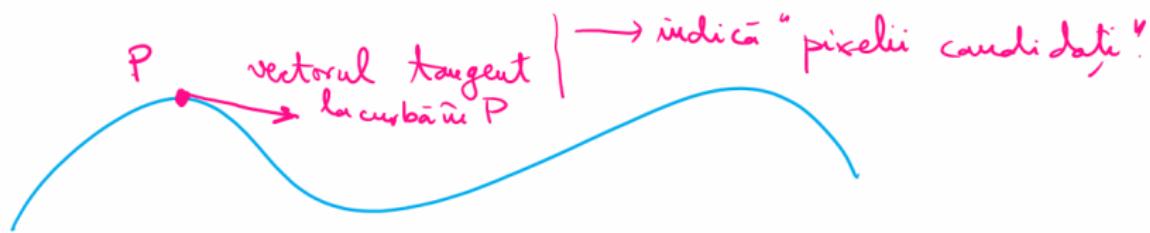
$$d_s = y_k + 1 - f(x_k + 1)$$

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

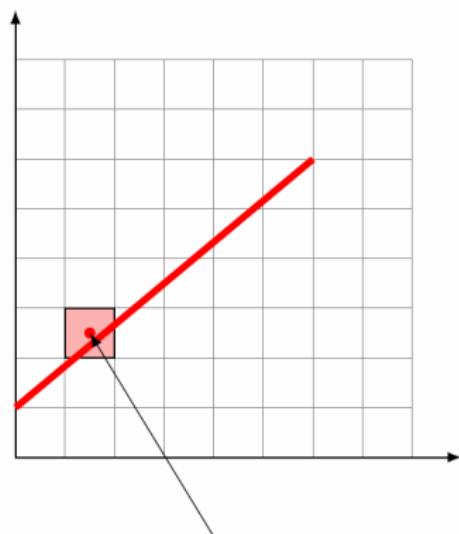
(sunt diferențe de ordonate, pt. au aceeași abscisă)

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.



Algoritmul lui Bresenham. Factorul de decizie



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

$$d_s = \underbrace{y_k + 1}_{\text{ordonata pixelului de sus}} - \underbrace{f(x_k + 1)}_{\text{ordonata pct. de pe dreapta}}$$

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

$$d_s = y_k + 1 - m x_k - m - m$$

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

$d_s \geq d_j$

?

Algoritmul lui Bresenham. Semnul factorului de decizie

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

$$d_j - d_s = \frac{2m(x_k + 1) - 2y_k + 2m - 1}{\Delta y} =$$

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

$\Delta x > 0$

$$\boxed{\delta = 2\Delta y + 2m \Delta x - \Delta x} \quad (\text{notatie})$$

Deci : semnul lui $d_j - d_s$ este semnul

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

Algoritmul lui Bresenham. Parametrul de decizie

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

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

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

Algoritmul lui Bresenham. Parametrul de decizie - rescriere

Evaluăm $p_{k+1} - p_k =$

$$\begin{aligned}
 &= 2\Delta y \cancel{x_{k+1}} - 2\Delta x \cancel{y_{k+1}} + \cancel{\delta} - \\
 &- 2\Delta y \cancel{x_k} + 2\Delta x \cancel{y_k} - \cancel{\delta} = \\
 &\quad \text{"} \quad (x_{k+1} - x_k = 1) \qquad \qquad y_{k+1} - y_k = \begin{cases} 0, & p_k < 0 \\ 1, & p_k \geq 0 \end{cases}
 \end{aligned}$$

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

Algoritmul lui Bresenham. Parametrul de decizie - valoarea lui p_0

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

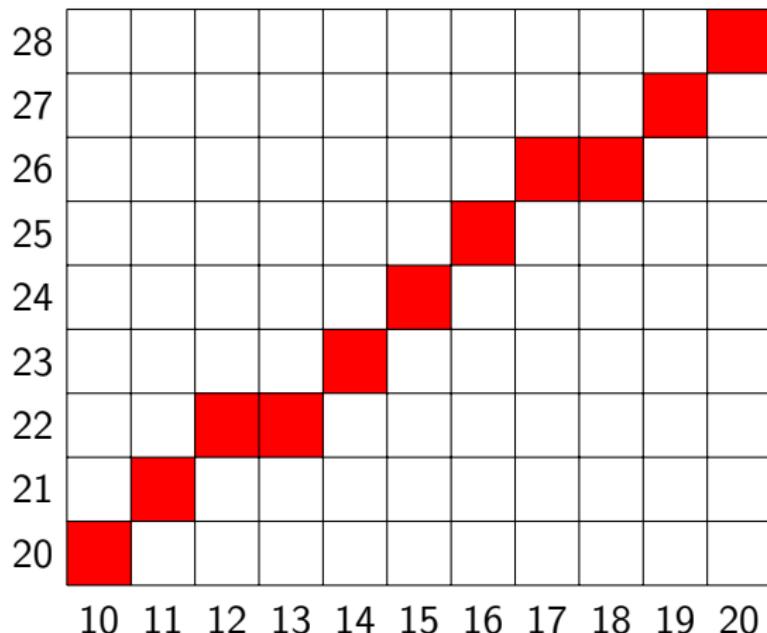


Figura: Algoritmul DDA / algoritmul lui Bresenham. Pixelii selectați pentru a uni punctele $(10, 20)$ și $(20, 28)$ sunt colorați cu roșu.

Primitive grafice. Fața și spatele unui poligon convex

Mihai-Sorin Stupariu

Sem. I, 2024 - 2025

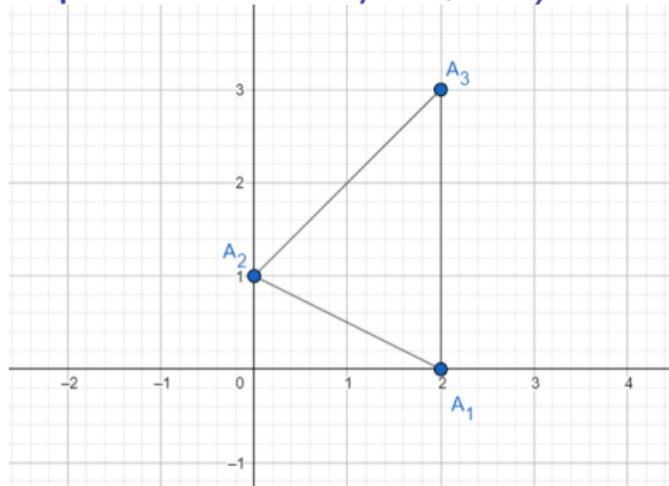
Problematizare

Condiții pentru poligoane

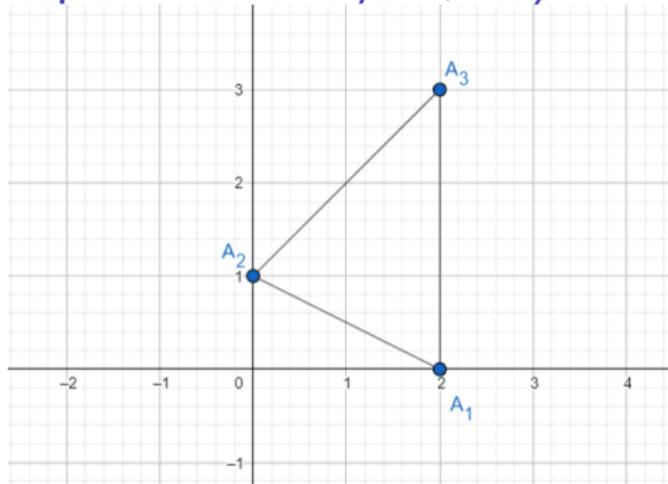
Vector normal. Față și spatele unui poligon convex

Linii poligonale închise cu autointersecții

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

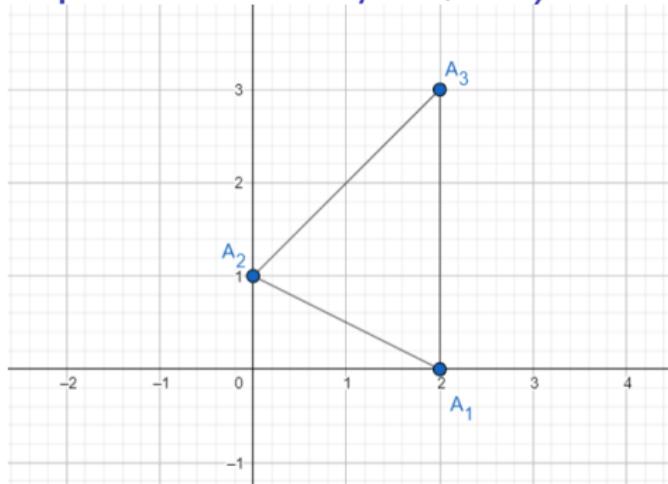


De reținut (și de aplicat la cerința 2, L2)!



Considerăm figura de mai sus. Dacă în codul sursă vârfurile sunt indicate în ordinea A_1, A_3, A_2 , atunci triunghiul din figură 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). Modul de parcursare poate fi schimbat.

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



Considerăm figura de mai sus. Dacă în codul sursă vârfurile sunt indicate în ordinea A_1, A_3, A_2 , atunci triunghiul din figură 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). Modul de parcursare poate fi schimbat. Exemplu: codul 02_02_fata_spate_poligon.cpp

Motivație

- ▶ Codurile sursă 02_02_fata_spate_poligon.cpp, 02_03_poligoane3D.cpp, 02_04_poligoane3D.cpp.

Motivație

- ▶ Codurile sursă 02_02_fata_spate_poligon.cpp, 02_03_poligoane3D.cpp, 02_04_poligoane3D.cpp.
- ▶ Ce proprietăți geometrice sunt / NU sunt implementate în OpenGL?

Motivație

- ▶ Codurile sursă 02_02_fata_spate_poligon.cpp, 02_03_poligoane3D.cpp, 02_04_poligoane3D.cpp.
- ▶ Ce proprietăți geometrice sunt / NU sunt implementate în OpenGL?
 - ▶ **NU:** reguli pentru aplicarea opțiunii GL_POLYGON - se presupune că **vârfurile determină un poligon convex** (exemplificare: luați $A_1 = (0, 0)$, $A_2 = (0.4, 0)$, $A_3 = (0.4, 0.4)$, $A_4 = (0.28, 0.12)$ și desenați, folosind modul GL_POLYGON, poligonul $A_1A_2A_3A_4$, apoi poligonul $A_4A_1A_2A_3$)

Motivație

- ▶ Codurile sursă 02_02_fata_spate_poligon.cpp, 02_03_poligoane3D.cpp, 02_04_poligoane3D.cpp.
- ▶ Ce proprietăți geometrice sunt / NU sunt implementate în OpenGL?
 - ▶ **NU:** reguli pentru aplicarea opțiunii GL_POLYGON - se presupune că **vârfurile determină un poligon convex** (exemplificare: luați $A_1 = (0, 0)$, $A_2 = (0.4, 0)$, $A_3 = (0.4, 0.4)$, $A_4 = (0.28, 0.12)$ și desenați, folosind modul GL_POLYGON, poligonul $A_1A_2A_3A_4$, apoi poligonul $A_4A_1A_2A_3$)
 - ▶ **DA:** fața 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:

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.

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.

În continuare primele două subpuncte sunt descrise succint, pentru cel de-al treilea sunt prezentate mai multe detalii (fapt esențial: **pentru un poligon convex vom putea defini fața și spatele poligonului**).

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 $\overrightarrow{P_1P_2} \times \overrightarrow{P_2P_3}$, $\overrightarrow{P_2P_3} \times \overrightarrow{P_3P_4}$, ..., $\overrightarrow{P_{N-1}P_N} \times \overrightarrow{P_NP_1}$, $\overrightarrow{P_NP_1} \times \overrightarrow{P_1P_2}$. 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.

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

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

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

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

vectorii sunt proporționali,
deci coliniari
 \Leftrightarrow punctele sunt coplanare

$$\overrightarrow{P_1P_2} = P_2 - P_1 = (-3, 3, 9) - (7, 1, 1) = (-10, 2, 8)$$

$$\overrightarrow{P_2P_3} = P_3 - P_2 = (1, -1, 9) - (-3, 3, 9) = (4, -4, 0)$$

$$\overrightarrow{P_1P_2} \times \overrightarrow{P_2P_3} = \begin{vmatrix} -10 & 4 & e_1 \\ 2 & -4 & e_2 \\ 8 & 0 & e_3 \end{vmatrix} \stackrel{\text{dezv.}}{\overbrace{\text{ultima}}} \stackrel{\text{columna}}{\overbrace{|}} |e_1 - \begin{vmatrix} 2 & -4 \\ 8 & 0 \end{vmatrix} e_2 +$$

$$+ \begin{vmatrix} -10 & 4 \\ 2 & -4 \end{vmatrix} e_3 = 32e_1 + 32e_2 + 32e_3 = (32, 32, 32)$$

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.

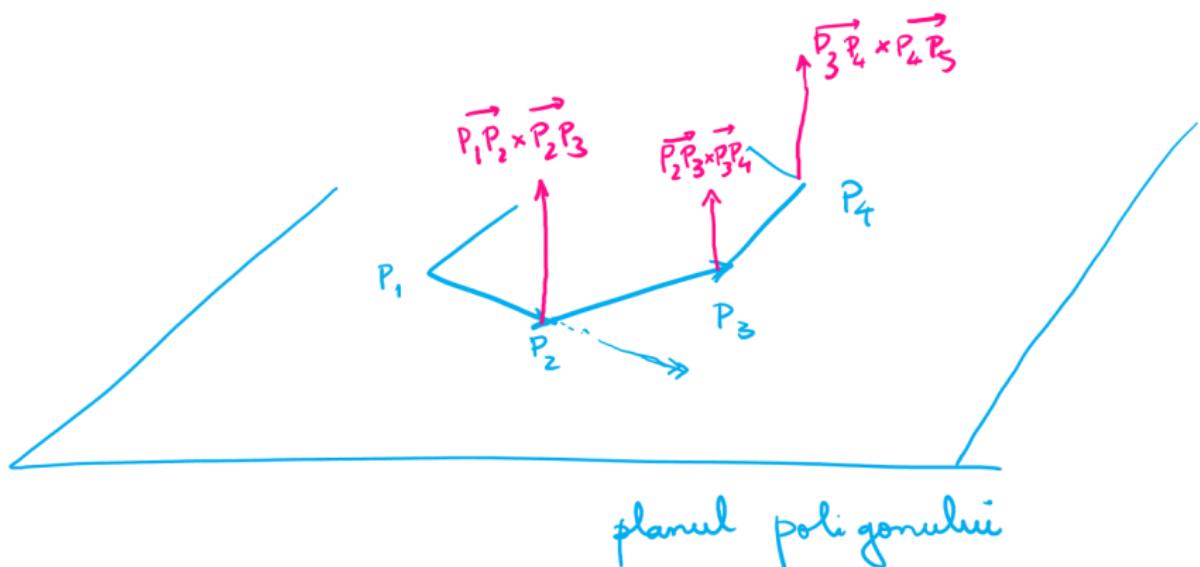
Varianta 1 Segmentele $[AB]$ și $[CD]$ se intersectează $\Leftrightarrow A$ și B sunt de o parte și de alta a dreptei CD și C și D sunt de o parte și de alta a dreptei AB . Două puncte M și N sunt de o parte și de alta a dreptei d de ecuație $f(x, y) = \alpha x + \beta y + \gamma = 0 \Leftrightarrow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

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!)

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) .

Convexitatea poligonului - observație

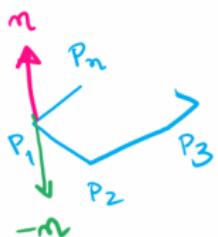
Obs. Fie (P_1, P_2, \dots, P_n) un poligon convex.

(i) Parcurgerea $P_1 P_2 \dots P_n \rightarrow$ vector normal

m

(ii) Parcurgerea $P_n P_{n-1} \dots P_1 \rightarrow -m$

$-m$



Modalitate de calcul (I)

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

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

Modalitate de calcul (II)

3 Are loc relația

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

4 În final:

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

5 În particular, există o legătură între vectorul $\mathbf{n} = \frac{1}{\sqrt{A^2 + B^2 + C^2}}(A, B, C)$ și ecuația $Ax + By + Cz + D = 0$ asociată planului poligonului considerat (observați ce se întâmplă dacă se schimbă ordinea parcurgerii vârfurilor!).

Conceptul de față / spate ale unui poligon convex

Considerăm un poligon (P_1, P_2, \dots, P_n) pentru care am calculat ecuația planului $Ax + By + Cz + D = 0$ ca pe slide-ul 13 (ordinea parcurgerii vârfurilor contează!).

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 ale unui poligon convex

Considerăm un poligon (P_1, P_2, \dots, P_n) pentru care am calculat ecuația planului $Ax + By + Cz + D = 0$ ca pe slide-ul 13 (ordinea parcurgerii vârfurilor contează!).

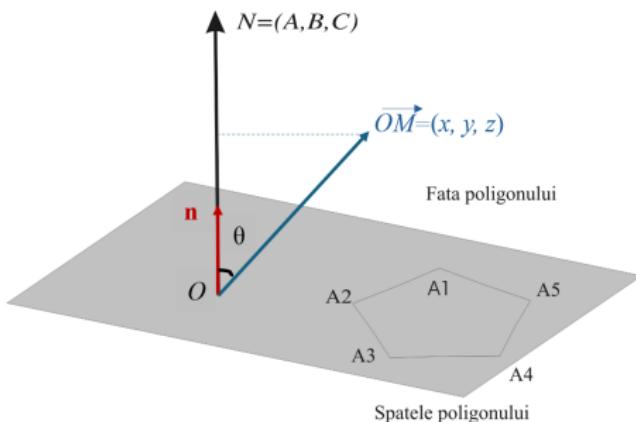
Definiție. Pentru un punct $M = (x, y, z) \in \mathbb{R}^3$ notăm

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

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

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

Figura - “normala indică fața poligonului”



În figură, punctul M este în fața poligonului $A_1A_2A_3A_4A_5$.

Definiția este echivalentă cu faptul că unghiul θ dintre vectorul \overrightarrow{OM} și vectorul \mathbf{n} este mai mic de 90° , adică proiecția vectorului \overrightarrow{OM} / a punctului M pe dreapta ce trece prin O și este direcționată de \mathbf{n} are același sens cu \mathbf{n} , altfel spus, că vectorii \mathbf{n} și \overrightarrow{OM} sunt de aceeași parte a planului.

Justificare teoretică - “normala indică fața poligonului”

- ▶ Presupunem că $D = 0$, adică planul trece prin originea $O = (0, 0, 0)$ - cf. Figura de pe slide-ul 16.

Justificare teoretică - “normala indică fața poligonului”

- ▶ Presupunem că $D = 0$, adică planul trece prin originea $O = (0, 0, 0)$ - cf. Figura de pe slide-ul 16.
- ▶ Conform definiției de pe slide-ul 15, un punct $M = (x, y, z)$ este în fața poligonului \Leftrightarrow

$$A \cdot x + B \cdot y + C \cdot z > 0. \quad (3)$$

- ▶ Expresia din membrul stâng al relației (3) este, de fapt, produsul scalar dintre vectorii (A, B, C) și (x, y, z) . Mai departe, întrucât vectorul (A, B, C) este coliniar și de același sens cu \mathbf{n} , iar $(x, y, z) = \overrightarrow{OM}$, inegalitatea (3) devine

$$\langle \mathbf{n}, \overrightarrow{OM} \rangle > 0 \Leftrightarrow \cos(\angle(\mathbf{n}, \overrightarrow{OM})) > 0 \Leftrightarrow \angle(\mathbf{n}, \overrightarrow{OM}) < 90^\circ. \quad (4)$$

Justificare teoretică - “normala indică fața poligonului”

- ▶ Presupunem că $D = 0$, adică planul trece prin originea $O = (0, 0, 0)$ - cf. Figura de pe slide-ul 16.
- ▶ Conform definiției de pe slide-ul 15, un punct $M = (x, y, z)$ este în fața poligonului \Leftrightarrow

$$A \cdot x + B \cdot y + C \cdot z > 0. \quad (3)$$

- ▶ Expresia din membrul stâng al relației (3) este, de fapt, produsul scalar dintre vectorii (A, B, C) și (x, y, z) . Mai departe, întrucât vectorul (A, B, C) este coliniar și de același sens cu \mathbf{n} , iar $(x, y, z) = \overrightarrow{OM}$, inegalitatea (3) devine

$$\langle \mathbf{n}, \overrightarrow{OM} \rangle > 0 \Leftrightarrow \cos(\angle(\mathbf{n}, \overrightarrow{OM})) > 0 \Leftrightarrow \angle(\mathbf{n}, \overrightarrow{OM}) < 90^\circ. \quad (4)$$

- ▶ Condiția $\angle(\mathbf{n}, \overrightarrow{OM}) < 90^\circ$ este echivalentă cu faptul că proiecția vectorului \overrightarrow{OM} / a punctului M pe dreapta ce trece prin O și este direcționată de \mathbf{n} are același sens cu \mathbf{n} , altfel spus, că vectorii \mathbf{n} și \overrightarrow{OM} sunt de aceeași parte a planului.

Justificare teoretică - “normala indică fața poligonului”

- ▶ Presupunem că $D = 0$, adică planul trece prin originea $O = (0, 0, 0)$ - cf. Figura de pe slide-ul 16.
- ▶ Conform definiției de pe slide-ul 15, un punct $M = (x, y, z)$ este în fața poligonului \Leftrightarrow

$$A \cdot x + B \cdot y + C \cdot z > 0. \quad (3)$$

- ▶ Expresia din membrul stâng al relației (3) este, de fapt, produsul scalar dintre vectorii (A, B, C) și (x, y, z) . Mai departe, întrucât vectorul (A, B, C) este coliniar și de același sens cu \mathbf{n} , iar $(x, y, z) = \overrightarrow{OM}$, inegalitatea (3) devine

$$\langle \mathbf{n}, \overrightarrow{OM} \rangle > 0 \Leftrightarrow \cos(\angle(\mathbf{n}, \overrightarrow{OM})) > 0 \Leftrightarrow \angle(\mathbf{n}, \overrightarrow{OM}) < 90^\circ. \quad (4)$$

- ▶ Condiția $\angle(\mathbf{n}, \overrightarrow{OM}) < 90^\circ$ este echivalentă cu faptul că proiecția vectorului \overrightarrow{OM} / a punctului M pe dreapta ce trece prin O și este direcționată de \mathbf{n} are același sens cu \mathbf{n} , altfel spus, că vectorii \mathbf{n} și \overrightarrow{OM} sunt de aceeași parte a planului.
- ▶ În concluzie, normala indică fața unui poligon convex.

Justificare teoretică - sinteză / reformulări

- ▶ Presupunem că $D = 0$, deci planul trece prin origine, iar ecuația sa este $\pi(x, y, z) = Ax + By + Cz = 0$.
- ▶ Considerând vectorul $\mathbf{n} = (A, B, C)$ care direcționează normala la plan, avem $\pi(A, B, C) > 0$, deci vectorul \mathbf{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. $\langle (x, y, z), \mathbf{n} \rangle > 0$, ceea ce înseamnă că proiecția vectorului (x, y, z) pe N este la fel orientată ca și \mathbf{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 fața poligonului (vezi figura de pe slide-ul 17).

De reținut

- ▶ Pentru un poligon convex putem defini fața / spatele poligonului. Ordinea parcurgerii vârfurilor este esențială!

De reținut

- ▶ Pentru un poligon convex putem defini față / spatele poligonului. Ordinea parcurgerii vârfurilor este esențială!
- ▶ Putem stabili dacă un punct este în față / spatele unui poligon cu un criteriu **algebric**, folosind ecuația planului asociat (vezi slide 15 - aceasta este definiția formală).

De reținut

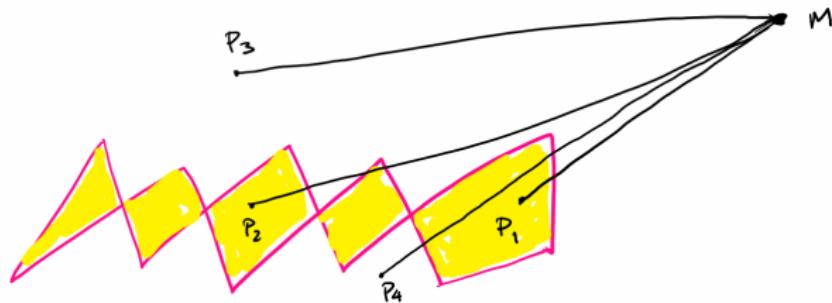
- ▶ Pentru un poligon convex putem defini față / spatele poligonului. Ordinea parcurgerii vârfurilor este esențială!
- ▶ Putem stabili dacă un punct este în față / spatele unui poligon cu un criteriu **algebric**, folosind ecuația planului asociat (vezi slide 15 - aceasta este definiția formală).
- ▶ Conceptul de față / spate pentru un poligon convex este legat de vectorul normal (normală), care indică fața poligonului (vezi slide 17).

De reținut

- ▶ Pentru un poligon convex putem defini față / spatele poligonului. Ordinea parcurgerii vârfurilor este esențială!
- ▶ Putem stabili dacă un punct este în față / spatele unui poligon cu un criteriu **algebric**, folosind ecuația planului asociat (vezi slide 15 - aceasta este definiția formală).
- ▶ Conceptul de față / spate pentru un poligon convex este legat de vectorul normal (normală), care indică fața poligonului (vezi slide 17).
- ▶ **Intuitiv / geometric:** 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 (vezi slide 18).

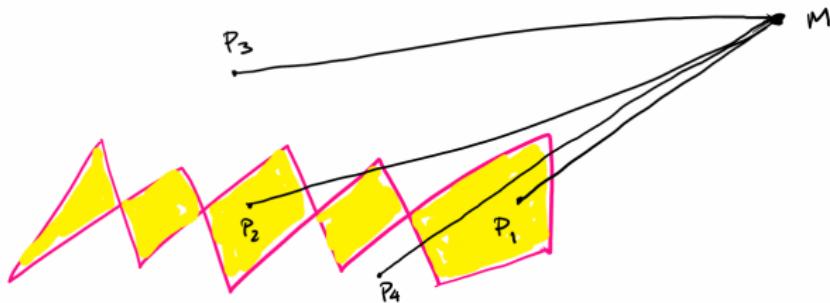
Linii poligonale închise cu autointersecții: interior/exterior

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



Linii poligonale închise cu autointersecții: interior/exterior

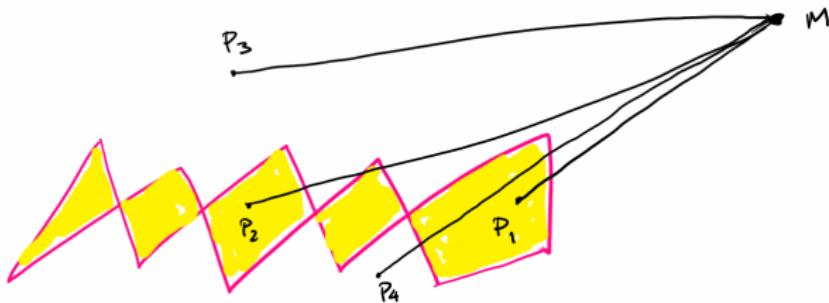
Regula par-impar (*odd-even rule*)



- ▶ Se alege un punct M “departe” de linia poligonală (de exemplu în afara dreptunghiului determinat de $x_{\min}, x_{\max}, y_{\min}, y_{\max}$)

Linii poligonale închise cu autointersecții: interior/exterior

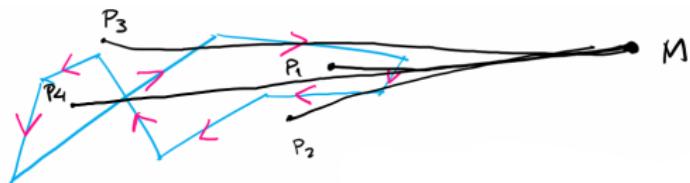
Regula par-impar (*odd-even rule*)



- ▶ Se alege un punct M “departe” de linia poligonală (de exemplu în afara dreptunghiului determinat de $x_{\min}, x_{\max}, y_{\min}, y_{\max}$)
- ▶ Pentru un punct P : paritatea numărului de intersecții dintre linia poligonală și **segmentul** $[MP]$ (cu convenții...) este factor de decizie:
 - punctul P este **exterior** dacă numărul de intersecții este **par**
 - punctul P este **interior** dacă numărul de intersecții este **impar**

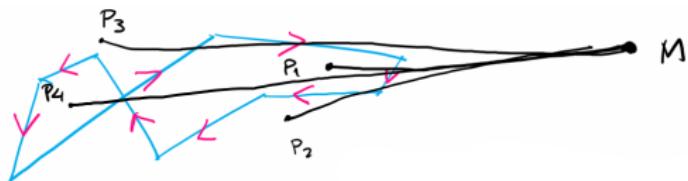
Linii poligonale închise cu autointersecții: interior/exterior

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



Linii poligonale închise cu autointersecții: interior/exterior

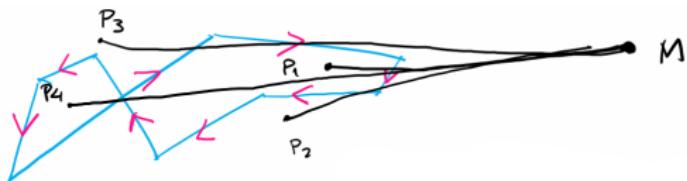
Regula indexului nenul (*non-zero winding number rule*)



- ▶ Pe lângă alegerea lui M , este fixat un sens de parcurgere, se stabilesc convenții de semn pentru modul în care, în punctele de intersecție cu linia poligonală, segmentul orientat $[MP]$ (deplasare de la M la P) se raportează la acest sens (de exemplu: stânga "-", dreapta "+"). Numărul de intersecții cu semnul "-" este notat n_- , cel cu semnul "+" este notat n_+ .

Linii poligonale închise cu autointersecții: interior/exterior

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



- ▶ Pe lângă alegerea lui M , este fixat un sens de parcurgere, se stabilesc convenții de semn pentru modul în care, în punctele de intersecție cu linia poligonală, segmentul orientat $[MP]$ (deplasare de la M la P) se raportează la acest sens (de exemplu: stânga "-", dreapta "+"). Numărul de intersecții cu semnul "-" este notat n_- , cel cu semnul "+" este notat n_+ .
- ▶ **Indexul lui P** este $i_P = n_+ - n_-$
 - punctul P este **exterior** dacă indexul este **0**
 - punctul P este **interior** dacă indexul este $\neq 0$

Linii poligonale închise cu autointersecții: interior/exterior

Legătura dintre cele două reguli

- Are loc relația $(n_+ + n_-) = \text{nr. total de intersecții}$, iar paritatea acestui număr ne dă regula par/impar.

Linii poligonale închise cu autointersecții: interior/exterior

Legătura dintre cele două reguli

- ▶ Are loc relația $(n_+ + n_-) =$ nr. total de intersecții, iar paritatea acestui număr ne dă regula par/impar.
- ▶ Dacă un punct P este exterior pentru regula indexului:

$$n_+ = n_- \Rightarrow (n_+ + n_-) \text{ este par,}$$

deci P este exterior și pentru regula par-impar.

Linii poligonale închise cu autointersecții: interior/exterior

Legătura dintre cele două reguli

- ▶ Are loc relația $(n_+ + n_-) =$ nr. total de intersecții, iar paritatea acestui număr ne dă regula par/impar.
- ▶ Dacă un punct P este exterior pentru regula indexului:

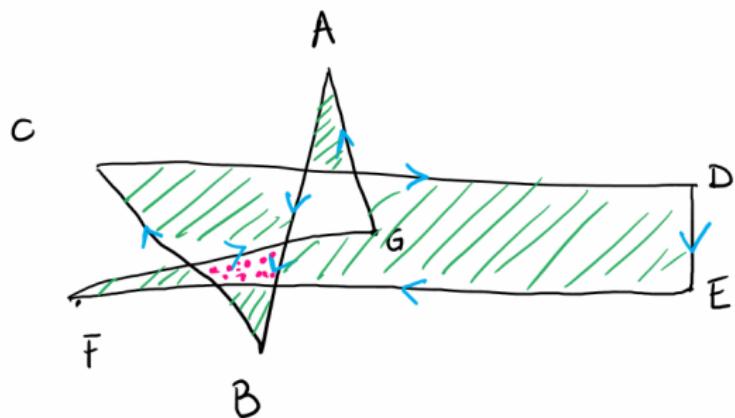
$$n_+ = n_- \Rightarrow (n_+ + n_-) \text{ este par,}$$

deci P este exterior și pentru regula par-impar.

- ▶ Reciproc nu este adevărat, adică există puncte care sunt exterioare pentru regula par-impar, dar nu sunt exterioare pentru regula indexului (v. exemplu).

Linii poligonale închise cu autointersecții: interior/exterior

Exemplu



exterior și
ptr.p/i și
ptr.index

exterior
ptr.p/i,
interior ptr.
index

interior
n ptv.p/i
n ptv.
index

Transformări

Mihai-Sorin Stupariu

Sem. I, 2024 - 2025

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

```
// Parametri pentru glOrtho2D() - decupare;  
GLfloat xMin = 0, yMin = 0, xMax = 800.0, yMax = 600.0;  
  
// Setarea parametrilor necesari pentru fereastra de viz  
void Initialize(void)  
{  
    glClearColor(1.0, 1.0, 1.0, 0.0);  
    gluOrtho2D(xMin, xMax, yMin, yMax);
```

Coordonata x intre 0 si 800, coordonata y intre 0 si 600 - "dreptunghi decupat"

```
// Se translateaza dreptunghiul  
glPushMatrix();  
glTranslated(i, 200.0, 0.0);  
// Se roteste dreptunghiul - se  
glPushMatrix();  
glRotated(j, 0.0, 0.0, 1.0);  
// Se deseneaza dreptunghiul;  
glColor3f(1.0, 0.0, 0.0);  
glRecti(-5, 30, 5, 40);
```

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

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 de transformare pentru dreptunghiul ALBASTRU;
myMatrix = resizeMatrix * matrTransl * matrScale1;
codCol = 1;
// Transmiterea variabilelor uniforme pentru MATRICEA DE TRANSFORMARE si COLOR
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0]);
glUniform1i(codCollocation, codCol);
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 \mathbf{T}_t de vector $\mathbf{t} = (t_1, t_2, t_3)$

Functii pentru transformări în OpenGL

- `glTranslate*(t); // glm::translate` Translația \mathbf{T}_t de vector $\mathbf{t} = (t_1, t_2, t_3)$

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

Funcții pentru transformări în OpenGL

- `glTranslate*(t); // glm::translate` Translația \mathbf{T}_t de vector $\mathbf{t} = (t_1, t_2, t_3)$

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

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

Funcții pentru transformări în OpenGL

- `glTranslate*(t); // glm::translate` Translația \mathbf{T}_t de vector $\mathbf{t} = (t_1, t_2, t_3)$

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

- `glScale*(s); // glm::scale` Scalarea σ_s de factor $\mathbf{s} = (s_1, s_2, s_3)$ (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 \mathbf{T}_t de vector $\mathbf{t} = (t_1, t_2, t_3)$

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

- `glScale*(s); // glm::scale` Scalarea σ_s de factor $\mathbf{s} = (s_1, s_2, s_3)$ (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 $\mathbb{R}_{\mathbf{u}, \theta}$ de unghi θ și axă dată de versorul \mathbf{u} // Rotația 2D $\mathbb{R}_{3, \theta}$ de axă Ox_3 (adică $\mathbf{u} = (0, 0, 1)$ și unghi θ (centrul rotației fiind în origine - punct fix al transformării) este dată de

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

Functii pentru transformari in OpenGL

- `glTranslate*(t); // glm::translate` Translația \mathbf{T}_t de vector $\mathbf{t} = (t_1, t_2, t_3)$

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

- `glScale*(s); // glm::scale` Scalarea σ_s de factor $\mathbf{s} = (s_1, s_2, s_3)$ (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 $\mathbb{R}_{\mathbf{u}, \theta}$ de unghi θ și axă dată de versorul \mathbf{u} // Rotația 2D $\mathbb{R}_{3, \theta}$ de axă Ox_3 (adică $\mathbf{u} = (0, 0, 1)$ și unghi θ (centrul rotației fiind în origine - punct fix al transformării) este dată de

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

- 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 este reprezentat (intern) ca având 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

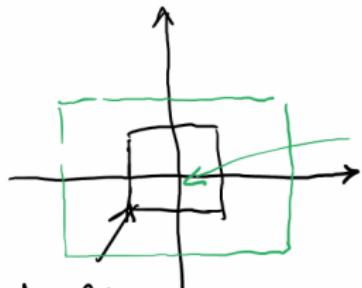
Componerea transformărilor; despre `glm::ortho` - Codul sursă 03_03_resize.cpp

- 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”
 $[-1, 1] \times [-1, 1]$.

Componerea transformărilor; despre `glm::ortho` - 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 componerea dintre o translație și o scalare. **Atenție la ordine!**

Componerea transformărilor; despre `glm::ortho` - Codul sursă 03_03_resize.cpp



"dreptunghi standard"
 $[-1, 1] \times [-1, 1]$



$(x_c, y_c) \rightarrow$ centru

$x_c = \frac{x_{\min} + x_{\max}}{2}$ "dreptunghiul în care se află primitivele)

$$y_c = \frac{y_{\min} + y_{\max}}{2}$$

- (1) "recenterăm", $\overline{T}(-x_c, -y_c) \Rightarrow M_T$ (matrice)

\Rightarrow dreptunghi cu centru în origine)

- (2) scalău: scalare σ cu factorii $\frac{2}{Δx}$, $\frac{2}{Δy} \Rightarrow$ matrice M_B
 \Rightarrow matricea $M_B \times M_T$

Obs. importantă. Rotările și scalările au originea ca punct fix.
Dacă dorim să aplicăm o rotație / scalare cu centru oarecare, avem de realizat o compunere:

- fie C centrul rotației

• aplicăm translatărea de vector $\vec{OC} = \vec{T}_{OC}(M_1)$

• aplicăm rotația / scalarea (M_2) matrice

• aplicăm translatația de vector $\vec{OC} = \vec{T}_{OC}(M_3)$

\Rightarrow matricea folosită este $M_3 * M_2 * M_1$

Transformări

Mihai-Sorin Stupariu

Sem. I, 2024 - 2025

Exemple de transformări

Coordinate omogene

Coordinate omogene - breviar teoretic

Sinteză

Functii pentru transformări în OpenGL

- `glm::translate` Translația \mathbf{T}_t de vector $\mathbf{t} = (t_1, t_2, t_3)$

Functii pentru transformări în OpenGL

- `glm::translate` Translația \mathbf{T}_t de vector $\mathbf{t} = (t_1, t_2, t_3)$

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

Funcții pentru transformări în OpenGL

- **glm::translate** Translația \mathbf{T}_t de vector $\mathbf{t} = (t_1, t_2, t_3)$

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

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

Funcții pentru transformări în OpenGL

- **glm::translate** Translația \mathbf{T}_t de vector $\mathbf{t} = (t_1, t_2, t_3)$

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

- **glm::scale** Scalarea σ_s de factor $\mathbf{s} = (s_1, s_2, s_3)$ (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 \mathbf{T}_t de vector $\mathbf{t} = (t_1, t_2, t_3)$

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

- **glm::scale** Scalarea σ_s de factor $\mathbf{s} = (s_1, s_2, s_3)$ (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 $\mathbf{R}_{\mathbf{u}, \theta}$ de unghi θ și axă dată de vesorul \mathbf{u} //Rotația 2D $\mathbf{R}_{3,\theta}$ de axă Ox_3 (adică $\mathbf{u} = (0, 0, 1)$ și unghi θ (centrul rotației fiind în origine - punct fix al transformării) este dată de

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

Funcții pentru transformări în OpenGL

- **glm::translate** Translația \mathbf{T}_t de vector $\mathbf{t} = (t_1, t_2, t_3)$

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

- **glm::scale** Scalarea σ_s de factor $\mathbf{s} = (s_1, s_2, s_3)$ (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 $\mathbf{R}_{\mathbf{u}, \theta}$ de unghi θ și axă dată de vesorul \mathbf{u} //Rotația 2D $\mathbf{R}_{3,\theta}$ de axă Ox_3 (adică $\mathbf{u} = (0, 0, 1)$ și unghi θ (centrul rotației fiind în origine - punct fix al transformării) este dată de

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

- 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 $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, $f(x_1, x_2) = (2x_1 + 4x_2, 3x_1 - x_2)$.

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

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

$$f(2,5) = (24,11); \quad \begin{pmatrix} 2 \\ 5 \end{pmatrix} \mapsto \begin{pmatrix} 24 \\ 11 \end{pmatrix}$$

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

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

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 $f(e_1), f(e_2)$).

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

(
" $f(e_1)$ " $f(e_2)$ "
)

↓ coloanele matricei M_f
sunt $f(e_1)$ și $f(e_2)$

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

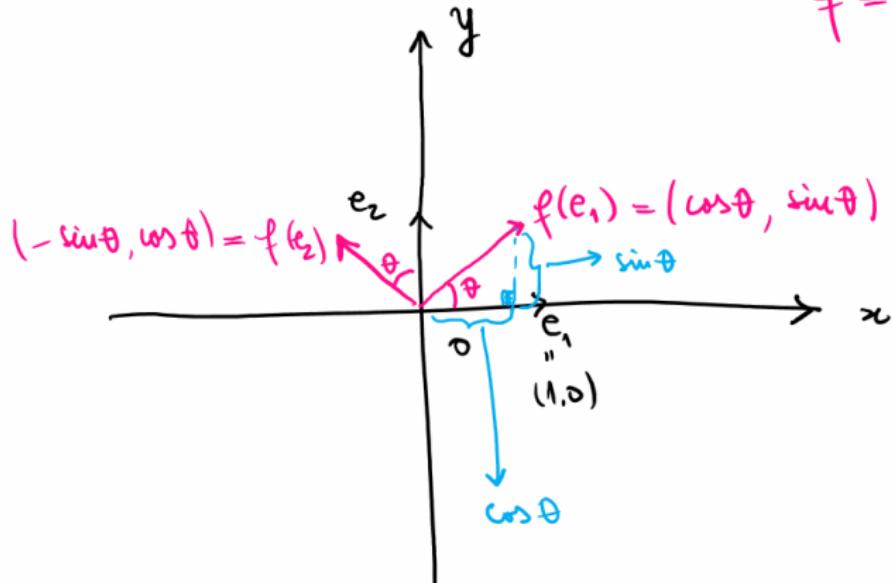
Rotația 2D $\mathbf{R}_{\mathbf{u},\theta}$ (cu axa de rotație $\mathbf{u} = (0, 0, 1)$) de unghi θ , cu centrul în origine are matricea 2×2 asociată dată de

$$M_{\mathbf{R}_{\mathbf{u},\theta}} = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}.$$

Reprezentată ca matrice 3×3 , aceasta devine

$$M_{\mathbf{R}_{\mathbf{u},\theta}} = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Figura - rotații 2D



$$f = R_{0,s} \text{ in plan}$$

1

$$M_f =$$

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

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) i 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

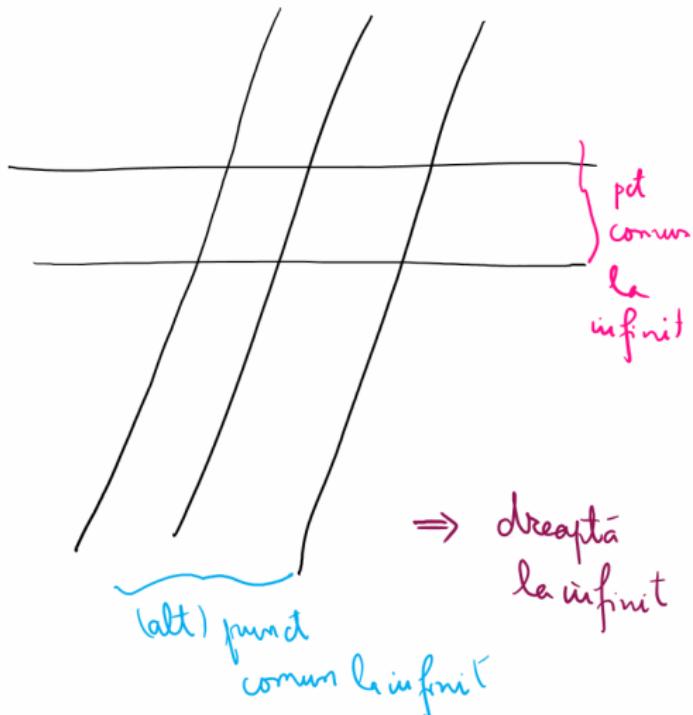
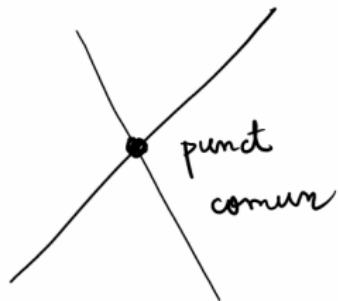
Pe lângă funcțiile din biblioteca glm, transformările de modelare pot fi apelate în codul sursă indicând explicit matricea 4×4 asociată. Prin convenție, elementele sunt introduse pe coloane, de sus în jos, de la stânga la dreapta

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

În continuare...

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

Construcție I - geometrică



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**.
- ▶ Putem 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.

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:

Fie (x_1, x_2) și (y_1, y_2) cu $\iota(x_1, x_2) = \iota(y_1, y_2)$, adică $[x_1 : x_2 : 1] = [y_1 : y_2 : 1]$. Aceasta înseamnă, conform definiției, că există $\lambda \neq 0$ astfel încât $\lambda(x_1, x_2, 1) = (y_1, y_2, 1)$. Scriind acastă relație pentru fiecare coordonată separat avem

$$\begin{cases} \lambda \cdot x_1 = y_1 \\ \lambda \cdot x_2 = y_2 \\ \lambda \cdot 1 = 1 \end{cases}$$

Se deduce că $\lambda = 1$, deci $(x_1, x_2) = (y_1, y_2)$, adică 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.

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 direcatori 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:

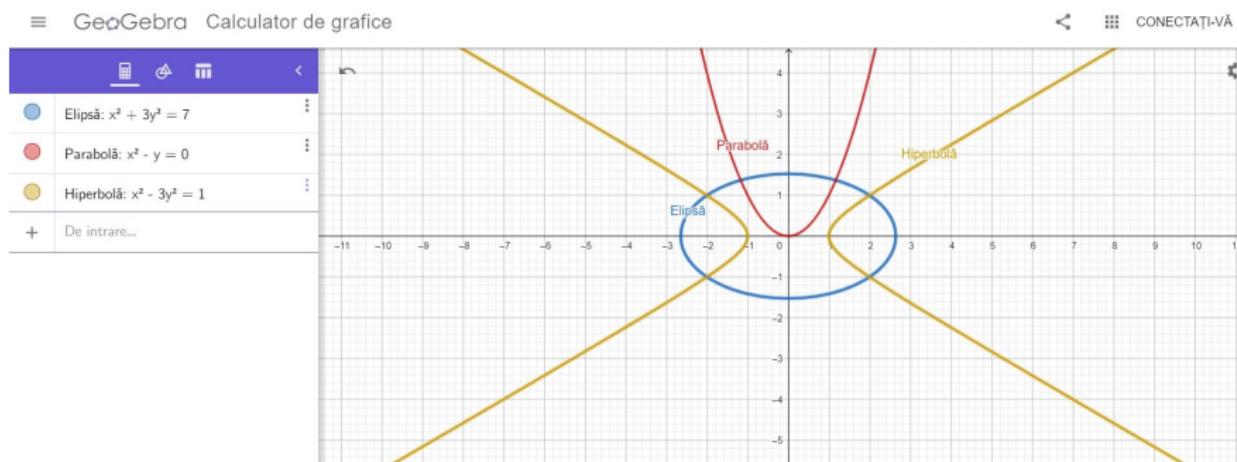
- ▶ **"Puncte reale"**: elemente de forma $P = [X_1 : X_2 : X_0]$ cu $X_0 \neq 0$; P îi corespunde punctului $\left(\frac{X_1}{X_0}, \frac{X_2}{X_0}\right)$ din spațiul geometric \mathbb{R}^2 (deoarece $[X_1 : X_2 : X_0] = [\frac{X_1}{X_0} : \frac{X_2}{X_0} : 1]$). În codul sursă 04_01_coord_omogene.cpp punctul $[-100 : -100 : 0 : 0.5]$ coincide cu $[-200 : -200 : 0 : 1]$, deci corespunde punctului $(-200, -200)$.

Sinteză

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

- ▶ **"Puncte 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 $[X_1 : X_2 : X_0] = [\frac{X_1}{X_0} : \frac{X_2}{X_0} : 1]$). În codul sursă 04_01_coord_omogene.cpp punctul $[-100 : -100 : 0 : 0.5]$ coincide cu $[-200 : -200 : 0 : 1]$, deci corespunde punctului $(-200, -200)$.
- ▶ **"Puncte de la infinit"**: elemente de forma $Q = [X_1 : X_2 : 0]$. În codul sursă 04_01_coord_omogene.cpp punctul $[-100 : -100 : 0 : 0]$ este punct de la infinit și corespunde direcției $(-100, -100)$, adică primei bisectoare. 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, hiperbolă?



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

► **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: $x_1 = \frac{x_1}{x_0}$, $x_2 = \frac{x_2}{x_0}$.

► **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: $X_1^2 + X_2^2 - X_0^2 = 0$.
- 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: $x_1 = \frac{x_1}{x_0}$, $x_2 = \frac{x_2}{x_0}$.
- 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 2:**

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

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

Se deduce că hiperbola $x_1^2 - x_2^2 - 1 = 0$ are două puncte la infinit, $[t : t : 0] = [1 : 1 : 0]$ și $[t : -t : 0] = [1 : -1 : 0]$.

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 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

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

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

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

- OpenGL utilizează coordonate omogene, fiind utilizate 4 coordonate.
Semnificația:

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

- OpenGL utilizează coordonate omogene, fiind utilizate 4 coordonate.
Semnificația:
 - Unui **vârf** de coordonate (x_1, x_2, x_3) , notate și (x, y, z) , i se asociază **coordonatele omogene** $[x_1 : x_2 : x_3 : 1]$ (sau $[x : y : z : 1]$),
$$(x_1, x_2, x_3) \mapsto [x_1 : x_2 : x_3 : 1] (= [\alpha x_1 : \alpha x_2 : \alpha x_3 : \alpha], \forall \alpha \neq 0).$$

De exemplu, pentru **punctul** $(1, 2, 5)$:

$$(1, 2, 5) \mapsto [1 : 2 : 5 : 1] = [3 : 6 : 15 : 3] = \dots, \text{etc.}$$

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

- OpenGL utilizează coordonate omogene, fiind utilizate 4 coordonate.
Semnificația:

- Unui **vârf** de coordonate (x_1, x_2, x_3) , notate și (x, y, z) , i se asociază **coordonatele omogene** $[x_1 : x_2 : x_3 : 1]$ (sau $[x : y : z : 1]$),

$$(x_1, x_2, x_3) \mapsto [x_1 : x_2 : x_3 : 1] (= [\alpha x_1 : \alpha x_2 : \alpha x_3 : \alpha], \forall \alpha \neq 0).$$

De exemplu, pentru **punctul** $(1, 2, 5)$:

$$(1, 2, 5) \mapsto [1 : 2 : 5 : 1] = [3 : 6 : 15 : 3] = \dots, \text{etc.}$$

- Unei **drepte / direcții** date de vectorul (v_1, v_2, v_3) , i se asociază **coordonatele omogene** $[v_1 : v_2 : v_3 : 0]$,

$$(v_1, v_2, v_3) \mapsto [v_1 : v_2 : v_3 : 0].$$

De exemplu, pentru **direcția** $(1, 4, 3)$:

$$(1, 4, -3) \mapsto [1 : 4 : -3 : 0] = [2 : 8 : -6 : 0] = \dots, \text{etc.}$$

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 $(b_i)_i$.

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

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

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

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

- De exemplu, matricea asociată translației T de vector $(2, -3, 5)$ este

$$\mathcal{M}_T = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & -3 \\ 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

De reținut: cum acționează matricele?

- **Principiu:** Dat un vârf / o direcție având coordonate omogene reprezentate de un vector coloană cu 4 componente

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

aplicarea transformării f generează un nou vector coloană, și anume

$$M_f \cdot \xi$$

(înmulțire la stânga cu M_f).

De reținut: cum acționează matricele?

- **Principiu:** Dat un vârf / o direcție având coordonate omogene reprezentate de un vector coloană cu 4 componente

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

aplicarea transformării f generează un nou vector coloană, și anume

$$M_f \cdot \xi$$

(înmulțire la stânga cu M_f).

- Din punctul de vedere al **implementării**, înmulțirea este realizată în shader-ul de vârfuri:

```
uniform mat4 Mf;

void main(void)
{
    gl_Position = Mf*in_Position;
```

Componerea transformărilor

- ▶ Dat un vârf / o direcție având coordonate omogene reprezentate de un vector coloană ξ , aplicarea unei transformări f cu matrice M_f generează un nou vector coloană, și anume $M_f \cdot \xi$.

Componerea transformărilor

- ▶ Dat un vârf / o direcție având coordonate omogene reprezentate de un vector coloană ξ , aplicarea unei transformări f cu matrice M_f generează un nou vector coloană, și anume $M_f \cdot \xi$.
- ▶ Fie f_1, f_2 transformări cu matrice M_{f_1}, M_{f_2} . Dacă **ordinea apelării în cod** este f_1, f_2 , atunci **ordinea aplicării este** f_2 , apoi f_1 (altfel spus, folosind compunerea funcțiilor, aplicăm $f_1 \circ f_2$). Matriceal avem:

$$\xi \mapsto M_{f_1} \cdot (M_{f_2} \cdot \xi) = (M_{f_1} \cdot M_{f_2}) \cdot \xi, \quad \text{deci}$$

$$M_{f_1 \circ f_2} = M_{f_1} \cdot M_{f_2}.$$

Așadar, compunerea transformărilor (aplicațiilor) \leftrightarrow înmulțirea matricelor (poate fi realizată în shader sau în programul principal).

Componerea transformărilor: exemplu

Dacă în codul sursă avem o secvență de tipul

```
m1 = glm::translate(...);  
m2 = glm::scale(...);  
m3 = glm::rotate(...);  
m = (m1 * m2) * m3;
```

ținând cont că în shader-ul de vârfuri avem

```
gl_Position = m * in_Position,
```

ordinea aplicării asupra vârfului (primitivei) este

Componerea transformărilor: exemplu

Dacă în codul sursă avem o secvență de tipul

```
m1 = glm::translate(...);  
m2 = glm::scale(...);  
m3 = glm::rotate(...);  
m = (m1 * m2) * m3;
```

ținând cont că în shader-ul de vârfuri avem

```
gl_Position = m * in_Position,
```

ordinea aplicării asupra vârfului (primitivei) este
rotația, scalarea, translația.

Sinteză

- ▶ Am discutat despre transformări (numite **transformări de modelare**), care sunt aplicate obiectelor din scenă. În OpenGL pot fi utilizate funcții din biblioteca glm pentru o serie de transformări (translație, rotație, scalare).

Sinteză

- ▶ Am discutat despre transformări (numite **transformări de modelare**), care sunt aplicate obiectelor din scenă. În OpenGL pot fi utilizate funcții din biblioteca glm pentru o serie de transformări (translație, rotație, scalare).
- ▶ Pentru modelarea / implementarea transformărilor sunt utilizate 4 coordonate. Unei transformări i se asociază o matrice 4×4 care acționează prin înmulțire (la stânga, asupra unui vector coloană 4×1 , ce reprezintă coordonate omogene).

Sinteză

- ▶ Am discutat despre transformări (numite **transformări de modelare**), care sunt aplicate obiectelor din scenă. În OpenGL pot fi utilizate funcții din biblioteca `glm` pentru o serie de transformări (translație, rotație, scalare).
- ▶ Pentru modelarea / implementarea transformărilor sunt utilizate 4 coordonate. Unei transformări i se asociază o matrice 4×4 care acționează prin înmulțire (la stânga, asupra unui vector coloană 4×1 , ce reprezintă coordonate omogene).
- ▶ Componerea transformărilor corespunde înmulțirii matricelor.

Texturare

Mihai-Sorin Stupariu

Sem. I, 2024 - 2025

Principii generale

Etape și funcții asociate

Coordinate de modelare și coordinate de texturare

Concluzii

Ce este o textură?

- ▶ În general este o entitate (imagine externă) / generată procedural “aplicată” pe primitivele randate.
- ▶ O textură are o structură discretă, ca o imagine = **structură de texeli**.
- ▶ Pentru a putea “lega” textura de imagine, aceasta are nevoie de **coordonate de texturare**. Prin referire la textura dată, acestea sunt între 0 și 1.

Elemente necesare

Folosirea unei biblioteci dedicate (de exemplu *SOIL – Simple OpenGL Image Library*) permite incarcarea rapida a unor texturi din fisiere avand formate standard, precum JPEG, PNG, etc.

Template-ul pus la dispoziție are integrate această bibliotecă (fisierele lib aferente și fișierul SOIL.h utilizat ca fișier de tip header în proiect.).

Etape - sinteză

Din punctul de vedere al implementării, trebuie urmărite o serie de etape:

Etape - sinteză

Din punctul de vedere al implementării, trebuie urmărite o serie de etape:

- ▶ indicarea, în VBO a legăturii dintre coordonatele vârfurilor (x, y, z, w) și coordonatele de texturare (s, t) pentru fiecare vârf (slide 5),

Etape - sinteză

Din punctul de vedere al implementării, trebuie urmărite o serie de etape:

- ▶ indicarea, în VBO a legăturii dintre coordonatele vârfurilor (x, y, z, w) și coordonatele de texturare (s, t) pentru fiecare vârf (slide 5),
- ▶ crearea unei texturi sau a unui obiect textură și specificarea caracteristicilor texturii, precum și indicarea modului în care textura este aplicată pe pixelii imaginii (slide 6),

Etape - sinteză

Din punctul de vedere al implementării, trebuie urmărite o serie de etape:

- ▶ indicarea, în VBO a legăturii dintre coordonatele vârfurilor (x, y, z, w) și coordonatele de texturare (s, t) pentru fiecare vârf (slide 5),
- ▶ crearea unei texturi sau a unui obiect textură și specificarea caracteristicilor texturii, precum și indicarea modului în care textura este aplicată pe pixelii imaginii (slide 6),
- ▶ activarea texturării / transmiterea către shader a obiectului textură ca variabilă uniformă (slide 18).

Asocierea coordonatelor de texturare pentru fiecare vârf

În funcția de creare a *VAO* / *VBO* sunt indicate, pentru fiecare vârf, coordonatele de texturare aferente (trebuie să existe o coerență între coordonatele vârfurilor și modul de alegere a coordonatelor de texturare).

ACEste coordonate au asociată o locație specifică, urmând să devină *vertex attributes* (în shader-ul de vârfuri). În codul sursă 04_04_texturare.cpp coordonatele de texturare au locația 2.

Template - funcția LoadTexture

Funcția LoadTexture() conține elementele necesare creării obiectului textură (slide 7 și următoarele), incluzând generarea și legarea texturii, precum și precizarea proprietăților acesteia.

Indicarea modului în care textura este aplicată pe pixelii imaginii (glTexParameter, slide 13 și următoarele).

Nu trebuie uitată eliberarea memoriei și reallocarea.

LoadTexture() poate fi apelată la inițializare.

Crearea unui obiect textură și specificarea proprietăților acestuia

Obiectele textură memorează date referitoare la textură, făcându-le accesibile imediat. Fiecare obiect îi este asociată o singură textură. Sunt parcursi câțiva pași intermediari, descriși în continuare, împreună cu funcțiile OpenGL asociate.

Generarea numelor obiectelor textură

:

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

Prin această funcție sunt *rezervați* n identificatori de textură în vectorul `texNames`, declarat explicit în procedura de inițializare `init`. Rezervarea are ca semnificație faptul că numele din `texNames` sunt marcate ca fiind utilizate, ele primesc caracteristici efective atunci când sunt prima dată legate cu funcțiile specifice. **Observație.** De menționat că funcția OpenGL cu efect contrar celei descrise este funcția de stergere

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

Crearea și utilizarea obiectelor textură

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

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

Specificarea caracteristicilor unei texturi 1D

Definirea unei texturi 1D se face apelând funcția

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

Funcția `glTexImage1D` are parametrii

- ▶ target: tipul de textură (`GL_TEXTURE_1D`);
- ▶ level: nivelul de texturare (0);
- ▶ intformat: numărul valorilor (de culoare) utilizate pentru fiecare pixel; în cazul codului RGBA acest număr este 4;
- ▶ width: lățimea este o putere a lui 2; trebuie să fie coerentă și consistentă cu modul în care a fost definită textura;
- ▶ border: este un număr egal cu 0,1 sau 2 și indică numărul pixelilor de pe frontieră;
- ▶ format: poate fi o constantă simbolică de forma `GL_RGB`; `GL_RGBA`;
- ▶ type: tipul este indicat printr-o constantă simbolică `GL_UNSIGNED_BYTE`; `GL_BYTE`;
- ▶ texels: se indică unde este localizată textura.

Specificarea caracteristicilor unei texturi 2D

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

Funcția `glTexImage2D` are parametrii

- ▶ target: tipul de textură (`GL_TEXTURE_2D`);
- ▶ level,intformat: similar cu cazul 1D;
- ▶ width, height: lățimea și înălțimea texturii;
- ▶ border: dimensiunea frontierei (b_w, b_h);
- ▶ format, type, texels: similar cu cazul 1D.

Subtexturi

Trebuie menționat faptul că pot fi definite și subtexturi, prin funcția

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

având parametrii similari

- ▶ target: tipul de textură (GL_TEXTURE_2D);
- ▶ level: similar cu funcția glTexImage2D;
- ▶ xoffset, yoffset: sunt numere întregi, pozitive, care precizează unde anume este așezată subtextura în structura de texeli (cu convenția că (0,0) este în stânga jos);
- ▶ width, height: lățimea și înălțimea subtexturii;
- ▶ format, type, texels: similar cu funcția glTexImage2D.

Semnificația acesti funcții este că se definește o structură prin care este înlocuită parțial / total o porțiune a texturii active (curente).

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

Două reguli sunt luate în considerare: *repetarea texturii și filtrare*.

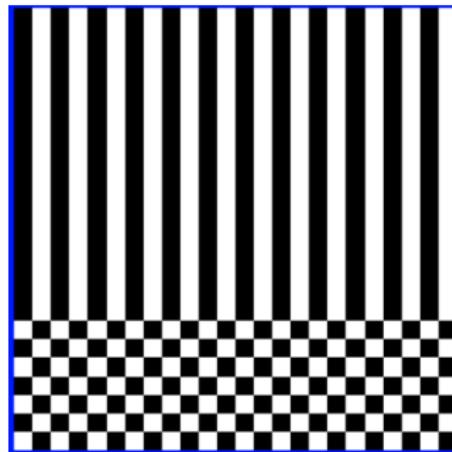
Ambele sunt legate de faptul că atât structura de texeli cât și cea de pixeli sunt discrete, iar scopul lor este de a indica modul în care se procedează atunci când nu există o corespondență 1:1 între cele două structuri. Forma generală a funcției asociate este

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

Parametrul target indică tipul de textură, în cazul 2D (considerat în continuare) acesta fiind GL_TEXTURE_2D. Celalți parametri depind ca semnificație și valoare de regula la care se face referire.

Repetarea texturii.

Motivație. Coordonatele de texturare, asociate inițial texturii, sunt reprezentate de pătratul $[0.0, 1.0] \times [0.0, 1.0]$. Cum se poate proceda în cazul în care sunt indicate valori în afara acestui pătrat? - detalii teoretice și exemple în secțiunea 3.



Repetarea texturii.

În acest caz (și ținând cont că suntem în context bidimensional) parametrul poate lua valorile

GL_TEXTURE_WRAP_S

GL_TEXTURE_WRAP_T

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

Parametrul value poate avea una din valorile

GL_CLAMP

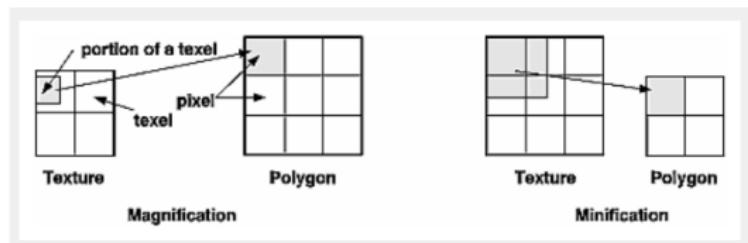
GL_REPEAT

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

Filtrare.

Motivație. Aici apare explicit faptul ca se poate întâmpla ca la randarea unei scene să nu existe o corespondență biunivocă între texeli (elementele texturii) și pixeli (elementele buffer-ului de cadru).

Astfel, se poate ca un pixel să corespundă unei porțiuni mici a structurii de texeli sau, invers, un pixel să corespundă unei structuri de texeli, iar rolul regulii de filtrare este de a stabili cum anume structura de texeli este transferată pe cea de pixeli.



Sursa: *OpenGL RedBook*

Filtrare.

Parametrul `pname` poate avea, corespunzător celor două situații, valorile

`GL_TEXTURE_MAG_FILTER`

`GL_TEXTURE_MIN_FILTER`

Parametrul `value` poate avea una din valorile

`GL_NEAREST`

`GL_LINEAR`

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

Utilizarea propriu-zisă a texturii

În funcția de desenare/randare: textura trebuie activată / legată folosind funcțiile `glActiveTexture()` și `glBindTexture()`.

Ulterior, trebuie transmisă shader-ului de fragment ca variabilă de tip uniform - exemplu:

```
glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0) .
```

Comunicare cu shader-ele

Shader-ul de vârfuri: i se transmit, pe lângă coordonatele vârfurilor și culori, coordonatele de texturare (v. atributele); ca output sunt și poziția și culoarea și coordonatele de texturare.

Shader-ul de fragmente: are ca date de intrare atât informațiile transmise de shader-ul de vârfuri, cât și textura – folosind o variabilă uniformă (`uniform sampler2D`). Se poate folosi funcția `mix` pentru a “combina” culoarea sau diferite texturi.

Coordinate de modelare și coordonate de texturare

- ▶ Coordonatele de modelare sunt cele utilizate în mod curent în funcția de desenare. Fiecărui vârf îi sunt asociate, aşa cum se știe, coordonatele spațiale (x, y) în cazul scenelor 2D).

Coordonate de modelare și coordonate de texturare

- ▶ Coordonatele de modelare sunt cele utilizate în mod curent în funcția de desenare. Fiecărui vârf îi sunt asociate, aşa cum se ştie, coordonatele spațiale (x, y) în cazul scenelor 2D.
- ▶ În cazul texturilor 2D coordonatele de texturare sunt s și t . Prin referire strict la **textură**, ambele sunt în intervalul $[0.0, 1.0]$. Cu alte cuvinte, coordonatele de texturare (teoretic) sunt reprezentate de pătratul $[0.0, 1.0] \times [0.0, 1.0]$.

Coordonate de modelare și coordonate de texturare

- ▶ Coordonatele de modelare sunt cele utilizate în mod curent în funcția de desenare. Fiecărui vârf îi sunt asociate, aşa cum se ştie, coordonatele spațiale (x, y) în cazul scenelor 2D.
- ▶ În cazul texturilor 2D coordonatele de texturare sunt s și t . Prin referire strict la **textură**, ambele sunt în intervalul $[0.0, 1.0]$. Cu alte cuvinte, coordonatele de texturare (teoretic) sunt reprezentate de pătratul $[0.0, 1.0] \times [0.0, 1.0]$.
- ▶ În cadrul funcției de desenare, fiecărui vârf îi sunt asociate anumite coordonate de texturare. Când se realizează “legarea” de vârfuri pot fi considerate coordonate arbitrarе, pe baza regulilor menționate anterior (v. slide-ul 15).

Coordonate de modelare și coordonate de texturare

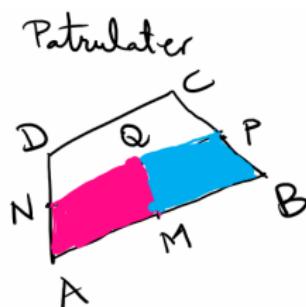
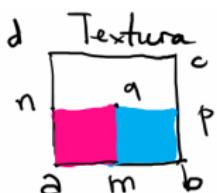
- ▶ Coordonatele de modelare sunt cele utilizate în mod curent în funcția de desenare. Fiecărui vârf îi sunt asociate, aşa cum se ştie, coordonatele spațiale (x, y) în cazul scenelor 2D.
- ▶ În cazul texturilor 2D coordonatele de texturare sunt s și t . Prin referire strict la **textură**, ambele sunt în intervalul $[0.0, 1.0]$. Cu alte cuvinte, coordonatele de texturare (teoretic) sunt reprezentate de pătratul $[0.0, 1.0] \times [0.0, 1.0]$.
- ▶ În cadrul funcției de desenare, fiecărui vârf îi sunt asociate anumite coordonate de texturare. Când se realizează “legarea” de vârfuri pot fi considerate coordonate arbitrarе, pe baza regulilor menționate anterior (v. slide-ul 15).
- ▶ Este suficient, pentru o primitivă grafică 2D, să fie indicate coordonatele de texturare pentru trei dintre vârfuri, coordonatele de texturare ale punctelor din interior rezultă automat. Motivație: dacă au fost fixate trei puncte necoliniare a, b, c în spațiul de texturare și trei puncte necoliniare A, B, C în spațiul de modelare, există o unică aplicație afină care transformă punctele a, b, c în A, B , respectiv C . Dacă sunt mai mult de trei vârfuri: coerentă!

Despre aplicarea texturii

Coordonatele de texturare (teoretic) sunt reprezentate de pătratul $[0.0, 1.0] \times [0.0, 1.0]$. În cadrul funcției de desenare, fiecărui vârf îi sunt asociate anumite coordonate de texturare (valorile pot fi arbitrale, fiind aplicate regulile legate de repetare (silde 15)).

Despre aplicarea texturii

Este suficient, pentru o primitivă grafică 2D, să fie indicate coordonatele de texturare pentru trei dintre vârfuri, coordonatele de texturare ale punctelor din interior rezultă automat, folosind coliniaritatea și rapoarte de puncte coliniare. Dacă sunt mai mult de trei vârfuri: coerență!

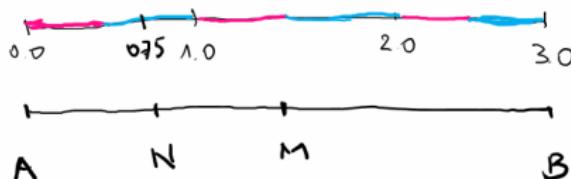


- Vf. A îi coresp. a, lui B...b, etc.
- Raporturile din textură sunt "preluate" pt. patrulater

Textura $abcd$ este aplicată pe patrulaterul $ABCD$, modul de aplicare fiind dat de păstrarea coliniarității și a rapoartelor de puncte coliniare.

Un exemplu numeric

Se aplică o textură 1D  așa încât vârfurile $A = (30, 50)$ și $B = (70, 40)$ au coordonata de texturare 0.0, respectiv 3.0, opțiunea fiind GL_REPEAT. Fie N mijlocul lui $[AM]$, unde M este mijlocul lui $[AB]$. Ce culoare are N ?



N are coord. de texturare 0.75,
deci va fi reprezentat cu albastru

Vârf	Coord. test.
A	0.0
B	3.0
M	1.5
<i>mijl. [AB]</i>	
<i>N</i>	0.75
<i>mijl. [AM]</i>	

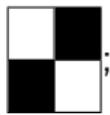
Concluzie

În concluzie, o textură 2D este:

- (i) un tablou dreptunghiular de texeli;
- (ii) o mulțime înzestrată cu coordonate de texturare (pătratul standard).

Atunci când tabloul de texeli are același număr de linii / coloane și atunci când textura este aplicată direct pe un pătrat, corespondența dintre pixeli și texeli este ușor de controlat și nu apar fenomene de deformare. În caz contrar (situație mai des întâlnită), trebuie manevrați în mod convenabil parametrii disponibili.

Exerciții

1. Se presupune că punctelor $(8.0, 7.0)$, $(6.0, 11.0)$, $(13.0, 13.0)$ din spațiul de modelare le sunt asociate coordonatele de texturare $(0.2, 0.4)$, $(0.6, 0.8)$, respectiv $(0.2, 0.2)$. Care sunt coordonatele de texturare ale punctului $(10.0, 11.0)$?
2. Se presupune că am generat o textură reprezentând o tablă de șah 8×8 și că aceasta este apelată folosind coordonatele de texturare $(0.0, 0.0)$, $(2.0, 0.0)$, $(2.0, 2.0)$, $(0.0, 2.0)$ și opțiunea `GL_REPEAT`. Câte pătrățele albe apar? (fondul este albastru)
3. Pe un fundal verde este desenat un pătrat folosind textura ; coordonatele de texturare asociate vârfurilor păratului sunt $(0.0, 0.0)$, $(2.0, 0.0)$, $(2.0, 2.0)$, $(0.0, 2.0)$, iar opțiunea utilizată este `GL_CLAMP`. Care este raportul dintre suprafața colorată cu alb și cea colorată cu negru?

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

Mihai-Sorin Stupariu

Sem. I, 2024 - 2025

Coordinate de modelare. Coordinate de vizualizare

Reperul de vizualizare

Transformarea de vizualizare

Exemple

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 $\mathbf{u}, \mathbf{v}, \mathbf{n}$ (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: $P_0 = (0, 0, 0)$, $P_{ref} = (0, 0 - 1)$, $V = (0, 1, 0)$

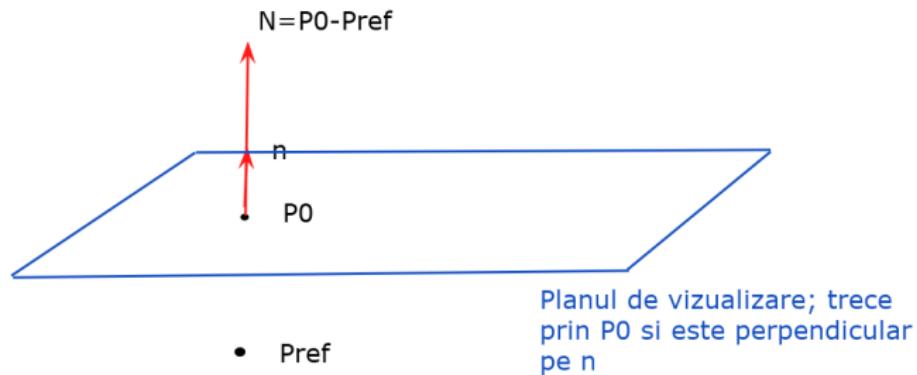
Funcția `glm::lookAt()`

- ▶ 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()`;

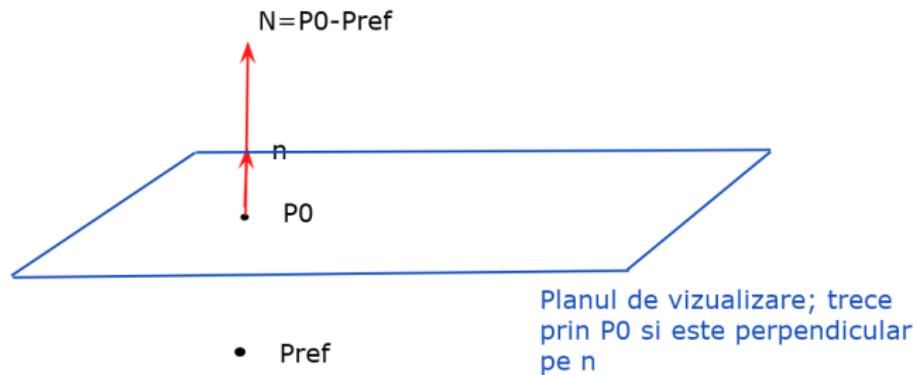
Funcția `glm::lookAt()`

- ▶ 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 $\mathbf{u}, \mathbf{v}, \mathbf{n}$

Reperul de vizualizare - vectorul n

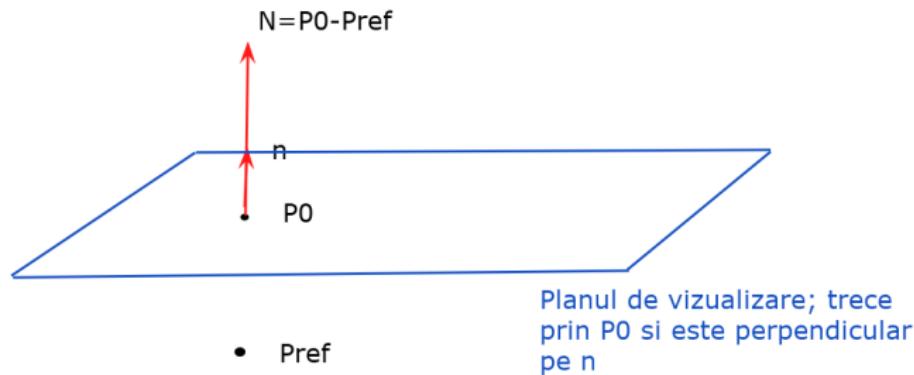


Reperul de vizualizare - vectorul n



$$N = \overrightarrow{P_{ref}P_0} = P_0 - P_{ref}; \quad n = \frac{N}{\|N\|}$$

Reperul de vizualizare - vectorul n



$$N = \overrightarrow{P_{ref}P_0} = P_0 - P_{ref}; \quad n = \frac{N}{\|N\|}$$

Comentarii: de ce $P_0 - P_{ref}$ și nu $P_{ref} - P_0$? 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 \mathbf{v} , \mathbf{u}

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

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

Reperul de vizualizare - vectorii \mathbf{v} , \mathbf{u}

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

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

- ▶ **al doilea versor \mathbf{v}** - verticala “reală” din planul de vizualizare

$$\mathbf{v} = \mathbf{n} \times \mathbf{u}$$

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?

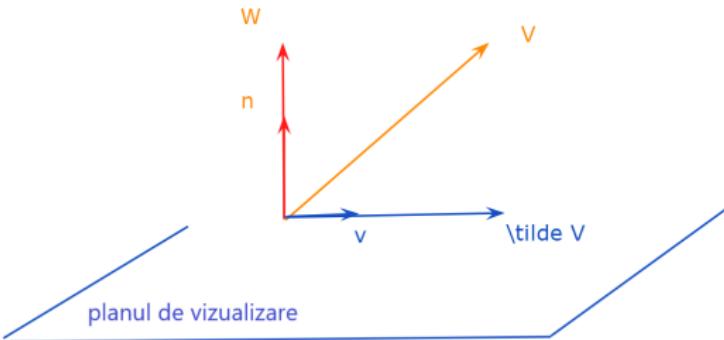
R: Vectorul V se descompune ca suma dintre un vector \tilde{V} (=proiecția lui V pe planul de vizualizare) și un vector W , perpendicular pe planul de vizualizare (coliniar cu n), altfel spus $V = \tilde{V} + W$. Are loc relația $v = \frac{\tilde{V}}{\|\tilde{V}\|}$.

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?

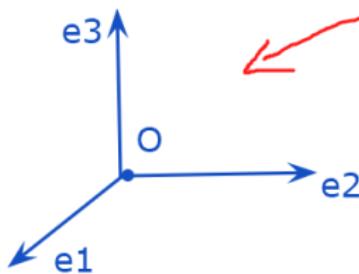
R: Vectorul V se descompune ca suma dintre un vector \tilde{V} (=proiecția lui V pe planul de vizualizare) și un vector W , perpendicular pe planul de vizualizare (coliniar cu \mathbf{n}), altfel spus $V = \tilde{V} + W$. Are loc relația $\mathbf{v} = \frac{\tilde{V}}{\|\tilde{V}\|}$.

Obs: Dacă modificăm vectorul V , adăugând multipli ai lui N (sau \mathbf{n} , deoarece N și \mathbf{n} sunt coliniari), vectorul \mathbf{v} nu se modifică.

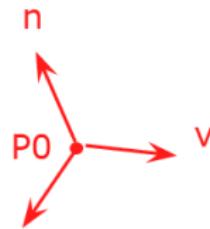


Schimbarea reperului ca transformare

Schimbarea de reper \leftrightarrow Efectuarea unei transformări



Reperul de modelare (canonic)

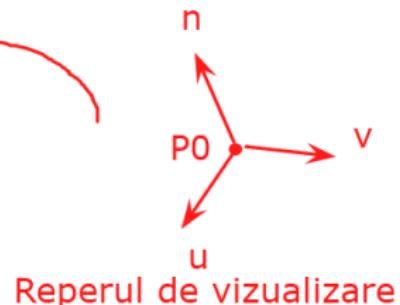
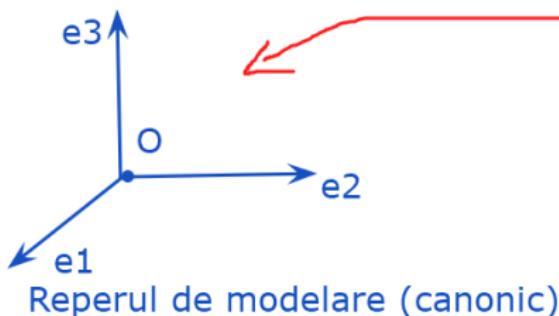


Reperul de vizualizare

“Aducem reperul de vizualizare ca să se suprapună peste reperul de modelare”.

Schimbarea reperului ca transformare

Schimbarea de reper \leftrightarrow Efectuarea unei transformări

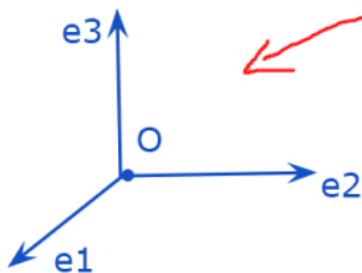


“Aducem reperul de vizualizare ca să se suprapună peste reperul de modelare”.

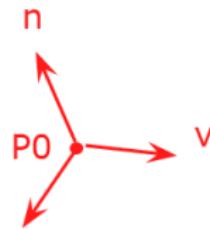
Descrierea transformărilor:

Schimbarea reperului ca transformare

Schimbarea de reper \leftrightarrow Efectuarea unei transformări



Reperul de modelare (canonic)



Reperul de vizualizare

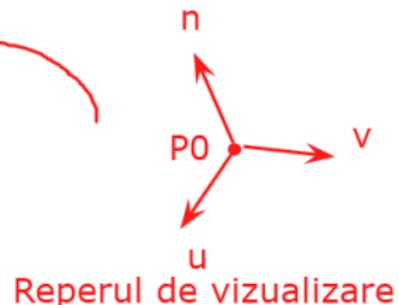
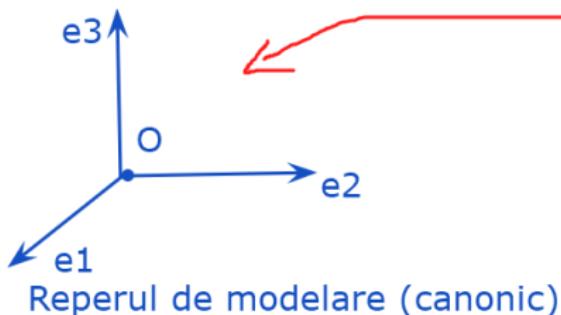
"Aducem reperul de vizualizare ca să se suprapună peste reperul de modelare".

Descrierea transformărilor:

- ▶ translatăm astfel încât P_0 să devină originea, adică aplicăm
 $T_{(-x_0, -y_0, -z_0)}$

Schimbarea reperului ca transformare

Schimbarea de reper \leftrightarrow Efectuarea unei transformări



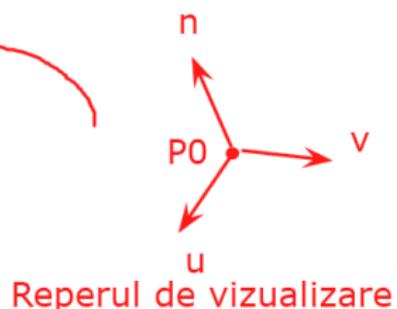
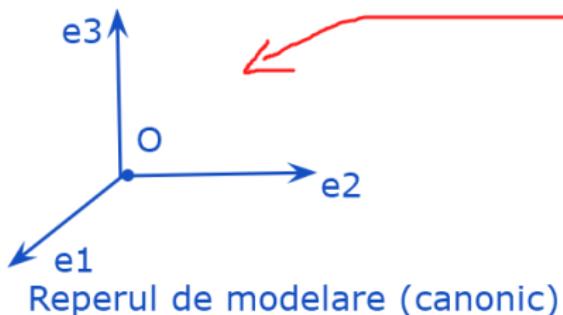
"Aducem reperul de vizualizare ca să se suprapună peste reperul de modelare".

Descrierea transformărilor:

- ▶ translăm astfel încât P_0 să devină originea, adică aplicăm $T_{(-x_0, -y_0, -z_0)}$
- ▶ aplicăm o rotație 3D R astfel încât reperul ortonormat $(\mathbf{u}, \mathbf{v}, \mathbf{n})$ să se suprapună cu reperul ortonormat (e_1, e_2, e_3) (reperul canonic) - un reper este ortonormat dacă vectorii sunt perpendiculari 2 către 2 și de normă 1.

Schimbarea reperului ca transformare

Schimbarea de reper \leftrightarrow Efectuarea unei transformări



"Aducem reperul de vizualizare ca să se suprapună peste reperul de modelare".

Descrierea transformărilor:

- ▶ translăm astfel încât P_0 să devină originea, adică aplicăm $T_{(-x_0, -y_0, -z_0)}$
- ▶ aplicăm o rotație 3D R astfel încât reperul ortonormat (u, v, n) să se suprapună cu reperul ortonormat (e_1, e_2, e_3) (reperul canonic) - un reper este ortonormat dacă vectorii sunt perpendiculari 2 către 2 și de normă 1.

Care este matricea asociată?

Matricea asociată schimbării de reper

Matricele asociate celor două transformări:

- pentru translația $\mathbf{T}_{(-x_0, -y_0, -z_0)}$

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

Matricea asociată schimbării de reper

- pentru rotația \mathbf{R} :

Matricea asociată schimbării de reper

- pentru rotația \mathbf{R} :

- matricea 3×3 care transformă reperul (e_1, e_2, e_3) în reperul ortonormat $(\mathbf{u}, \mathbf{v}, \mathbf{n})$ este

$$A = \begin{pmatrix} \mathbf{u}_x & \mathbf{v}_x & \mathbf{n}_x \\ \mathbf{u}_y & \mathbf{v}_y & \mathbf{n}_y \\ \mathbf{u}_z & \mathbf{v}_z & \mathbf{n}_z \end{pmatrix}$$

(coloanele acestei matrice sunt componentele lui $\mathbf{u}, \mathbf{v}, \mathbf{n}$ în reperul canonic)

Matricea asociată schimbării de reper

- pentru rotația \mathbf{R} :

- matricea 3×3 care transformă reperul (e_1, e_2, e_3) în reperul ortonormat $(\mathbf{u}, \mathbf{v}, \mathbf{n})$ este

$$A = \begin{pmatrix} \mathbf{u}_x & \mathbf{v}_x & \mathbf{n}_x \\ \mathbf{u}_y & \mathbf{v}_y & \mathbf{n}_y \\ \mathbf{u}_z & \mathbf{v}_z & \mathbf{n}_z \end{pmatrix}$$

(coloanele acestei matrice sunt componentele lui $\mathbf{u}, \mathbf{v}, \mathbf{n}$ în reperul canonic)

- matricea care transformă $(\mathbf{u}, \mathbf{v}, \mathbf{n})$ în (e_1, e_2, e_3) este A^{-1}

Matricea asociată schimbării de reper

- pentru rotația \mathbf{R} :

- matricea 3×3 care transformă reperul (e_1, e_2, e_3) în reperul ortonormat $(\mathbf{u}, \mathbf{v}, \mathbf{n})$ este

$$A = \begin{pmatrix} \mathbf{u}_x & \mathbf{v}_x & \mathbf{n}_x \\ \mathbf{u}_y & \mathbf{v}_y & \mathbf{n}_y \\ \mathbf{u}_z & \mathbf{v}_z & \mathbf{n}_z \end{pmatrix}$$

(coloanele acestei matrice sunt componentele lui $\mathbf{u}, \mathbf{v}, \mathbf{n}$ în reperul canonic)

- matricea care transformă $(\mathbf{u}, \mathbf{v}, \mathbf{n})$ în (e_1, e_2, e_3) este A^{-1}
- întrucât $(\mathbf{u}, \mathbf{v}, \mathbf{n})$ este reper ortonormat, A este matrice **ortogonală**, adică matricea A verifică relația $A^t \cdot A = \mathbb{I}_3$ (verificați!), iar matricea inversă este $A^{-1} = A^t$
- din A^t se construiește (în mod natural) matricea 4×4 asociată rotației, M_R

Matricea asociată schimbării de reper

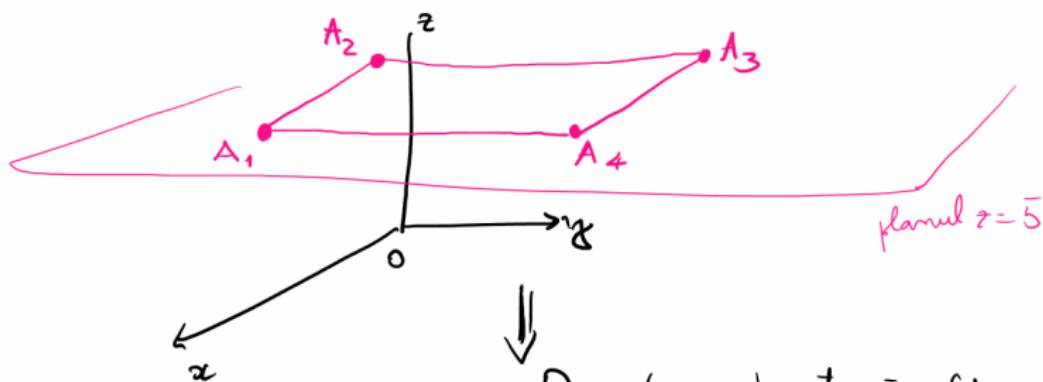
În final, matricea 4×4 asociată transformării de vizualizare este

$$\begin{aligned}
 M &= M_R \cdot M_{T_{(-x_0, -y_0, -z_0)}} = \begin{pmatrix} \mathbf{u}_x & \mathbf{u}_y & \mathbf{u}_z & 0 \\ \mathbf{v}_x & \mathbf{v}_y & \mathbf{v}_z & 0 \\ \mathbf{n}_x & \mathbf{n}_y & \mathbf{n}_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \\
 &= \begin{pmatrix} \mathbf{u}_x & \mathbf{u}_y & \mathbf{u}_z & -\langle \mathbf{u}, P_0 \rangle \\ \mathbf{v}_x & \mathbf{v}_y & \mathbf{v}_z & -\langle \mathbf{v}, P_0 \rangle \\ \mathbf{n}_x & \mathbf{n}_y & \mathbf{n}_z & -\langle \mathbf{n}, P_0 \rangle \\ 0 & 0 & 0 & 1 \end{pmatrix}
 \end{aligned}$$

Exemplul 1. Cod sursă 06_01_poligoane3D.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 06_01_poligoane3D.cpp

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

- scriem ecuația planului sub forma $\underbrace{Ax + By + Cz + D = 0}_{\pi(x, y, z)}$

- folosim determinantul

$$\begin{array}{c} \left| \begin{array}{cccc} x & y & z & 1 \\ 5 & -5 & 5 & 1 \\ -5 & -5 & 5 & 1 \\ -5 & 5 & 5 & 1 \end{array} \right| = \dots = x \cdot 0 - y \cdot 0 + \\ (calculat!) + z \cdot (-100) - \\ - (-500) \end{array}$$

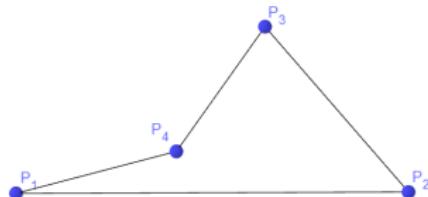
- avem $\pi(x, y, z) = -100z + 500$

- avem $\pi(0, 0, 0) = 500 > 0 \rightarrow$ punctul $(0, 0, 0)$ este
în fața poligonului

Exemplul 2. Cod sursă 06_02_poligoane3D_exemplu2.cpp

Fie punctele $P_1 = (6, 2, 0)$, $P_2 = (-4, 4, 8)$, $P_3 = (0, 0, 8)$ (toate trei situate în planul de ecuație $x + y + z = 8$).

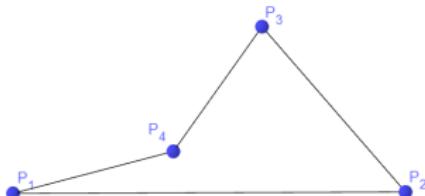
- a) Să se aleagă P_4 astfel ca patrulaterul $P_1P_2P_3P_4$ să fie concav.



Exemplul 2. Cod sursă 06_02_poligoane3D_exemplu2.cpp

Fie punctele $P_1 = (6, 2, 0)$, $P_2 = (-4, 4, 8)$, $P_3 = (0, 0, 8)$ (toate trei situate în planul de ecuație $x + y + z = 8$).

- a) Să se aleagă P_4 astfel ca patrulaterul $P_1P_2P_3P_4$ să fie concav.



Dacă alegem un punct P_4 în interiorul triunghiului $P_1P_2P_3$ (combinația convexă a punctelor P_1, P_2, P_3 , cu coeficienți > 0 cu suma 1), atunci patrulaterul $P_1P_2P_3P_4$ este concav. De exemplu, punctul P_4 poate fi ales ca fiind dat de

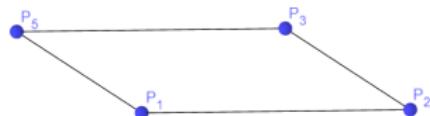
$$P_4 = \frac{1}{2}P_1 + \frac{1}{4}P_2 + \frac{1}{4}P_3$$

(P_4 este mijlocul segmentului $[P_1Q]$, unde Q este mijlocul lui $[P_2P_3]$). Explicit, avem $P_4 = (2, 2, 4)$.

Exemplul 2. Cod sursă06_02_poligoane3D_exemplu2.cpp

Fie punctele $P_1 = (6, 2, 0)$, $P_2 = (-4, 4, 8)$, $P_3 = (0, 0, 8)$ (toate trei situate în planul de ecuație $x + y + z = 8$).

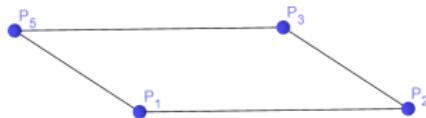
- b) Să se aleagă P_5 astfel ca patrulaterul $P_1P_2P_3P_5$ să fie convex.



Exemplul 2. Cod sursă06_02_poligoane3D_exemplu2.cpp

Fie punctele $P_1 = (6, 2, 0)$, $P_2 = (-4, 4, 8)$, $P_3 = (0, 0, 8)$ (toate trei situate în planul de ecuație $x + y + z = 8$).

b) Să se aleagă P_5 astfel ca patrulaterul $P_1P_2P_3P_5$ să fie convex.



Alegem P_5 astfel încât $P_1P_2P_3P_5$ să fie un paralelogram. Folosindu-ne de faptul că diagonalele unui paralelogram se taie în părți egale, adică

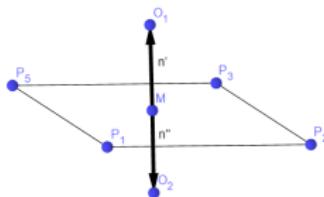
$$\frac{1}{2}P_1 + \frac{1}{2}P_3 = \frac{1}{2}P_2 + \frac{1}{2}P_5,$$

deducem $P_5 = P_1 + P_3 - P_2$, deci $P_5 = (10, -2, 0)$.

Exemplul 2. Cod sursă 06_02_poligoane3D_exemplu2.cpp

Fie punctele $P_1 = (6, 2, 0)$, $P_2 = (-4, 4, 8)$, $P_3 = (0, 0, 8)$ (toate trei situate în planul de ecuație $x + y + z = 8$).

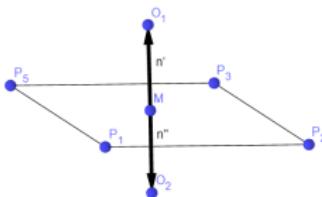
- c) Să se aleagă puncte O_1 și O_2 astfel ca poligonul $P_1P_2P_3P_5$ să fie văzut din față, respectiv din spate.



Exemplul 2. Cod sursă 06_02_poligoane3D_exemplu2.cpp

Fie punctele $P_1 = (6, 2, 0)$, $P_2 = (-4, 4, 8)$, $P_3 = (0, 0, 8)$ (toate trei situate în planul de ecuație $x + y + z = 8$).

- c) Să se aleagă puncte O_1 și O_2 astfel ca poligonul $P_1P_2P_3P_5$ să fie văzut din față, respectiv din spate.



Mai întâi calculăm produsul vectorial $\overrightarrow{P_1P_2} \times \overrightarrow{P_2P_3}$, care este egal cu $(32, 32, 32)$. Așadar, vectorul normal la plan este $n = (\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}})$.

Alegem convenabil un punct M din plan, un vector n' coliniar și de același sens cu n și un vector n'' coliniar și de sens opus cu n . Concret: $M = (3, 1, 4)$ (mijlocul segmentului $[P_1P_3]$), $n' = (10, 10, 10)$ și $n'' = (-10, -10, -10)$.

Definim O_1 astfel ca $\overrightarrow{MO_1} = n'$, adică $n' = O_1 - M$, așadar $O_1 = (3, 1, 4) + (10, 10, 10) = (13, 11, 14)$.

Analog, definim O_2 ca $\overrightarrow{MO_2} = n''$, deci $O_2 = (3, 1, 4) - (10, 10, 10) = (-7, -9, -6)$.

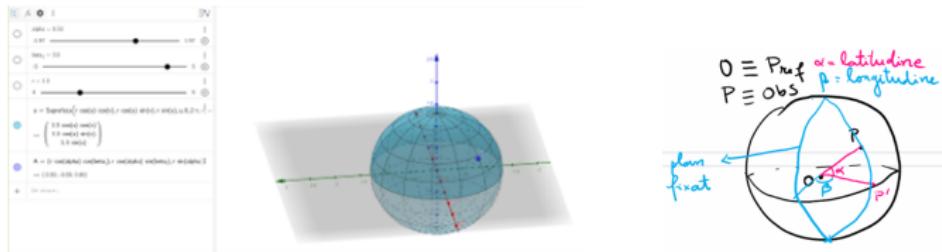
Survolarea unui obiect - codul 07_03_survolare_cub.cpp

- ▶ La ce revine a survola un obiect?

Survolarea unui obiect - codul 07_03_survolare_cub.cpp

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

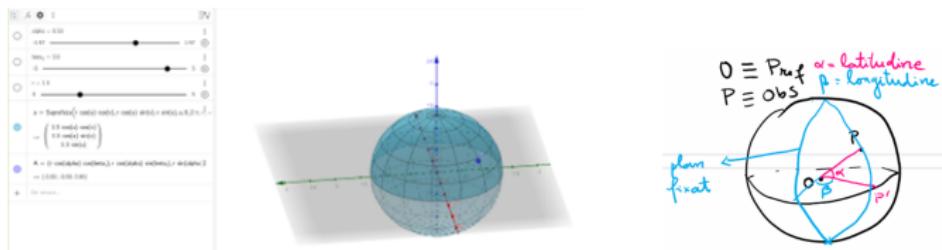
$$\begin{cases} x = C_x + r \cos(\alpha) \cos(\beta) \\ y = C_y + r \cos(\alpha) \sin(\beta) \\ z = C_z + r \sin(\alpha) \end{cases} \quad \alpha \in [-\frac{\pi}{2}, \frac{\pi}{2}], \beta \in [0, 2\pi]$$



Survolarea unui obiect - codul 07_03_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);
```

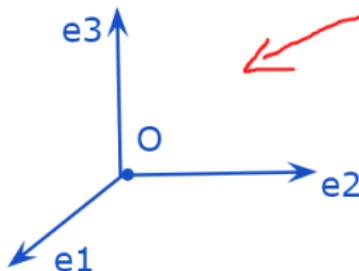
Transformări (IV). Proiecții

Mihai-Sorin Stupariu

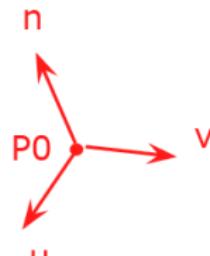
Sem. I, 2024 - 2025

Schimbarea reperului ca transformare

Schimbarea de reper \leftrightarrow Efectuarea unei transformări



Reperul de modelare (canonic)

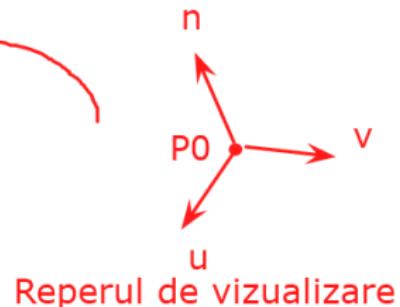
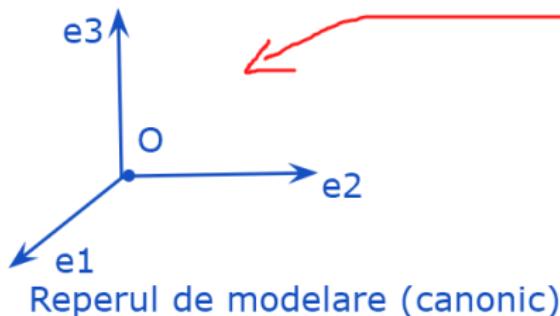


Reperul de vizualizare

"Aducem reperul de vizualizare ca să se suprapună peste reperul de modelare".

Schimbarea reperului ca transformare

Schimbarea de reper \leftrightarrow Efectuarea unei transformări

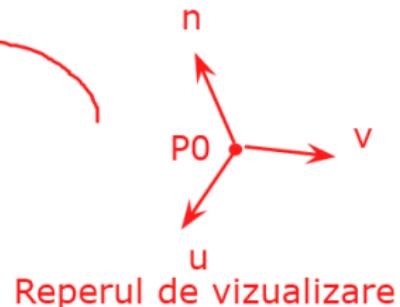
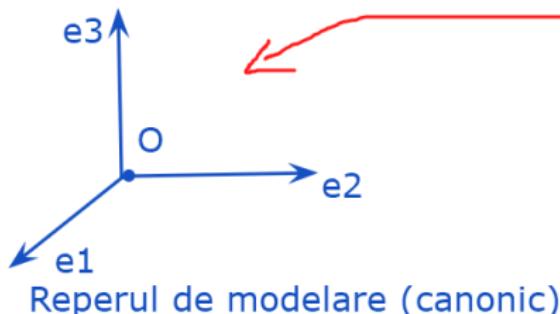


"Aducem reperul de vizualizare ca să se suprapună peste reperul de modelare".

Descrierea transformărilor:

Schimbarea reperului ca transformare

Schimbarea de reper \leftrightarrow Efectuarea unei transformări



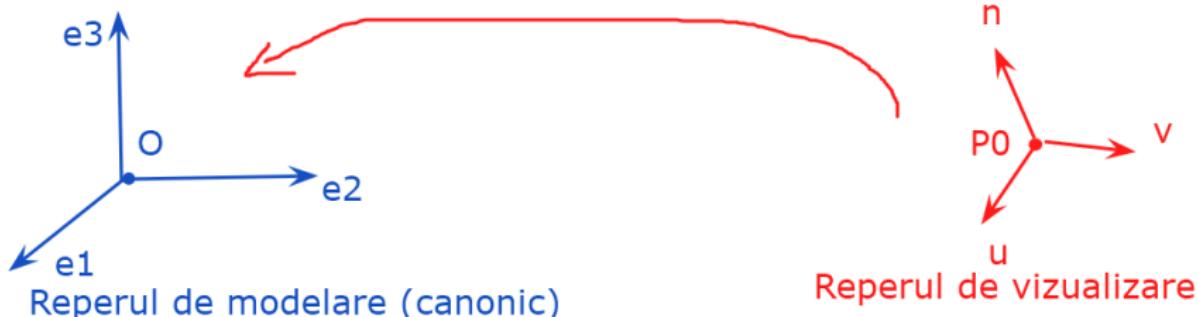
"Aducem reperul de vizualizare ca să se suprapună peste reperul de modelare".

Descrierea transformărilor:

- ▶ translatăm astfel încât P_0 să devină originea, adică aplicăm
 $T_{(-x_0, -y_0, -z_0)}$

Schimbarea reperului ca transformare

Schimbarea de reper \leftrightarrow Efectuarea unei transformări



"Aducem reperul de vizualizare ca să se suprapună peste reperul de modelare".

Descrierea transformărilor:

- ▶ translatăm astfel încât P_0 să devină originea, adică aplicăm $T_{(-x_0, -y_0, -z_0)}$
- ▶ aplicăm o rotație 3D R astfel încât reperul ortonormat (u, v, n) să se suprapună cu reperul ortonormat (e_1, e_2, e_3) (reperul canonic) - un reper este ortonormat dacă vectorii sunt perpendiculari 2 către 2 și de normă 1.

- 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 vizualizare, 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 vizualizare, 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 vizualizare, 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 `matrVizualizare` este matricea de vizualizare (dată de `glm::lookAt()`) și `matrProiectie` este matricea de proiecție, atunci în codul sursă trebuie să apară `matrProiectie * matrVizualizare`.

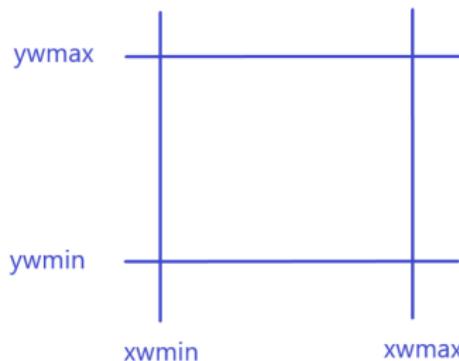
Cazul 2D

► `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.

Proiecții ortogonale 2D - exemple

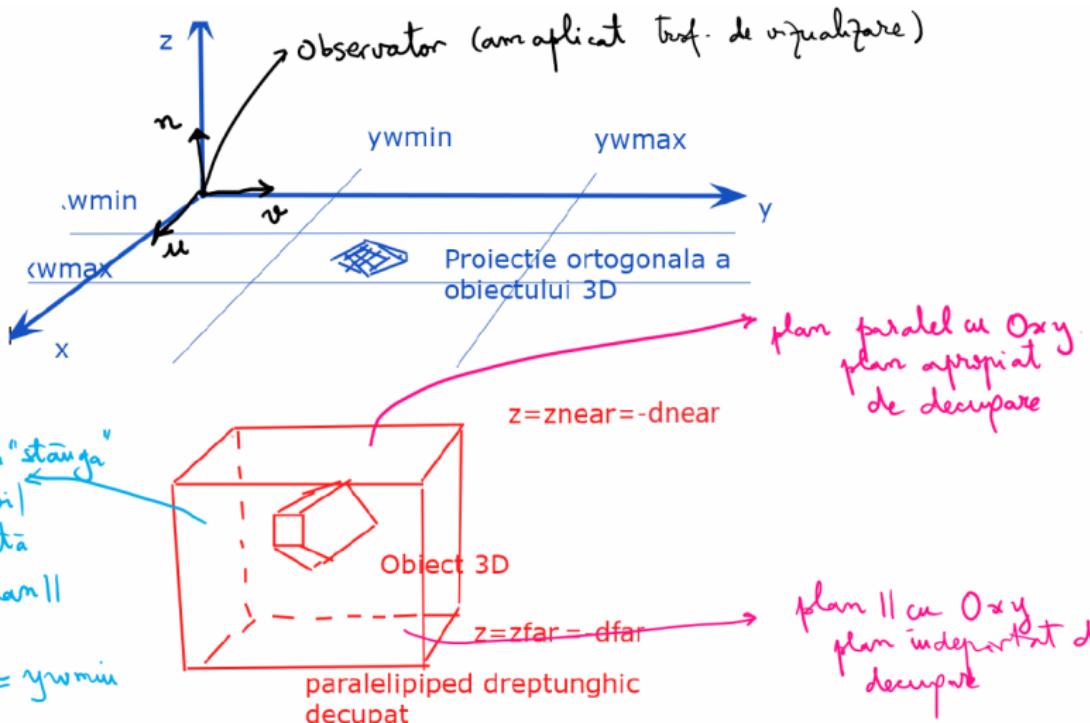
- ▶ Care este aria dreptunghiului decupat dacă se apelează funcția `glm::ortho (a, b, c, d)`? ($a < b, c < d$).

Proiecții ortogonale 2D - exemple

- ▶ Care este aria dreptunghiului ocupat dacă se apelează funcția `glm::ortho (a, b, c, d)`? ($a < b, c < d$).
- ▶ Ce diferențe sunt (din punctul de vedere al (i) dimensiunii scenei decupate, (ii) obiectelor - dimensiune, etc.) între apelarea funcției `glm::ortho (a, b, c, d)` și apelarea funcției `glm::ortho (2a, 2b, 2c, 2d)`? ($a < b, c < d$)

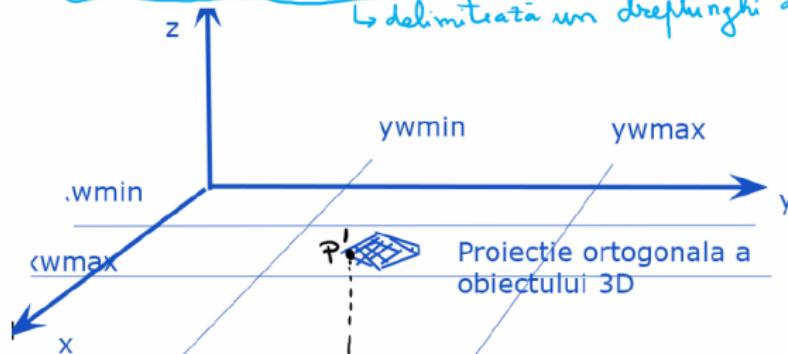
Cazul 3D - proiecții ortogonale

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



Ce este o proiecție ortogonală?

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



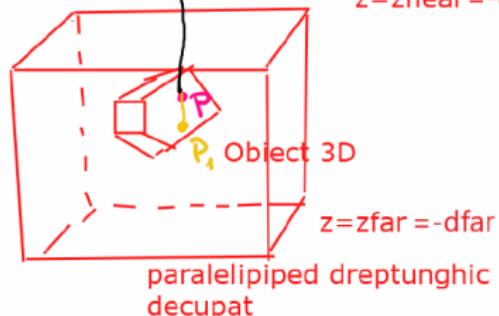
formula:

$$(x_p, y_p, z_p) \mapsto$$

$$\mapsto (x_p, y_p, 0)$$

||

$$(x_p, y_p)$$



P' = proiecție
ortogonală
a lui P
(se duc o
dreaptă core
tre ce printr P
și e \perp Oxy ...)

P' este și pr.
ortogonală
a lui P ,

Cazul 3D - proiecții ortogonale

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

Este decupat un **paralelipiped dreptunghic** delimitat de planele

$x = xW_{\min}, x = xW_{\max}; y = yW_{\min}, y = yW_{\max}$, respectiv $z = z_{\text{near}}$, unde $z_{\text{near}} = -d_{\text{near}}$, $z = z_{\text{far}}$, unde $z_{\text{far}} = -d_{\text{far}}$. Valorile implicate sunt $-1.0, 1.0, -1.0, 1.0, -1.0, 1.0$ (valori normalizate).

Matricea 4×4 asociată este

$$\mathcal{M}_{\text{ortho}} = \begin{pmatrix} \frac{2}{xW_{\max} - xW_{\min}} & 0 & 0 & -\frac{xW_{\max} + xW_{\min}}{xW_{\max} - xW_{\min}} \\ 0 & \frac{2}{yW_{\max} - yW_{\min}} & 0 & -\frac{yW_{\max} + yW_{\min}}{yW_{\max} - yW_{\min}} \\ 0 & 0 & -\frac{2}{d_{\text{far}} - d_{\text{near}}} & -\frac{d_{\text{far}} + d_{\text{near}}}{d_{\text{far}} - d_{\text{near}}} \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Matricea $\mathcal{M}_{\text{ortho}}$ are rolul de a transforma paralelipipedul dreptunghic decupat în paralelipipedul "standard" $[-1, 1] \times [-1, 1] \times [-1, 1]$, apoi, în mod implicit, sunt reținute primele două coordonate.

Proiecții ortogonale 3D - comentarii

- De discutat rolul elementelor care definesc paralelipipedul dreptunghic decupat.

Proiecții ortogonale 3D - comentarii

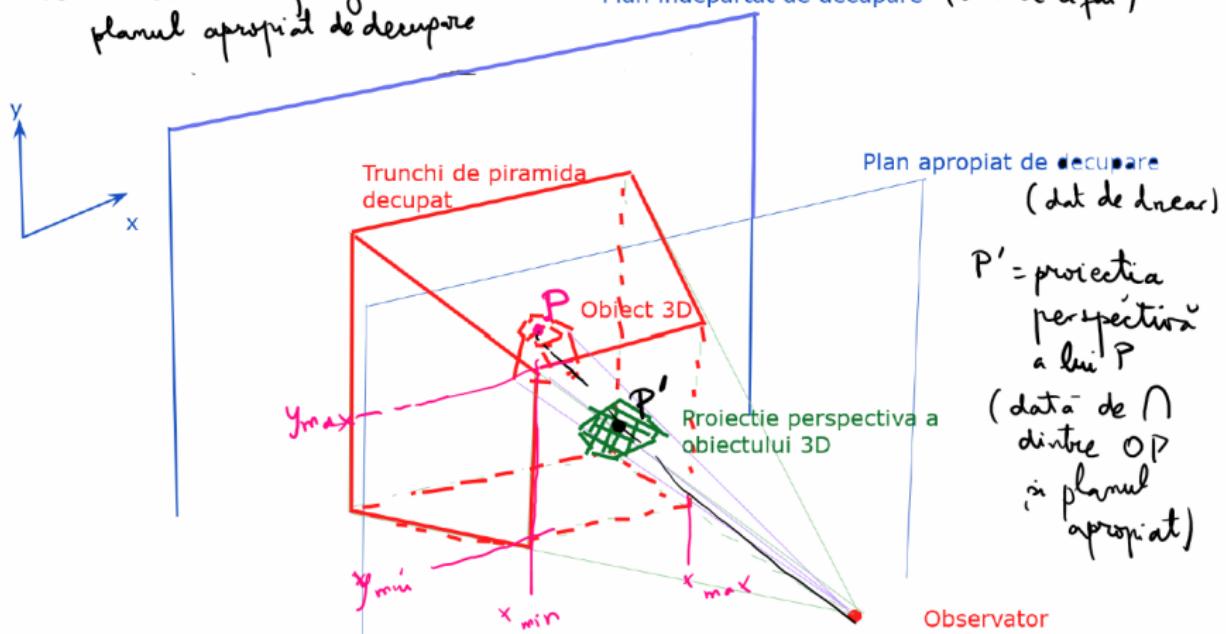
- ▶ De discutat rolul elementelor care definesc paralelipipedul dreptunghic decupat.
- ▶ De testat pe codul sursă `07_01_desenare_cub.cpp` cum este realizată proiecția dacă modificăm diversi parametri ai funcției `glm::ortho()`; . De urmărit: (i) cum este realizată decuparea; (ii) cum arată obiectul randat. De exemplu: ce se întâmplă dacă modificăm `dnear`?

Cazul 3D - proiecții perspective

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

delimită un dreptunghi din
planul apropiat de decupare

Plan îndepărtat de decupare (dat de $dfar$)



P' = proiecția perspectivă a lui P
(data de \cap dintre OP și planul apropiat)

Cazul 3D - proiecții perspective

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

Este decupat un **trunchi de piramidă** având vârful în origine (poziția observatorului). Bazele sunt date de planele $z = z_{\text{near}}$, unde $z_{\text{near}} = -d_{\text{near}}$, $z = z_{\text{far}}$, unde $z_{\text{far}} = -d_{\text{far}}$

Important: se delimitizează dreptunghiul din planul apropiat, având ecuațiile dreptelor de forma $z = z_{\text{near}}$, $x = xw_{\text{min}}$, etc., **apoi** se determină dreptunghiul din planul îndepărtat și trunchiul de piramidă.

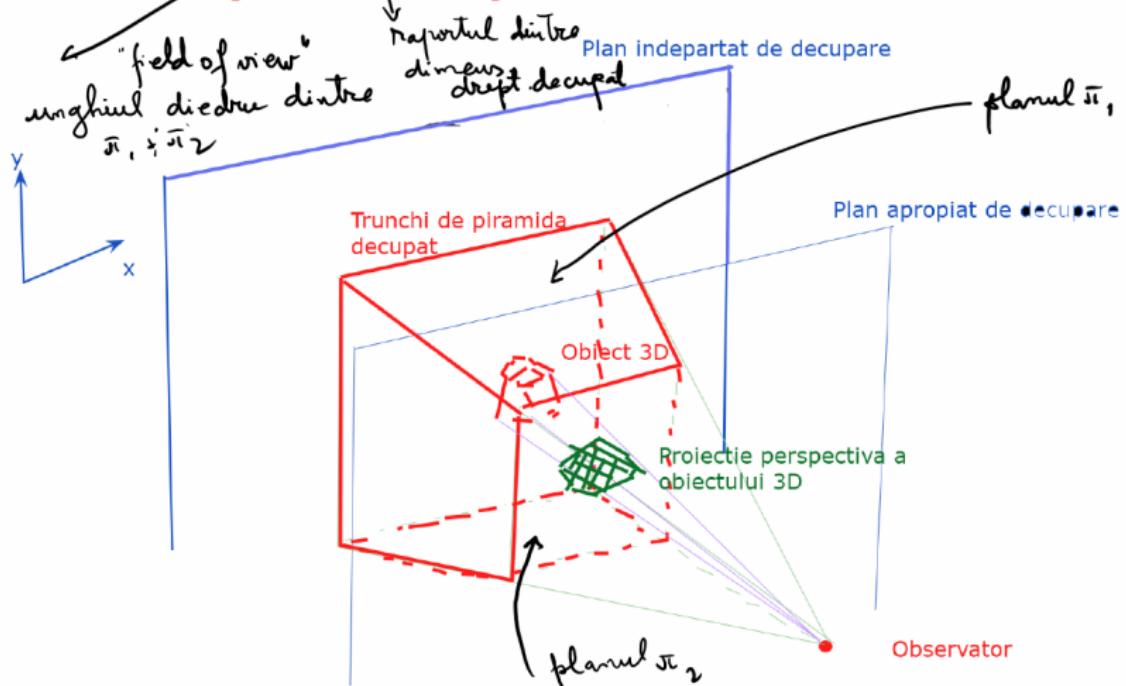
Matricea 4×4 asociată este

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

Matricea $\mathcal{M}_{\text{frustum}}$ are rolul de a transforma trunchiul de piramidă 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::perspective(fov, aspect, dnear, dfar);
glm::infinitePerspective(fov, aspect, dnear);
```



Cazul 3D - proiecții perspective

`glm::perspective(fov, aspect, dnear, dfar);`

`glm::perspective()` - este decupat un **trunchi de piramidă** decupat dintr-o **piramidă cu baza dreptunghi în care înălțimea dusă din vârful piramidei inițiale (observator) cade în centrul dreptunghiului.** “Deschiderea” piramidei este dată de `fov`, iar raportul lungimilor laturilor este dat de `aspect`.

Important: se delimitizează dreptunghiul din planul apropiat, **apoi** se determină dreptunghiul din planul îndepărtat și trunchiul de piramidă.

Matricea 4×4 asociată este

$$\mathcal{M}_{\text{perspective}} = \begin{pmatrix} \frac{\text{ctg}(\frac{\text{fov}}{2})}{\text{aspect}} & 0 & 0 & 0 \\ 0 & \text{ctg}(\frac{\text{fov}}{2}) & 0 & 0 \\ 0 & 0 & -\frac{d_{\text{far}} + d_{\text{near}}}{d_{\text{far}} - d_{\text{near}}} & -\frac{2d_{\text{near}}d_{\text{far}}}{d_{\text{far}} - d_{\text{near}}} \\ 0 & 0 & -1 & 0 \end{pmatrix}.$$

Matricea $\mathcal{M}_{\text{perspective}}$ are rolul de a transforma trunchiul de piramidă decupat în paralelipipedul “standard” $[-1, 1] \times [-1, 1] \times [-1, 1]$, apoi, în mod implicit, sunt reținute primele două coordonate.

Proiecții perspective - comentarii

- De discutat rolul elementelor care definesc trunchiul de piramidă decupat. De comentat diferența dintre `glm::frustum()`; - se poate decupa un trunchi de piramidă arbitrar și `glm::perspective()`; - se poate decupa un trunchi de piramidă care provine dintr-o piramidă având baza un dreptunghi, iar piciorul înălțimii piramidei (dusă din vârf) coincide cu centrul bazei. În cazul funcției `glm::perspective()`; sunt considerați ca parametri `fov=field of view`, un unghi între plane și `aspect=raportul` dintre lungimile laturilor dreptunghiului decupat.

Proiecții perspective - comentarii

- ▶ De discutat rolul elementelor care definesc trunchiul de piramidă decupat. De comentat diferența dintre `glm::frustum()`; - se poate decupa un trunchi de piramidă arbitrar și `glm::perspective()`; - se poate decupa un trunchi de piramidă care provine dintr-o piramidă având baza un dreptunghi, iar piciorul înălțimii piramidei (dusă din vârf) coincide cu centrul bazei. În cazul funcției `glm::perspective()`; sunt considerați ca parametri `fov=field of view`, un unghi între plane și `aspect=raportul` dintre lungimile laturilor dreptunghiului decupat.
- ▶ De testat pe codul sursă `07_01_desenare_cub.cpp` cum este realizată proiecția dacă modificăm diverși parametri ai funcției `glm::frustum()`. De urmărit: (i) cum este realizată decuparea; (ii) cum arată obiectele randate. De exemplu: ce se întâmplă dacă modificăm `dnear`? Aceleași întrebări pentru `glm::perspective()`.

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

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

Concluzie - vizualizare și proiecție

- Este necesară trecerea de la scena 3D (obiectele care se doresc a fi reprezentate), teoretic infinită, la imaginea 2D (primitivele care sunt randate), inclusă într-un dreptunghi cu dimensiuni date. Pentru a realiza acest lucru, sunt urmați doi pași. Acești pași corespund unor transformări specifice:

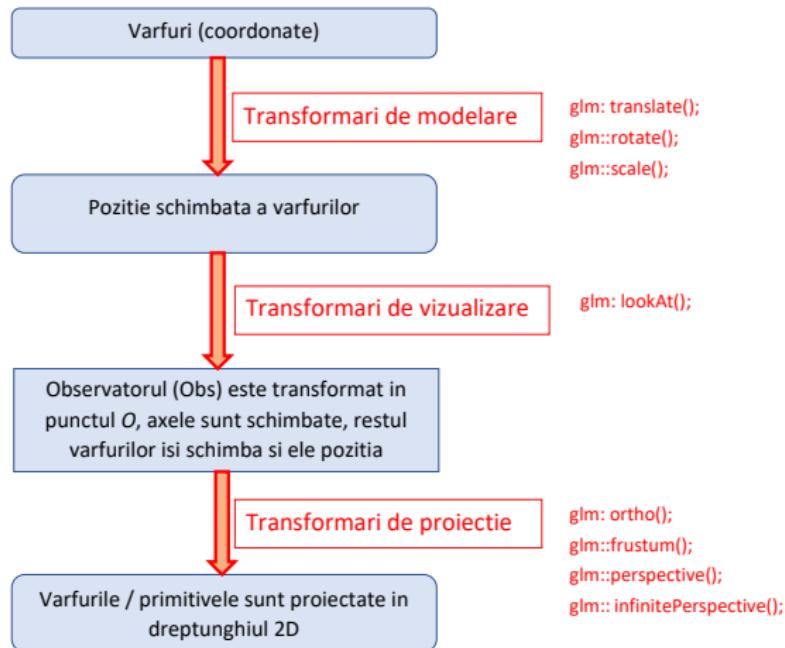
Concluzie - vizualizare și proiecție

- ▶ Este necesară trecerea de la scena 3D (obiectele care se doresc a fi reprezentate), teoretic infinită, la imaginea 2D (primitivele care sunt randate), inclusă într-un dreptunghi cu dimensiuni date. Pentru a realiza acest lucru, sunt urmăți doi pași. Acești pași corespund unor transformări specifice:
 - ▶ Este stabilit modul în care este vizualizată scena 3D (poziția observatorului), prin introducerea coordonatelor de vizualizare și schimbarea reperului. Funcția specifică din GLM este `glm::lookAt();`.

Concluzie - vizualizare și proiecție

- ▶ Este necesară trecerea de la scena 3D (obiectele care se doresc a fi reprezentate), teoretic infinită, la imaginea 2D (primitivele care sunt randate), inclusă într-un dreptunghi cu dimensiuni date. Pentru a realiza acest lucru, sunt urmăți doi pași. Acești pași corespund unor transformări specifice:
 - ▶ Este stabilit modul în care este vizualizată scena 3D (poziția observatorului), prin introducerea coordonatelor de vizualizare și schimbarea reperului. Funcția specifică din GLM este `glm::lookAt();`.
 - ▶ Este stabilit modul în care se realizează (i) decuparea (infinit → finit); (ii) proiecția (3D → 2D). În GLM sunt mai multe funcții specifice, depinzând de obiectivul urmărit: `glm::ortho`; `glm::frustum()`; `glm::perspective()`, `glm::infinitePerspective()`.

Concluzie - fluxul transformărilor

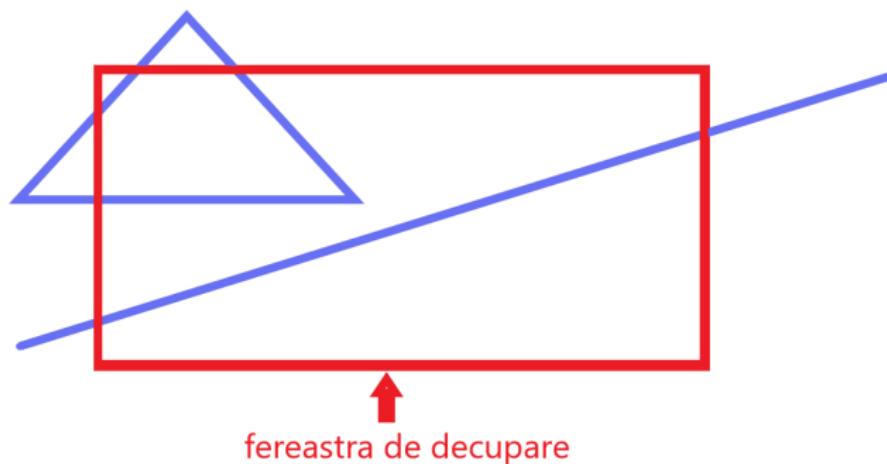


Atenție la ordinea înmulțirii matricelor corespunzătoare din shader:

matrProiectie * matrVizualizare * matrModelare.

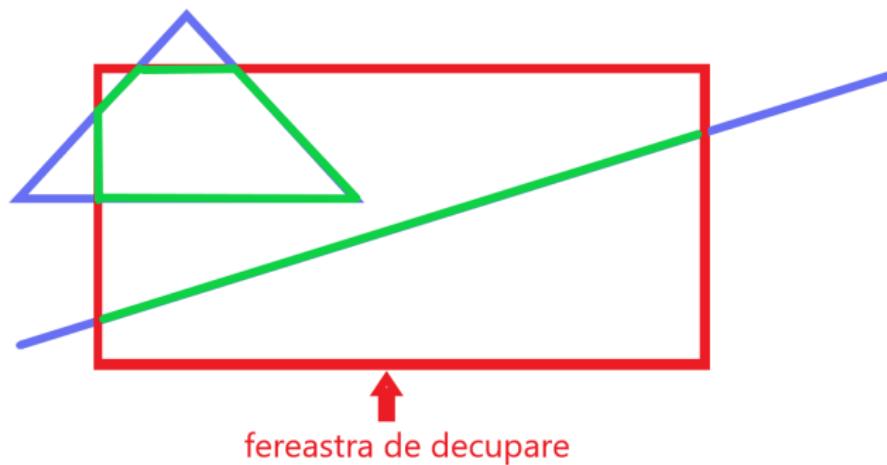
Algoritmi de decupare - motivație

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



Algoritmi de decupare - motivație

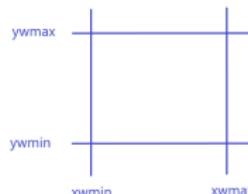
Pentru segment va fi randată doar o porțiune. În cazul triunghiului, va fi decupată o porțiune a sa, fiind randat un pentagon.



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

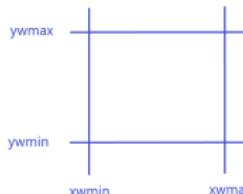
$$x = xwmin, \quad x = xwmax, \quad y = ywmin, \quad y = ywmax.$$



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

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

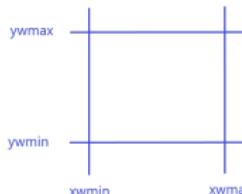


- Algoritmii de decupare depind de tipul primitivei. În cazul punctelor: se bazează pe inegalități.

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

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

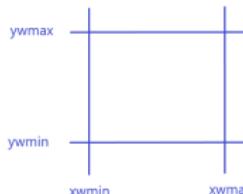


- Algoritmii de decupare depind de tipul primitivei. În cazul punctelor: se bazează pe inegalități.
- În continuare: algoritmi pentru segmente de dreaptă (pornind de la aceștia se poate ajunge la algoritmi pentru poligoane). Sunt descriși doi algoritmi: (i) Cohen-Sutherland (calitativ), (ii) **Liang-Barski**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

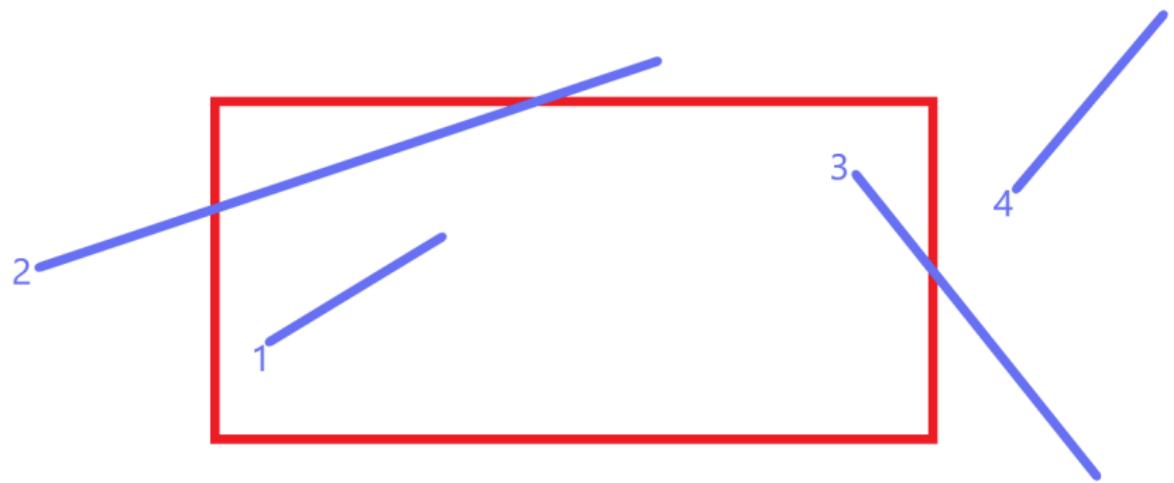
$$x = xwmin, \quad x = xwmax, \quad y = ywmin, \quad y = ywmax.$$



- ▶ 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*

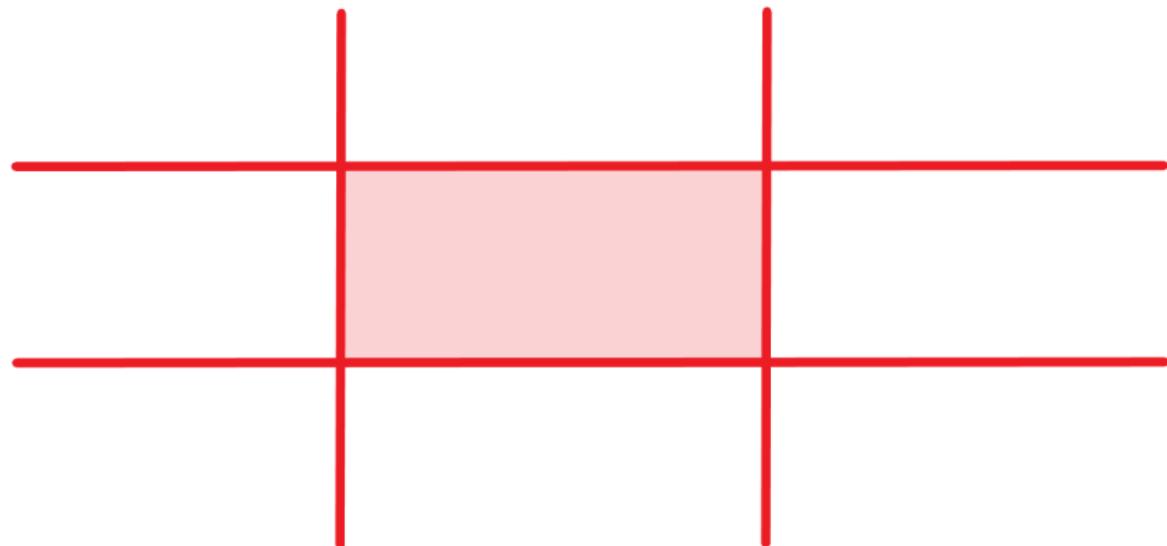
Abordare calitativă. Algoritmul Cohen-Sutherland

Diverse configurații ale segmentelor față de fereastra de decupare.



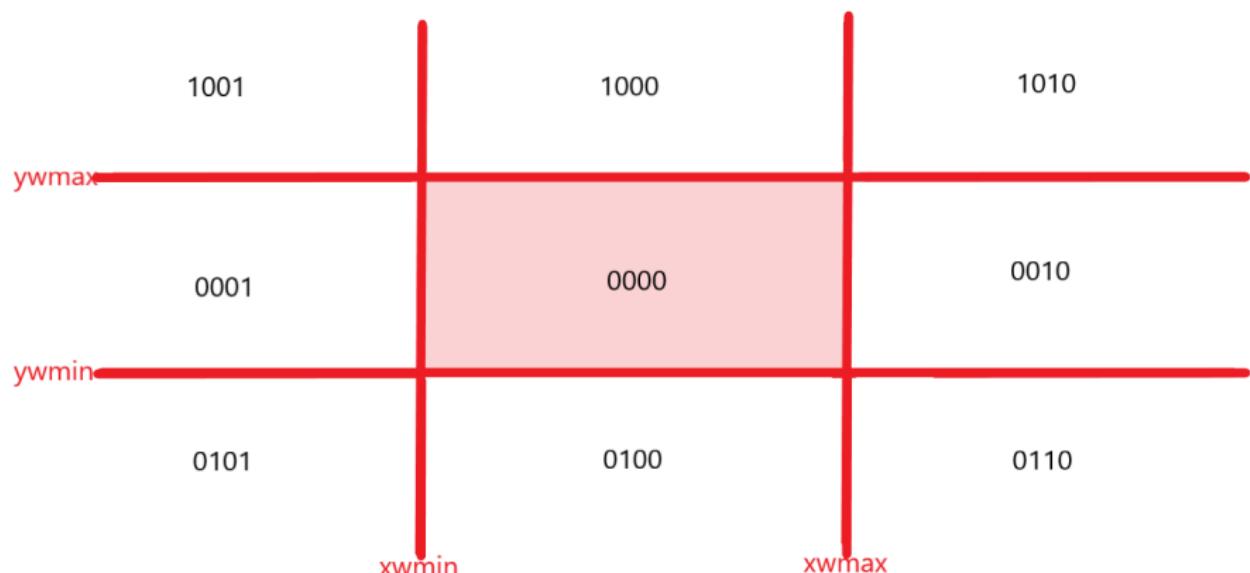
Algoritmul Cohen-Sutherland - împărțirea planului

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



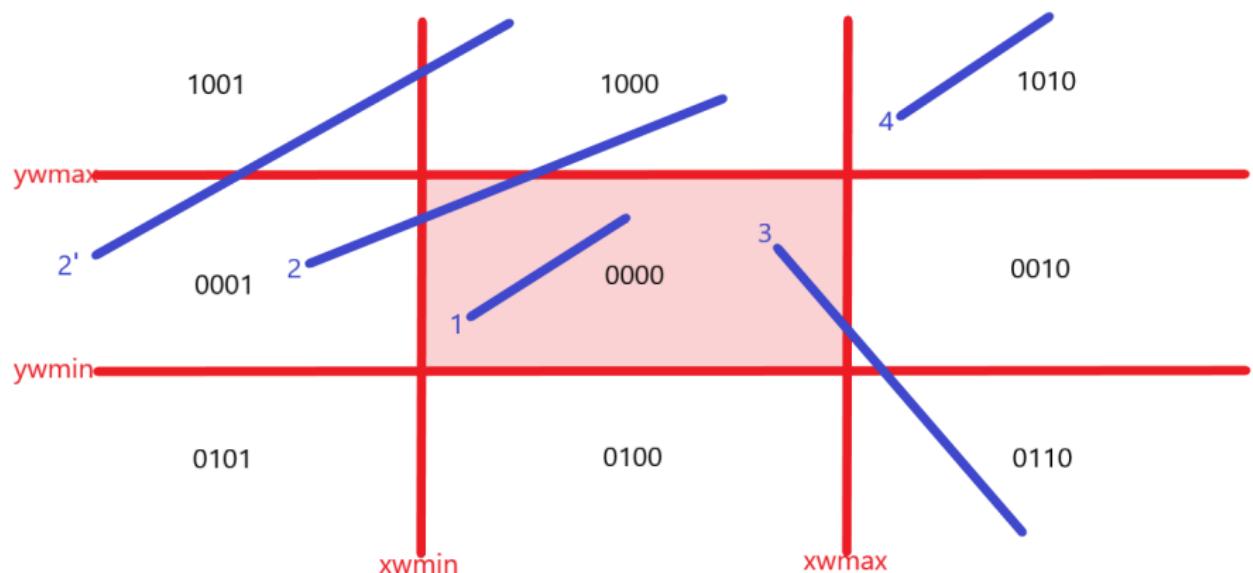
Algoritmul Cohen-Sutherland - codurile regiunilor

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

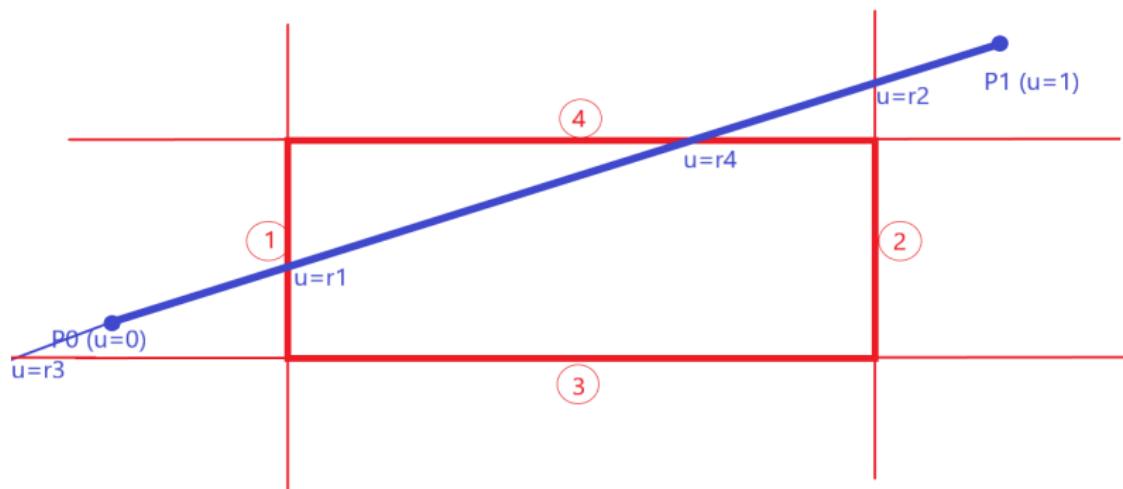
$$\begin{cases} x = x_0 + u \cdot \Delta x \\ y = y_0 + u \cdot \Delta y \end{cases}, u \in \mathbb{R}.$$

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

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



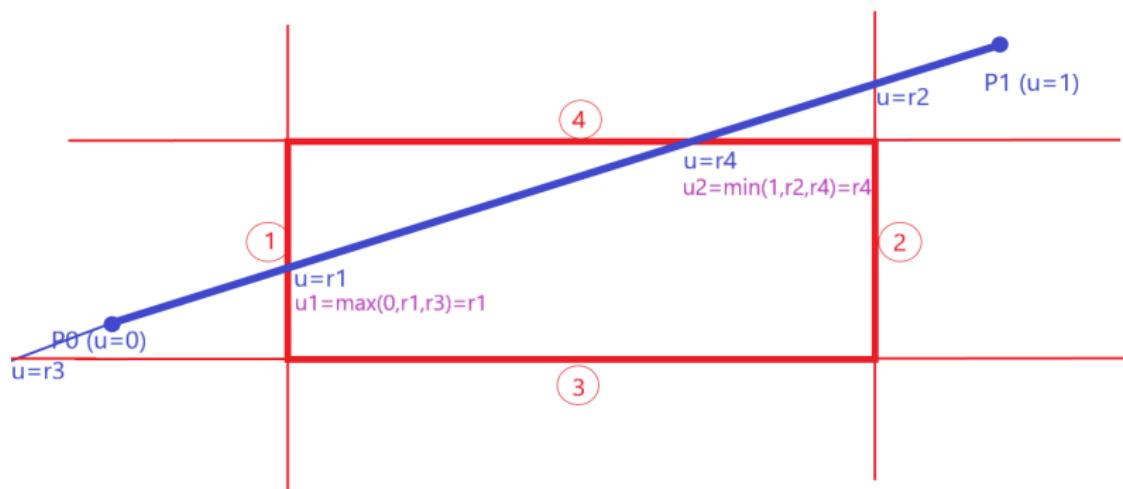
Algoritmul Liang-Barsky - valori importante ale parametrului

"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)



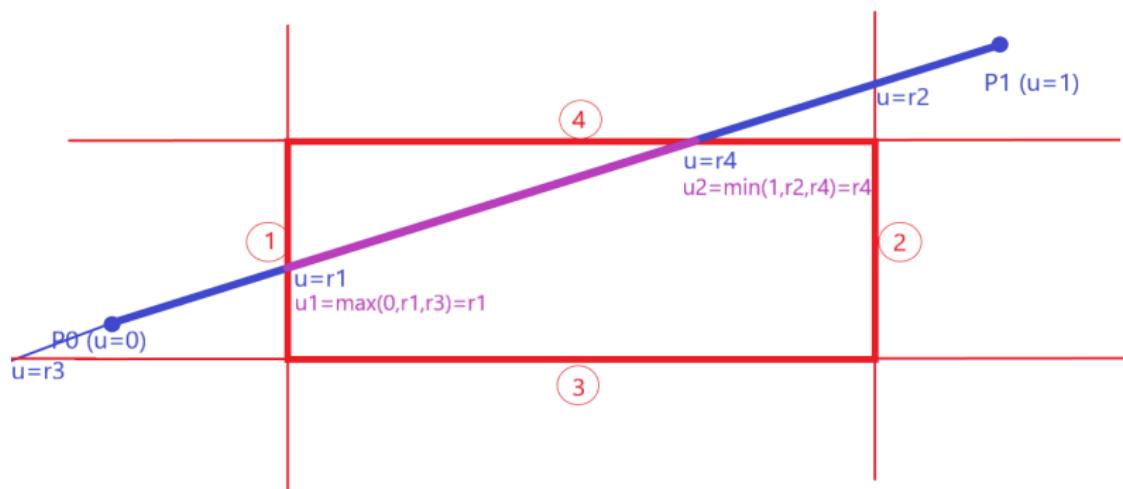
Algoritmul Liang-Barsky - condiția de intersecție

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, 2024 - 2025

Obiecte 3D - scurte repere istorice și resurse

Poliedre

Suprafețe implicate

Curbe parametrizate

Suprafețe 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)

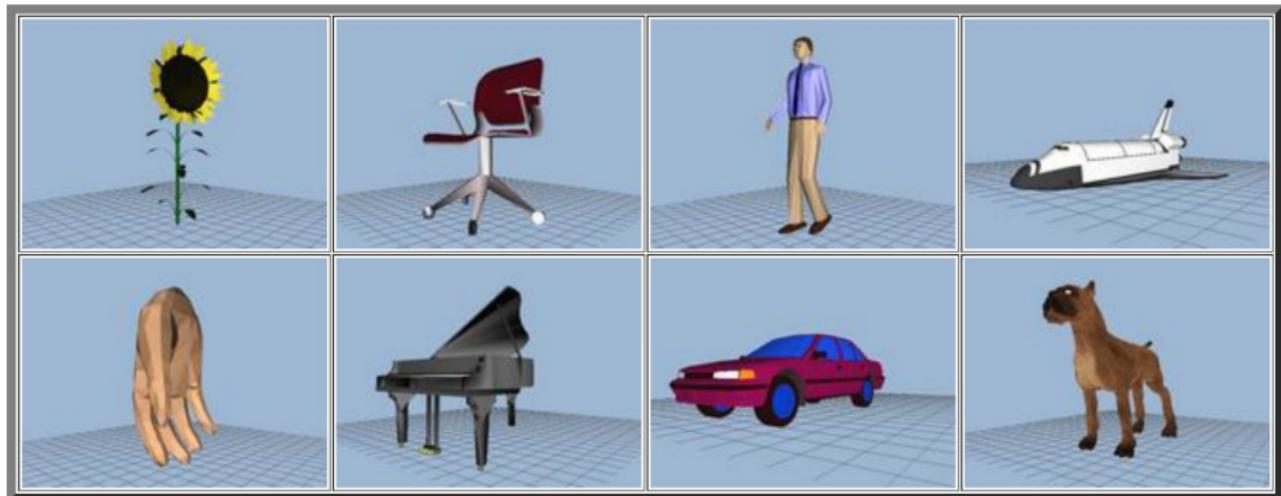
“The Stanford bunny” - 1994.



Sursa: [Stanford scanning repository](#)

Scurt istoric, resurse (iii)

“The Princeton shape benchmark” - ~ 2005



Sursa: *Princeton shape benchmark*

Scurt istoric, resurse (iv)

ModelNet - 2015

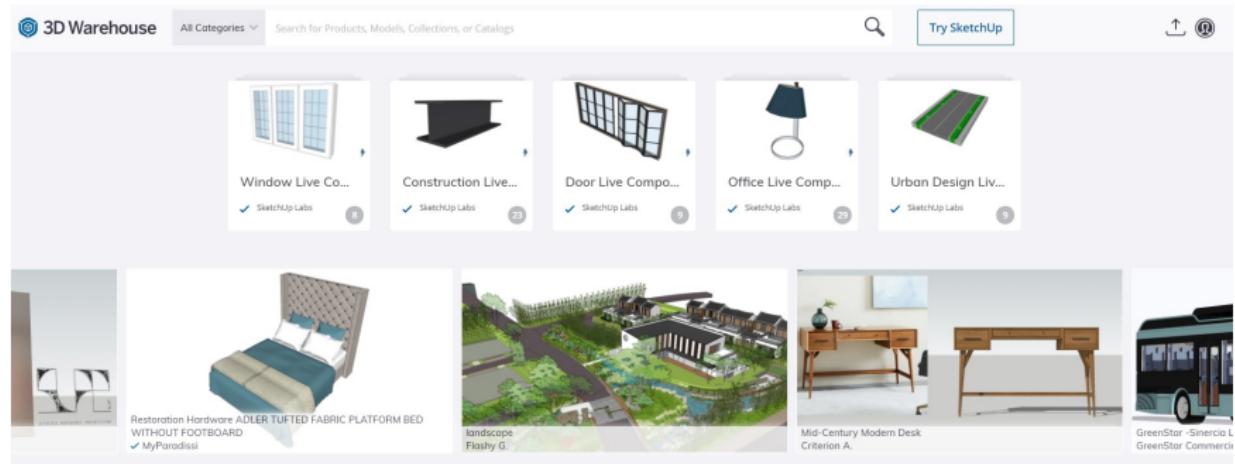


Figure 5: **ModelNet Dataset.** Left: word cloud visualization of the ModelNet dataset based on the number of 3D models in each category. Larger font size indicates more instances in the category. Right: Examples of 3D chair models.

Sursa: [Wu et al., 2015]

Scurt istoric, resurse (v)

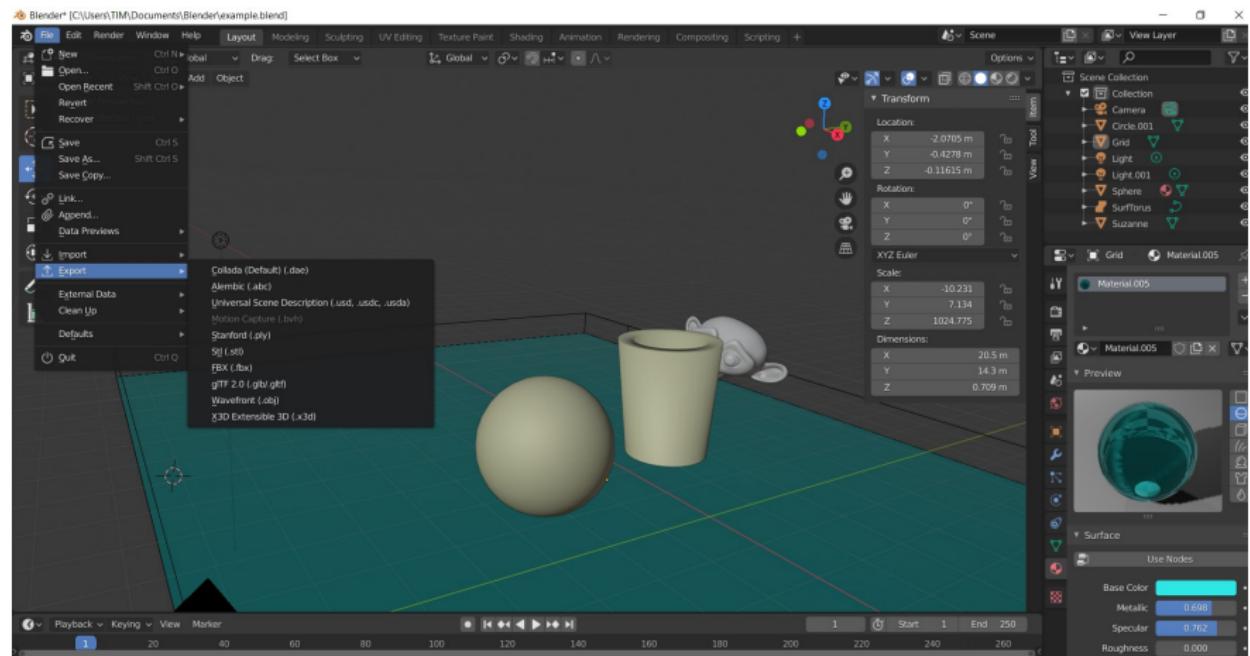
... în prezent există multe resurse disponibile!



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

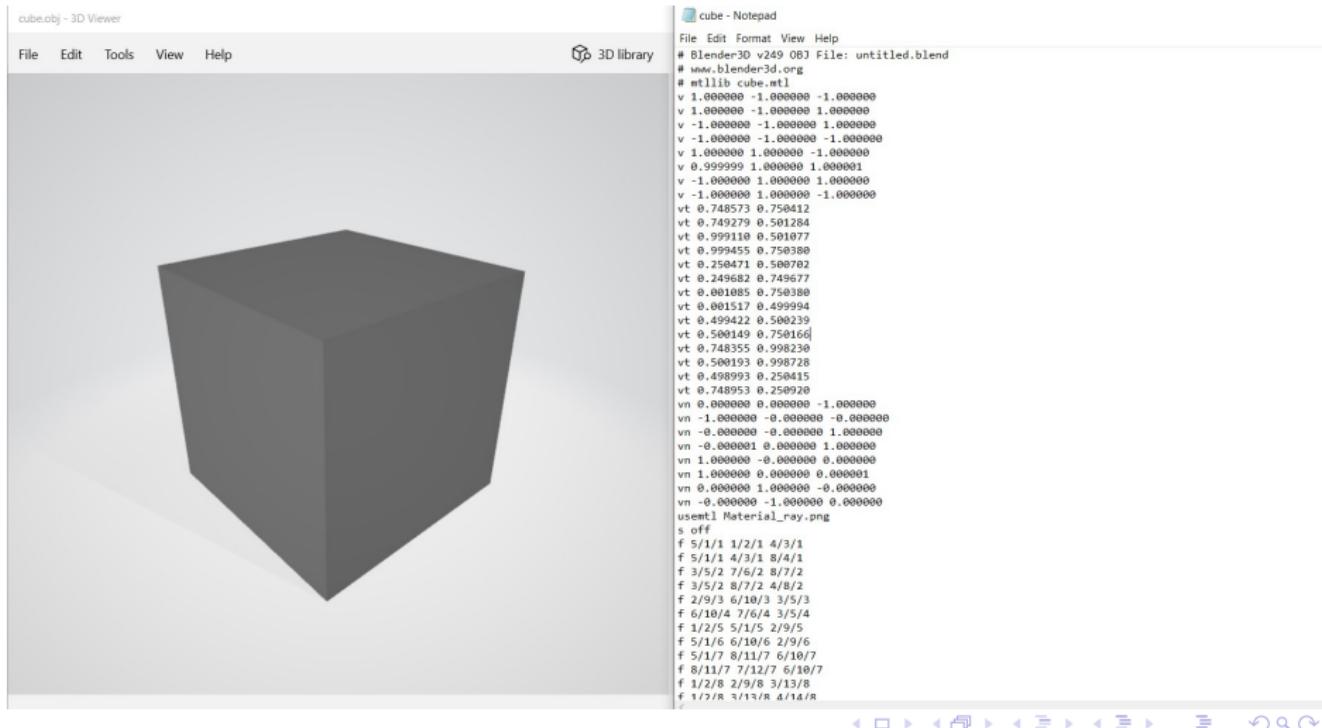
bun_zipper - Notepad

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

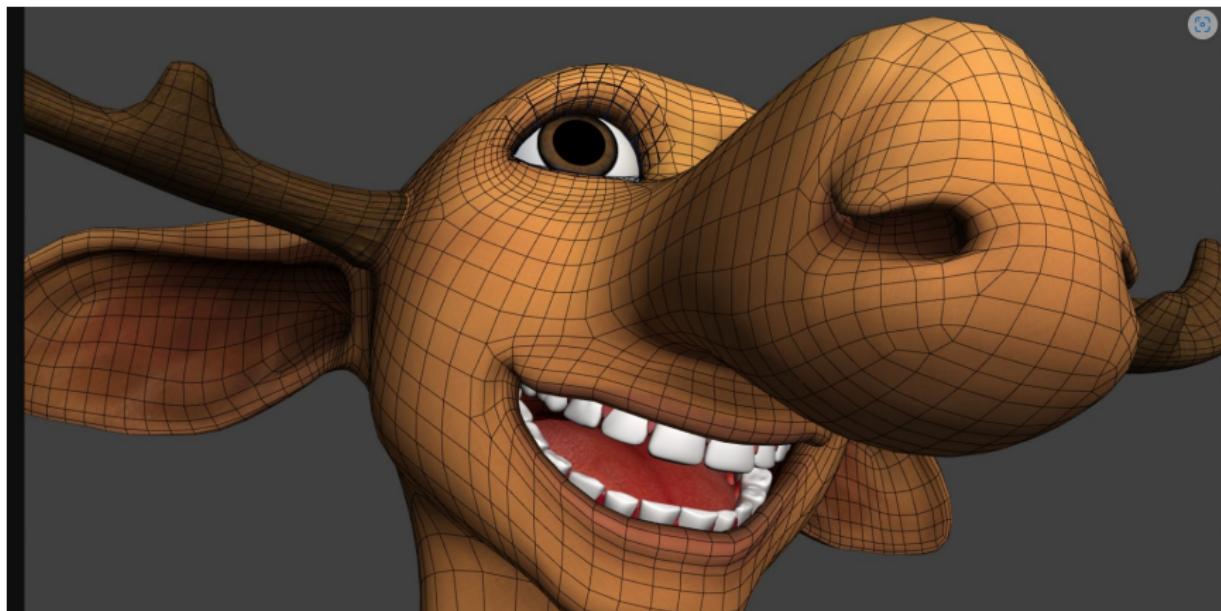
- ▶ Grafica 3D:
 - ce reprezentăm? (aspecte teoretice)
 - cum reprezentăm? (funcționalități OpenGL)
- ▶ Pentru a reprezenta cât mai realist un obiect 3D, pe lângă **coordonate, culori, coordonate de texturare**, un element cheie sunt **vectorii normali** asociați vârfurilor (un atribut al vârfurilor, 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 (v. și [assimp](#)), astfel de modele 3D pot fi utilizate în OpenGL.

1. Poliedre / alte obiecte

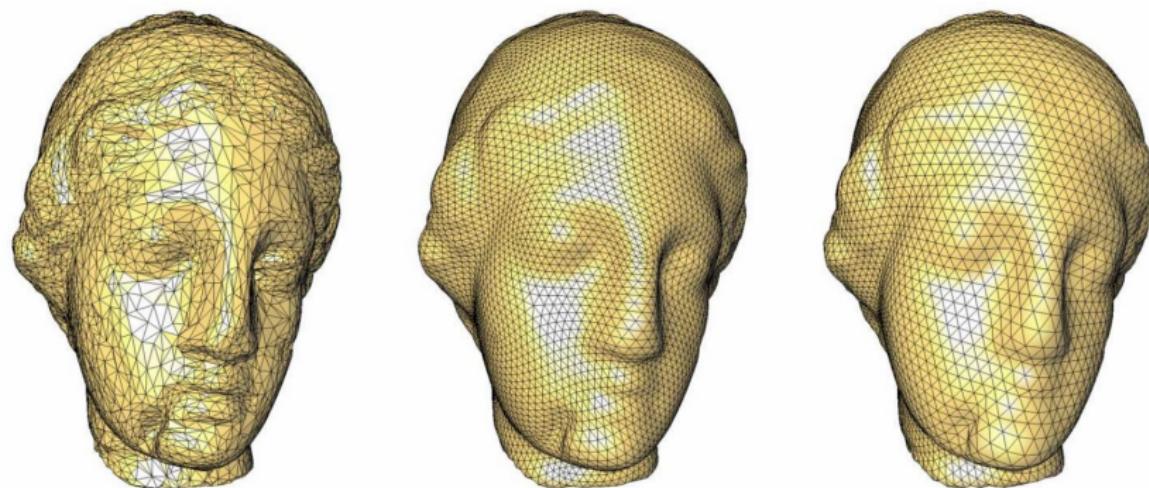
Sunt legate de rețele de poligoane/triunghiuri (*polygon/triangle meshes*)



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*)



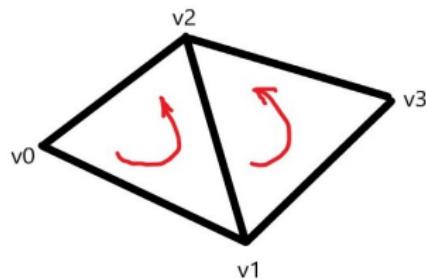
Sursa: <https://hal.inria.fr/file/index/docid/186820/filename/modeling-course.pdf>

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 (*triangle meshes*):

1. Poliedre: vectori normali

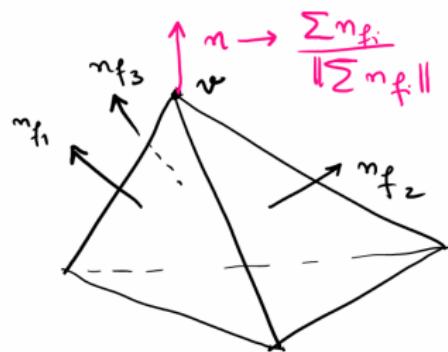
- a) Pentru triunghiuri/poligoane convexe: descriere în cursurile anterioare.
- b) Pentru rețele de triunghiuri (*triangle meshes*):
 - (i) se lucrează la nivel de vârfuri, pentru fiecare vârf se calculează normala (! normala este atribut al vârfurilor), folosind normalele fețelor adiacente.

1. Poliedre: vectori normali

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

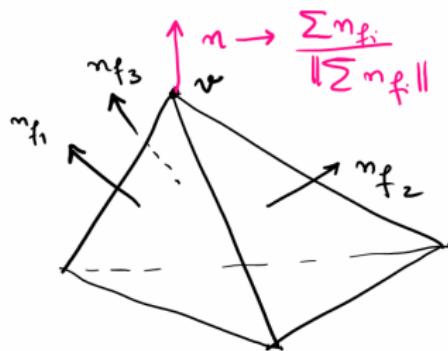
1. Poliedre: vectori normali - detaliere

- 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 $\mathbf{n}_1, \mathbf{n}_2, \dots, \mathbf{n}_q$ (respectiv).

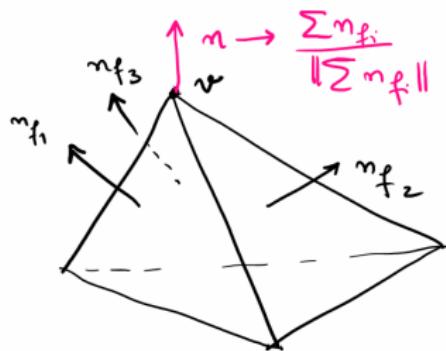


- Vectorul normal în v poate fi definit ca

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

1. Poliedre: vectori normali - detaliere

- Fie v un vârf incident cu fețele f_1, f_2, \dots, f_q , având vectorii normali $\mathbf{n}_1, \mathbf{n}_2, \dots, \mathbf{n}_q$ (respectiv).



- Vectorul normal în v poate fi definit ca

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

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

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

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

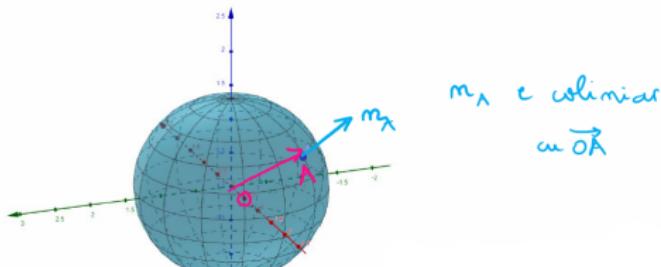
- ▶ **Vectori normali.** Fie (x_0, y_0, z_0) un punct al suprafeței. Normala exterioară la suprafață este dată de vectorul

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

- **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

Exemple.

1) Fie $c : \mathbb{R} \rightarrow \mathbb{R}^2$, $c(t) = (\cos t, \sin t)$, deci

$$\begin{cases} x = \cos t \\ y = \sin t, \quad t \in \mathbb{R}; \text{ } t \text{ parametru} \end{cases}$$

Imaginea curbei c este cercul de centru O și de rază 1.

Din punct de vedere practic: (i) se aleg valori pentru t , (ii) se calculează punctele corespunzătoare, (iii) se unesc aceste puncte și se trasează o aproximare a curbei.

2) În spațiu: $c : \mathbb{R} \rightarrow \mathbb{R}^3$, $c(t) = (\cos t, \sin t, t)$ - elice.

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 08_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

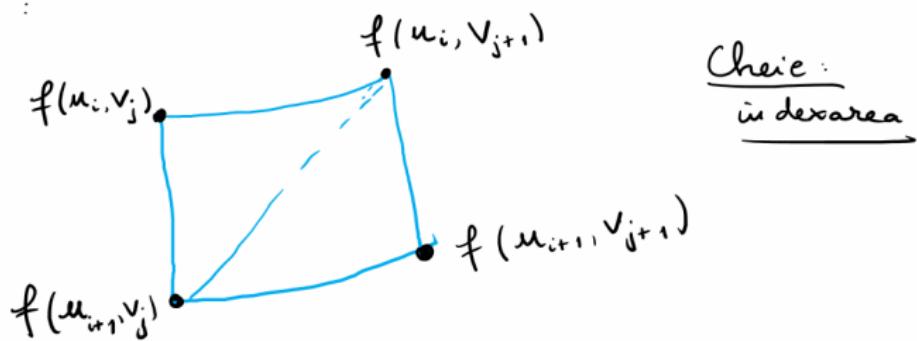
Se generează folosind triunghiuri sau patrulatere și “legând” în mod adecvat vârfurile înceinate.

3. Reprezentări parametrice ale suprafețelor.

Implementare: fețe

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

Principiu :



Cheie :
în dreapta

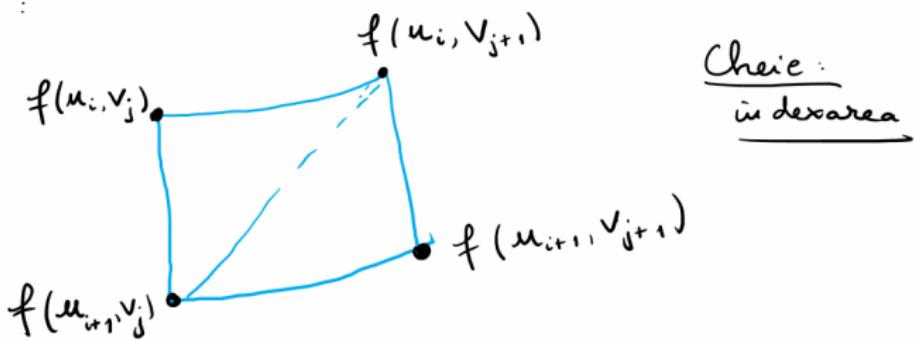
Sunt desenate două triunghiuri (sau un patrulater) care aproximează, local, suprafața respectivă. Se utilizează indexarea vârfurilor.

3. Reprezentări parametrice ale suprafețelor.

Implementare: fețe

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

Principiu :



Sunt desenate două triunghiuri (sau un patrulater) care aproximează, local, suprafața respectivă. Se utilizează indexarea vârfurilor.

Detalii pentru sferă.

3. Reprezentări parametrice ale suprafețelor.

Implementare: vectori normali

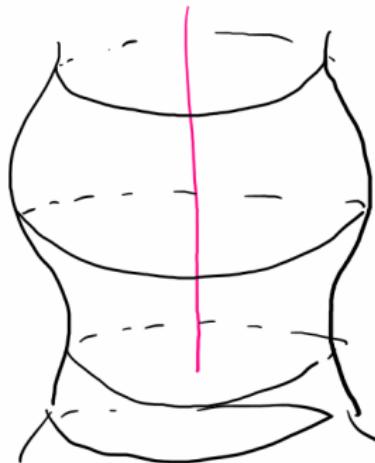
Pentru normale: două variante:

- (i) se aplică principiile pentru rețele de triunghiuri;
- (ii) fixând (u, v) și punctul corespunzător de pe suprafață, un vector s 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:

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

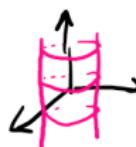
4. Un exemplu: suprafețe de rotație. Illustrare

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

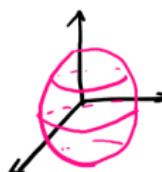
- (i) segment paralel cu axa de rotație \Rightarrow cilindru circular drept (sau o porțiune a acestuia)



- (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

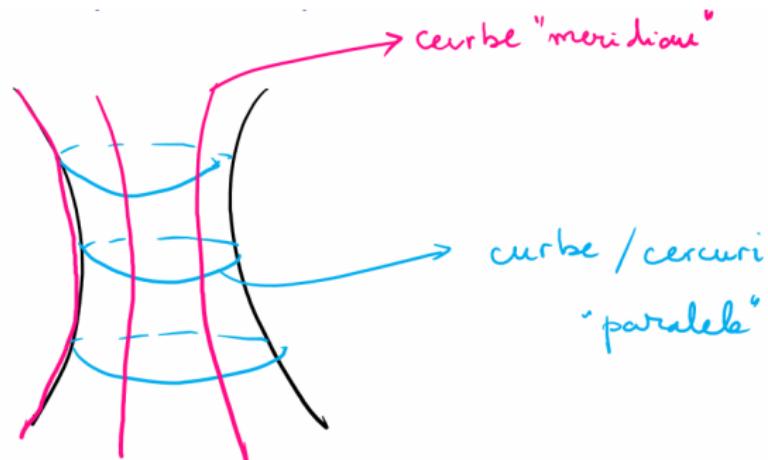
$$f : [0, 2\pi] \times I \rightarrow \mathbb{R}^3,$$

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

(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, 2024 - 2025

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)

Modelele de iluminare mai sunt numite modele de reflexie (*reflection models*) sau modele de umbrărire (*shading models*). Cele mai comune modele de iluminare sunt denumite *ADS models*.

Observație geometrică fundamentală

Context. Fie O un punct și d_0 o semidreaptă cu originea în acel punct. Considerăm o semidreaptă variabilă d cu originea în O ; fie θ unghiul dintre semidreptele d_0 și d . Este utilă o funcție depinzând de θ care să fie descrescătoare pe $[0^\circ, 90^\circ]$. Un exemplu de funcție care verifică această proprietate este funcția **cosinus**.

- (i) Funcția cos este descrescătoare pe $[0^\circ, 90^\circ]$.
- (ii) Valoarea funcției poate fi calculată folosind produsul scalar (*dot product*). Astfel, fie \mathbf{v}_0 și \mathbf{v} vectori directori pentru d_0 , respectiv d . Are loc relația

$$\cos \theta = \frac{\langle \mathbf{v}_0, \mathbf{v} \rangle}{\|\mathbf{v}_0\| \|\mathbf{v}\|} = \frac{\mathbf{v}_0 \cdot \mathbf{v}}{\|\mathbf{v}_0\| \|\mathbf{v}\|}.$$

- (iii) Pe intervalul $[0^\circ, 90^\circ]$ funcția cos ia valori în intervalul $[0, 1]$. Prin ridicare la putere se poate controla modul în care funcția descrește.

Modelul de iluminare

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

termenul legat de emisie
termen ambiental, independent de existenta unor surse de lumina (1)

termenul legat de emisie
termen ambiental, independent de existenta unor surse de lumina (1)

unde N este numărul surselor de lumină.

N termeni, fiecare pl. o sursă de lumină

Ecuatia (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:

$$\begin{aligned}(0.2, 0.4, 0.8) * (0.6, 0.7, 0.5) &= \\ &= (0.12, 0.28, 0.4)\end{aligned}$$

Pentru o sursă de lumină i

attenuation factor i^v · spotlight effect v^i

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



(i) Componenta ambientală

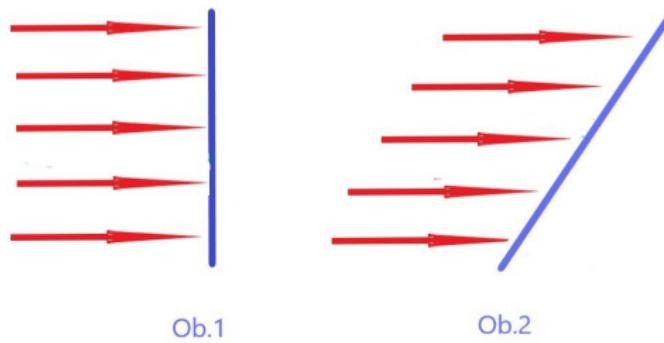
Termenul ambiental corespunzător unei surse de lumină este

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

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)

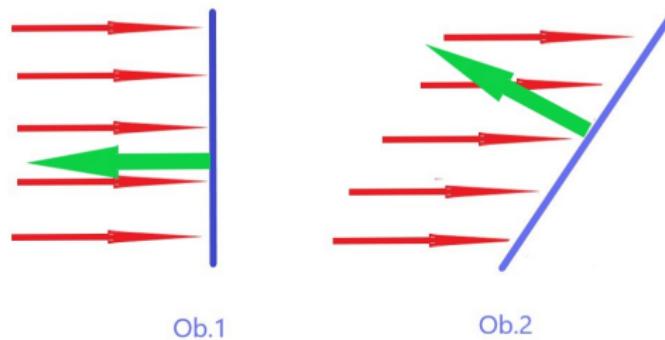
Are legătură cu geometria scenei, lumina reflectată depinde și de incidența luminii asupra obiectelor.



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

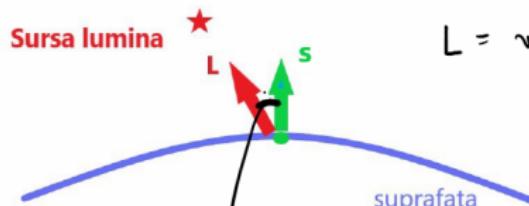
(ii) Reflexia difuză (diffuse term)

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)



s = vector normal la suprafață

L = versor al vectorului

din vîrf către

sursa de lumina

θ = unghiul dintre L și s

Lumina reflectată este legată de $\cos \theta$

$$\cos \theta = \frac{\langle L, s \rangle}{\|L\| \cdot \|s\|} = \langle L, s \rangle = L \cdot s = \text{dot}(L, s)$$

$\underbrace{\qquad\qquad}_{\substack{\|L\| \\ \|s\|}} \qquad \qquad \qquad (\text{produs scalar})$

L, s : vectori de
normă 1

... apoi se realizează în m. cu
diffuse

(ii) Reflexia difuză (diffuse term)

Reflexia difuză pentru o sursă de lumină este descrisă de

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

unde \mathbf{L} este vectorul unitar orientat de la vârf la sursa de lumină (în cazul surselor direcționale este opusul direcției acesteia, normat), iar \mathbf{s} este normala la suprafață în vârful considerat. Cazul $\mathbf{L} \cdot \mathbf{s} < 0$ 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 : poziție în \mathbb{R}^3 . vec3 Light Pos

Light Pos

- vîrf / fragment : vec3 Pos

Pos (Vîrf / Fragment)

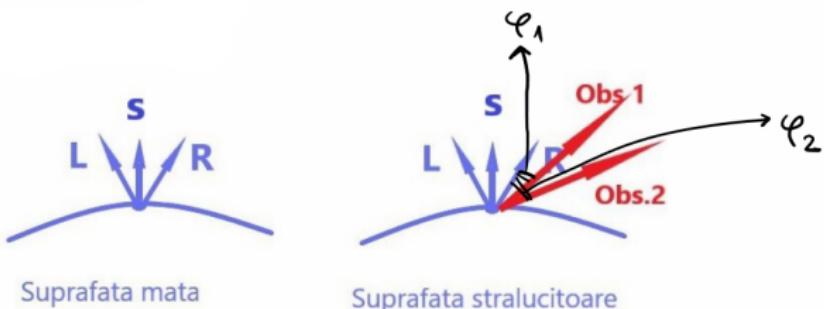
$$L = \frac{\text{Light Pos} - \text{Pos}}{\|\text{Light Pos} - \text{Pos}\|}$$

- sursă directională : direcție vec3 d

$\downarrow \downarrow \downarrow d$ $\uparrow -d$

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

(iii) Reflexia speculară



R = versor pentru direcția de reflexie a luminii ("direcție ideală")

φ = unghiul format de **R** cu versorul spre observator **Obs.**

În desen $\varphi_1 < \varphi_2$, adică Obs1 vede "mai bine" lumina reflectată decât Obs2.

Factorul care descrie atenuarea este $(\cos \varphi)^{\text{shininess}}$, unde shininess este o proprietate de material.

În unele implementări poate fi înlocuit cu unghiul dintre vectorul **H** (*halfway*) și normala **s** la suprafață.

(iii) Reflexia speculară

Astfel, sunt posibile două implementări pentru reflexia speculară:

- [Phong, 1973] Folosind unghiul dintre vesorul spre observator **Obs** și vesorul **R** pentru direcția de reflexie a luminii.

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

- [Blinn, 1977] Folosind unghiul dintre normala la suprafață **s** și vectorul $\mathbf{H} = \frac{\mathbf{L} + \mathbf{Obs}}{\|\mathbf{L} + \mathbf{Obs}\|}$, unde **Obs** este vesorul determinat de vârful considerat și poziția observatorului și **L** este vesor al vectorului din vârf către sursa de lumină.

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

(iv) Coeficientul de atenuare

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

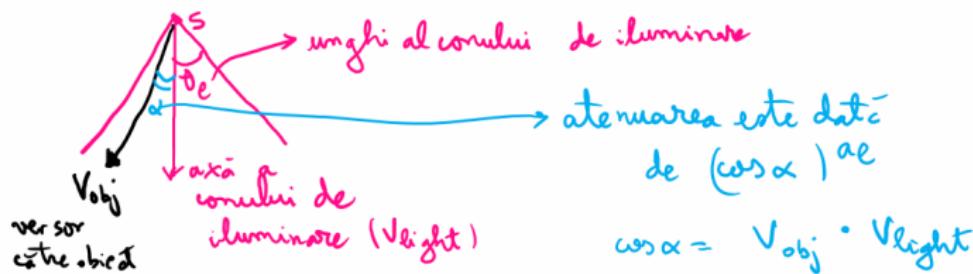
$$\text{attenuation factor} = \frac{1}{a_0 + a_1 d + a_2 d^2},$$

unde d este distanța de la sursa de lumină la vârful/fragmentul considerat ($d=\text{dist}(\text{Pos}, \text{LightPos})$).

(v) Efectul de tip spot

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

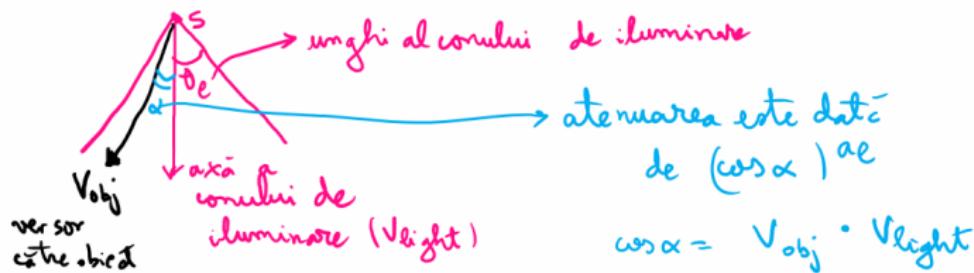
$$\text{spotlight effect} = \begin{cases} 1, & \text{dacă } \theta_l = 180^\circ \\ 0, & \text{dacă } \mathbf{v}_{\text{obj}} \cdot \mathbf{v}_{\text{light}} < \cos \theta_l, \\ (\mathbf{v}_{\text{obj}} \cdot \mathbf{v}_{\text{light}})^{\alpha_l}, & \text{în celelalte cazuri.} \end{cases}$$



(v) Efectul de tip spot

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

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

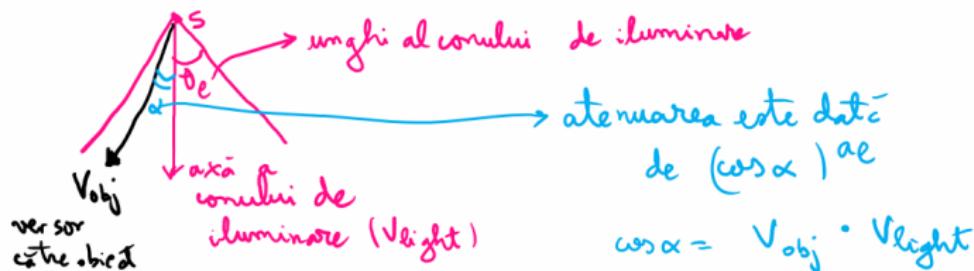


- Elementele definitorii: (i) $\mathbf{v}_{\text{light}}$ este un versor al axei conului de iluminare; (ii) θ_l este unghiul care definește conul de iluminare (deschiderea acestuia); (iii) α_l este coeficientul de atenuare, care caracterizează cum descrește intensitatea luminii prin îndepărțarea de axa conului.

(v) Efectul de tip spot

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

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



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

Coduri sursă

- ▶ 09_01_iluminare.cpp: aplicarea iluminării în cazul unui cub, modelul de iluminare este implementat în shader-ul de fragment.

Coduri sursă

- ▶ 09_01_iluminare.cpp: aplicarea iluminării în cazul unui cub, modelul de iluminare este implementat în shader-ul de fragment.
- ▶ 09_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

- ▶ 09_01_iluminare.cpp: aplicarea iluminării în cazul unui cub, modelul de iluminare este implementat în shader-ul de fragment.
- ▶ 09_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.
- ▶ 09_03_iluminare_sfera.cpp: aplicarea iluminării pentru sferă

Coduri sursă

- ▶ 09_01_iluminare.cpp: aplicarea iluminării în cazul unui cub, modelul de iluminare este implementat în shader-ul de fragment.
- ▶ 09_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.
- ▶ 09_03_iluminare_sfera.cpp: aplicarea iluminării pentru sferă
- ▶ Detalii despre codurile sursă se găsesc în fișierul [info_labs.pdf](#).

Umbre - problematizare: elementele relevante

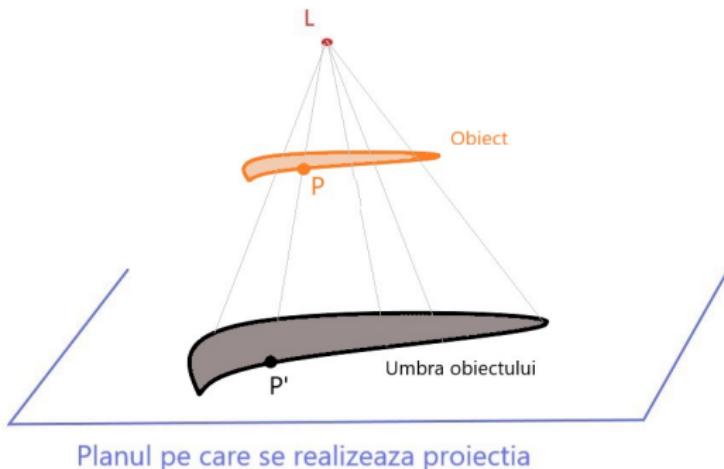
L
•



Planul pe care se realizeaza proiectia

Cadru: L: sursa de lumină (punktuală) : $L = (x_L, y_L, z_L)$
 planul pe care se realizează proiecție : $Ax + By + Cz + D = 0$

Problematizare - elementele relevante



Cadru: L : sursă de lumină (punktuală) : $L = (x_L, y_L, z_L)$
 planul pe care se realizează proiecția : $Ax + By + Cz + D = 0$

Dat P (vârf al obiectului) $\xrightarrow[\text{umbrări}]{\text{trsf.}} P'$ (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

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

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

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

Umbra este realizată pe un plan de ecuație $z + D = 0$
 $(A = B = 0, C = 1)$.

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

Efecte vizuale

Mihai-Sorin Stupariu

Sem. I, 2024 - 2025

Principiul amestecării

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

Principiul amestecării

- ▶ Legat de factorul $A(\text{alpha}, \alpha)$ din codul RGBA. Implicit $A = 1.0$ (obiectele sunt opace). Este specificat în cadrul funcțiilor de culoare / material. Elementele relevante:
 - ▶ **Destinație** (fragment deja procesat)
 - ▶ Dat 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 $A(alpha, \alpha)$ din codul RGBA. Implicit $A = 1.0$ (obiectele sunt opace). Este specificat în cadrul funcțiilor de culoare / material. Elementele relevante:
 - ▶ **Destinație** (fragment deja procesat)
 - ▶ Dat 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** (obiect procesat)
 - ▶ Dat 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

- ▶ Legat de factorul $A(alpha, \alpha)$ din codul RGBA. Implicit $A = 1.0$ (obiectele sunt opace). Este specificat în cadrul funcțiilor de culoare / material. Elementele relevante:
 - ▶ Destinație (fragment deja procesat)
 - ▶ Dat 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 (obiect procesat)
 - ▶ Dat de $S = (R_s, G_s, B_s, A_s)$ (cod RGBA), $F_S = (S_r, S_g, S_b, S_a)$ (factor sursă)
- ▶ Factorul sursă F_S și factorul destinație F_D sunt indicați prin `glBlendFunc(srcfactor, destfactor)`

Principiul amestecării

- ▶ Legat de factorul $A(alpha, \alpha)$ din codul RGBA. Implicit $A = 1.0$ (obiectele sunt opace). Este specificat în cadrul funcțiilor de culoare / material. Elementele relevante:
 - ▶ Destinație (fragment deja procesat)
 - ▶ Dat 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 (obiect procesat)
 - ▶ Dat de $S = (R_s, G_s, B_s, A_s)$ (cod RGBA), $F_S = (S_r, S_g, S_b, S_a)$ (factor sursă)
- ▶ Factorul sursă F_S și factorul destinație F_D sunt indicați prin `glBlendFunc(srcfactor, destfactor)`
- ▶ Factorul destinație (fragmentul deja procesat) și factorul sursă (obiectul care urmează să fie procesat și înregistrat) sunt "amestecate" utilizând o funcție $\varphi(D, F_d, S, F_s)$. Combinarea se realizează după formula

$$\varphi(D, F_d, S, F_s) = F_d * D + F_s * S, \quad (1)$$

urmată de '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 11_01_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 11_01_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

Exemplu: codul sursă 11_01_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 11_01_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 11_01_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 ($destinatie_1$)

$$\begin{aligned} F_S * sursa_1 + F_D * destinatie_1 &= (0.5, 0.5, 0.5, 0.5) * sursa_1 + (0.5, 0.5, 0.5, 0.5) * destinatie_1 = \\ &= (0.5, 0.5, 0.5, 0.5) * (1.0, 1.0, 0.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \end{aligned}$$

Exemplu: codul sursă 11_01_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 11_01_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 ($destinatie_2$):

Exemplu: codul sursă 11_01_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 ($destinatie_1$)

$$\begin{aligned}
 F_S * sursa_1 + F_D * destinatie_1 &= (\mathbf{0.5, 0.5, 0.5, 0.5}) * sursa_1 + (\mathbf{0.5, 0.5, 0.5, 0.5}) * destinatie_1 = \\
 &= (0.5, 0.5, 0.5, 0.5) * (1.0, 1.0, 0.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\
 &= (0.5, 0.5, 0.0, 0.25)
 \end{aligned}$$

Pas 2. Triunghiul turcoaz ($sursa_2$) pe ceea ce s-a calculat mai sus ($destinatie_2$):

$$F_S * sursa_2 + F_D * destinatie_2 = (\mathbf{0.5, 0.5, 0.5, 0.5}) * sursa_2 + (\mathbf{0.5, 0.5, 0.5, 0.5}) * destinatie_2 =$$

Exemplu: codul sursă 11_01_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 ($destinatie_1$)

$$\begin{aligned}
 F_S * sursa_1 + F_D * destinatie_1 &= (\mathbf{0.5, 0.5, 0.5, 0.5}) * sursa_1 + (\mathbf{0.5, 0.5, 0.5, 0.5}) * destinatie_1 = \\
 &= (0.5, 0.5, 0.5, 0.5) * (1.0, 1.0, 0.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\
 &= (0.5, 0.5, 0.0, 0.25)
 \end{aligned}$$

Pas 2. Triunghiul turcoaz ($sursa_2$) pe ceea ce s-a calculat mai sus ($destinatie_2$):

$$\begin{aligned}
 F_S * sursa_2 + F_D * destinatie_2 &= (\mathbf{0.5, 0.5, 0.5, 0.5}) * sursa_2 + (\mathbf{0.5, 0.5, 0.5, 0.5}) * destinatie_2 = \\
 &= (0.5, 0.5, 0.5, 0.5) * (0.0, 1.0, 1.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.5, 0.5, 0.0, 0.25) =
 \end{aligned}$$

Exemplu: codul sursă 11_01_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 ($destinatie_1$)

$$\begin{aligned}
 F_S * sursa_1 + F_D * destinatie_1 &= (\mathbf{0.5, 0.5, 0.5, 0.5}) * sursa_1 + (\mathbf{0.5, 0.5, 0.5, 0.5}) * destinatie_1 = \\
 &= (0.5, 0.5, 0.5, 0.5) * (1.0, 1.0, 0.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\
 &= (0.5, 0.5, 0.0, 0.25)
 \end{aligned}$$

Pas 2. Triunghiul turcoaz ($sursa_2$) pe ceea ce s-a calculat mai sus ($destinatie_2$):

$$\begin{aligned}
 F_S * sursa_2 + F_D * destinatie_2 &= (\mathbf{0.5, 0.5, 0.5, 0.5}) * sursa_2 + (\mathbf{0.5, 0.5, 0.5, 0.5}) * destinatie_2 = \\
 &= (0.5, 0.5, 0.5, 0.5) * (0.0, 1.0, 1.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.5, 0.5, 0.0, 0.25) = \\
 &= (0.0, 0.5, 0.5, 0.25) + (0.25, 0.25, 0, 0.125) =
 \end{aligned}$$

Exemplu: codul sursă 11_01_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 ($destinatie_1$)

$$\begin{aligned}
 F_S * sursa_1 + F_D * destinatie_1 &= (\mathbf{0.5, 0.5, 0.5, 0.5}) * sursa_1 + (\mathbf{0.5, 0.5, 0.5, 0.5}) * destinatie_1 = \\
 &= (0.5, 0.5, 0.5, 0.5) * (1.0, 1.0, 0.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\
 &\quad = (\mathbf{0.5, 0.5, 0.0, 0.25})
 \end{aligned}$$

Pas 2. Triunghiul turcoaz ($sursa_2$) pe ceea ce s-a calculat mai sus ($destinatie_2$):

$$\begin{aligned}
 F_S * sursa_2 + F_D * destinatie_2 &= (\mathbf{0.5, 0.5, 0.5, 0.5}) * sursa_2 + (\mathbf{0.5, 0.5, 0.5, 0.5}) * destinatie_2 = \\
 &= (0.5, 0.5, 0.5, 0.5) * (0.0, 1.0, 1.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.5, 0.5, 0.0, 0.25) = \\
 &\quad = (\mathbf{0.0, 0.5, 0.5, 0.25}) + (\mathbf{0.25, 0.25, 0, 0.125}) = \\
 &\quad \quad \quad \color{red}{(0.25, 0.75, 0.5, 0.375)}
 \end{aligned}$$

Exemplu: codul sursă 11_01_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 11_01_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 11_01_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 11_01_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_DST_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 11_01_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)

$$\begin{aligned} F_S * sursa_1 + F_D * destinatie_1 &= (0.5, 0.5, 0.5, 0.5) * sursa_1 + (0.5, 0.5, 0.5, 0.5) * destinatie_1 = \\ &= (0.5, 0.5, 0.5, 0.5) * (0.0, 1.0, 1.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \end{aligned}$$

Exemplu: codul sursă 11_01_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)

$$\begin{aligned}F_S*sursa_1 + F_D*destinatie_1 &= (0.5, 0.5, 0.5, 0.5)*sursa_1 + (0.5, 0.5, 0.5, 0.5)*destinatie_1 = \\&= (0.5, 0.5, 0.5, 0.5) * (0.0, 1.0, 1.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\&= (0.0, 0.5, 0.5, 0.25)\end{aligned}$$

Exemplu: codul sursă 11_01_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 11_01_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)

$$\begin{aligned}
 F_S * sursa_1 + F_D * destinatie_1 &= (\mathbf{0.5, 0.5, 0.5, 0.5}) * sursa_1 + (\mathbf{0.5, 0.5, 0.5, 0.5}) * destinatie_1 = \\
 &= (0.5, 0.5, 0.5, 0.5) * (0.0, 1.0, 1.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\
 &= (0.0, 0.5, 0.5, 0.25)
 \end{aligned}$$

Pas 2. Triunghiul galben (*sursa*₂) pe ceea ce s-a desenat (*destinatie*₂):

$$F_S * sursa_2 + F_D * destinatie_2 = (\mathbf{0.5, 0.5, 0.5, 0.5}) * sursa_2 + (\mathbf{0.5, 0.5, 0.5, 0.5}) * destinatie_2 =$$

Exemplu: codul sursă 11_01_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)

$$\begin{aligned}
 F_S * sursa_1 + F_D * destinatie_1 &= (\mathbf{0.5, 0.5, 0.5, 0.5}) * sursa_1 + (\mathbf{0.5, 0.5, 0.5, 0.5}) * destinatie_1 = \\
 &= (0.5, 0.5, 0.5, 0.5) * (0.0, 1.0, 1.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\
 &= (0.0, 0.5, 0.5, 0.25)
 \end{aligned}$$

Pas 2. Triunghiul galben (*sursa*₂) pe ceea ce s-a desenat (*destinatie*₂):

$$\begin{aligned}
 F_S * sursa_2 + F_D * destinatie_2 &= (\mathbf{0.5, 0.5, 0.5, 0.5}) * sursa_2 + (\mathbf{0.5, 0.5, 0.5, 0.5}) * destinatie_2 = \\
 &= (0.5, 0.5, 0.5, 0.5) * (1.0, 1.0, 0.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.5, 0.5, 0.25) =
 \end{aligned}$$

Exemplu: codul sursă 11_01_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)

$$\begin{aligned}
 F_S * sursa_1 + F_D * destinatie_1 &= (\mathbf{0.5, 0.5, 0.5, 0.5}) * sursa_1 + (\mathbf{0.5, 0.5, 0.5, 0.5}) * destinatie_1 = \\
 &= (0.5, 0.5, 0.5, 0.5) * (0.0, 1.0, 1.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\
 &= (0.0, 0.5, 0.5, 0.25)
 \end{aligned}$$

Pas 2. Triunghiul galben (*sursa*₂) pe ceea ce s-a desenat (*destinatie*₂):

$$\begin{aligned}
 F_S * sursa_2 + F_D * destinatie_2 &= (\mathbf{0.5, 0.5, 0.5, 0.5}) * sursa_2 + (\mathbf{0.5, 0.5, 0.5, 0.5}) * destinatie_2 = \\
 &= (0.5, 0.5, 0.5, 0.5) * (1.0, 1.0, 0.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.5, 0.5, 0.25) = \\
 &= (0.5, 0.5, 0.0, 0.25) + (0.0, 0.25, 0.25, 0.125) =
 \end{aligned}$$

Exemplu: codul sursă 11_01_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)

$$\begin{aligned}
 F_S * sursa_1 + F_D * destinatie_1 &= (\mathbf{0.5, 0.5, 0.5, 0.5}) * sursa_1 + (\mathbf{0.5, 0.5, 0.5, 0.5}) * destinatie_1 = \\
 &= (0.5, 0.5, 0.5, 0.5) * (0.0, 1.0, 1.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.0, 0.0, 0.0) = \\
 &= (0.0, 0.5, 0.5, 0.25)
 \end{aligned}$$

Pas 2. Triunghiul galben (*sursa*₂) pe ceea ce s-a desenat (*destinatie*₂):

$$\begin{aligned}
 F_S * sursa_2 + F_D * destinatie_2 &= (\mathbf{0.5, 0.5, 0.5, 0.5}) * sursa_2 + (\mathbf{0.5, 0.5, 0.5, 0.5}) * destinatie_2 = \\
 &= (0.5, 0.5, 0.5, 0.5) * (1.0, 1.0, 0.0, 0.5) + (0.5, 0.5, 0.5, 0.5) * (0.0, 0.5, 0.5, 0.25) = \\
 &= (0.5, 0.5, 0.0, 0.25) + (0.0, 0.25, 0.25, 0.125) = \\
 &\quad \color{red} (0.5, 0.75, 0.25, 0.375) \neq (0.25, 0.75, 0.5, 0.375)
 \end{aligned}$$

Funcții de amestecare pentru scenele 3D: codul 11_02_amestecare_3D.cpp

- Combinante:

Funcții de amestecare pentru scenele 3D: codul 11_02_amestecare_3D.cpp

- ▶ Combinări:
 - ▶ ordinea în care sunt desenate obiectele
 - ▶ testul de adâncime
 - ▶ efectele de amestecare

Funcții de amestecare pentru scenele 3D: codul 11_02_amestecare_3D.cpp

- ▶ Combinate:
 - ▶ ordinea în care sunt desenate obiectele
 - ▶ testul de adâncime
 - ▶ efectele de amestecare
- ▶ desenare obiecte opace cu:

Funcții de amestecare pentru scenele 3D: codul 11_02_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)`

Functii de amestecare pentru scenele 3D: codul 11_02_amestecare_3D.cpp

- ▶ Combinare:
 - ▶ 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 11_02_amestecare_3D.cpp

- ▶ Combinare:
 - ▶ 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 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 funcția `mix`.

Efectul de ceață

- **Principiu:** (mecanismul combinațiilor afine) este variată culoarea obiectelor în funcție de distanță, pe baza unei formule de tipul

$$C = f \cdot C_o + (1 - f) \cdot C_f,$$

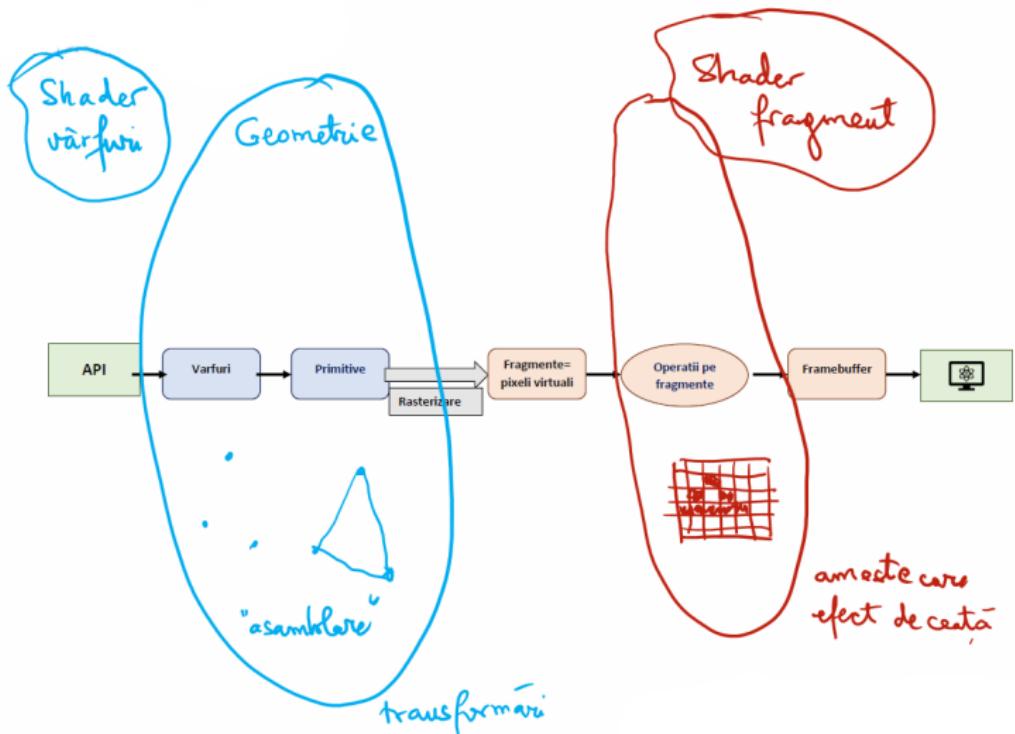
unde: f = factor ceață; C_o = culoarea inițială a obiectului, C_f = culoarea ceții. Pentru implementare: în shader funcția `mix`.

- 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, 2024 - 2025

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

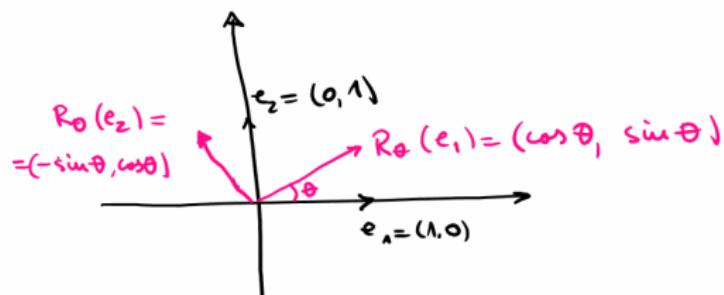
Context 2D, respectiv 3D :

- 2, respectiv 3 numere;
- $(\mathbb{R}^2, +)$, respectiv $(\mathbb{R}^3, +)$

2. Rotații 2D

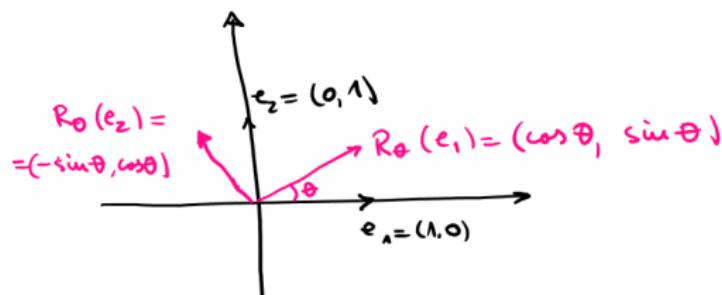
2. Rotații 2D

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

$$R_\theta(e_1) = (\cos \theta, \sin \theta) = \cos \theta \cdot e_1 + \sin \theta \cdot e_2$$

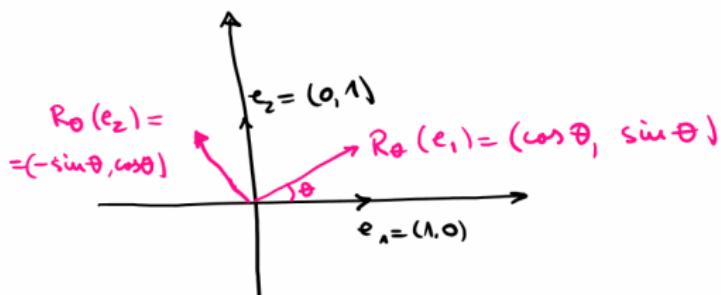
$$R_\theta(e_2) = (-\sin \theta, \cos \theta) = -\sin \theta \cdot e_1 + \cos \theta \cdot e_2$$

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) :

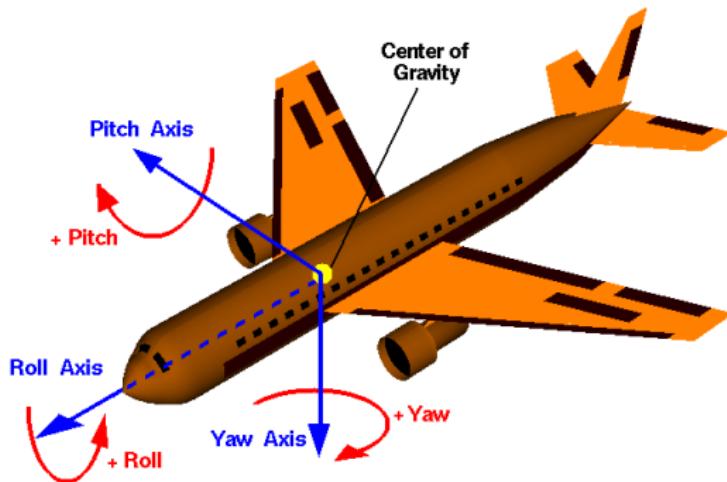
`glm :: rotate(theta, vec3(v1, v2, v3))`

3. Rotații 3D - problematizare: structura grupului $SO(3)$

Două posibilități:

- folosind unghiurile lui Euler
- folosind cuaternioni

3. Rotații 3D - unghiurile lui Euler: intuiție

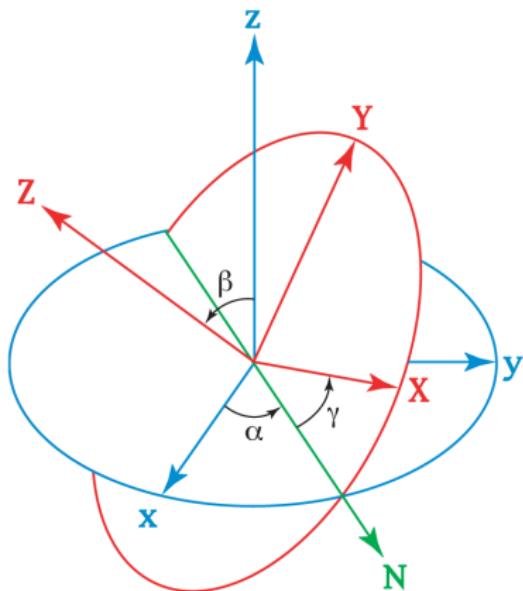


Sursa: <https://upload.wikimedia.org/wikipedia/commons/7/7e/Rollpitchyawplain.png>

Altă reprezentare:

<https://upload.wikimedia.org/wikipedia/commons/8/85/Euler2a.gif>

3. Rotații 3D - unghiurile lui Euler: intuiție



Sursa: <https://upload.wikimedia.org/wikipedia/commons/8/82/Euler.png>

3. Rotații 3D - unghiurile lui Euler: formalizare

Exemplu: Rotația de unghi θ în jurul axei Oy are matricea asociată

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

Altfel spus, considerând axa Oy , elementului $\theta \in S^1$ i se asociază matricea $M_{Oy,\theta} \in \text{SO}(3)$.

3. Rotații 3D - unghiurile lui Euler: formalizare

Exemplu: Rotația de unghi θ în jurul axei Oy are matricea asociată

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

Altfel spus, considerând axa Oy , elementului $\theta \in S^1$ i se asociază matricea $M_{Oy,\theta} \in SO(3)$.

Observație: Există o aplicație

$$S^1 \times S^1 \times S^1 \longrightarrow SO(3)$$

ce se obține utilizând rotațiile în jurul axelor de coordonate.

3. Rotații 3D - unghiurile lui Euler: formalizare

Exemplu: Rotația de unghi θ în jurul axei Oy are matricea asociată

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

Altfel spus, considerând axa Oy , elementului $\theta \in S^1$ i se asociază matricea $M_{Oy,\theta} \in SO(3)$.

Observație: Există o aplicație

$$S^1 \times S^1 \times S^1 \longrightarrow SO(3)$$

ce se obține utilizând rotațiile în jurul axelor de coordonate.

Fapt: Orice matrice din $SO(3)$ poate fi obținută ca produs al unor rotații (3) în jurul axelor de coordonate, cu unghiuri alese convenabil (**unghiurile lui Euler**).

3. Rotații 3D - unghiurile lui Euler: Gimbal Lock

Ilustrare Gimbal Lock

Exemplu: $R(\alpha, (1, 0, 0))$ (rotație de unghi α
împreună cu axa $(1, 0, 0)$)

$$R\left(\frac{\pi}{2}, (0, 1, 0)\right)$$

$$R(\gamma, (0, 0, 1))$$

compoziție (înmulțire matrice...)



matricea

$$\begin{pmatrix} 0 & 0 & 1 \\ \sin(\alpha+\gamma) & \cos(\alpha+\gamma) & 0 \\ -\cos(\alpha+\gamma) & \sin(\alpha+\gamma) & 0 \end{pmatrix}$$

"se pierde
libertate de mișcare"

3. Rotări 3D - reprezentare folosind cuaternioni

$$\mathbb{R}^4 = \{ (s, a, b, c) \mid s, a, b, c \in \mathbb{R} \}$$

???

$$\mathbb{H} = \{ \underbrace{s}_{\text{"partea reală"}} + \underbrace{a i + b j + c k}_{\text{"partea imaginară"}} \mid s, a, b, c \in \mathbb{R} \}$$

Pentru i, j, k se definesc reguli

$$\begin{cases} i^2 = j^2 = k^2 = -1 \\ i \cdot j = -j \cdot i = k \\ j \cdot k = -k \cdot j = i \\ k \cdot i = -i \cdot k = j \end{cases}$$

Pe \mathbb{H} se definesc $\{ "+", "\cdot" \}$ pe componente
induse de regulile de mai sus

Fapt $(\mathbb{H}, +, \cdot)$ corp necomutativ

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 $q = s + ai + bj + ck = (s, \mathbf{v})$
 $\mathbf{v} = (a, b, c)$

cu această notație:

(i) înmulțirea este dată de:

$$q \cdot q' = (s \cdot s' - \mathbf{v} \cdot \mathbf{v}', s\mathbf{v}' + s'\mathbf{v} + \mathbf{v} \times \mathbf{v}')$$

$$(ii) |q|^2 = s^2 + \|\mathbf{v}\|^2$$

$$(iii) \text{Prin } q \neq 0; q^{-1} = \frac{\bar{q}}{|q|^2}, \bar{q} = (s, -\mathbf{v})$$

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 \mathbf{u} și unghiul θ . Fie cuaternionul $q \in S^3$ dat de

$$q = (s, \mathbf{v}), \quad \begin{cases} s = \cos \frac{\theta}{2} \in \mathbb{R} \\ \mathbf{v} = \sin \frac{\theta}{2} \mathbf{u} \in \mathbb{R}^3 \end{cases}$$

Fie $P \in \mathbb{R}^3$ și P' punctul obținut aplicând rotația de unghi θ și axă \mathbf{u} lui P , adică

$$P' = R_{\mathbf{u}, \theta}(P).$$

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. Rotări 3D - reprezentare folosind cuaternioni

Exemplul 1. Considerăm rotarea $R_{\mathbf{u}, \theta}$ dată de vectorul $\mathbf{u} = (0, 0, 1)$ și de unghi $\theta = \frac{\pi}{2} (= 90^\circ)$. Avem $R_{\mathbf{u}, \theta}(1, 0, 0) = (0, 1, 0)$. Interpretarea acestei relații folosind cuaternioni este următoarea.

(i) Vectorul \mathbf{u} corespunde, de fapt, cuaternionului k . Conform propoziției anterioare, rotării $R_{\mathbf{u}, \theta}$ îi se asociază cuaternionul

$$q = \cos \frac{\theta}{2} + \sin \frac{\theta}{2} \mathbf{u} = \cos \frac{\pi}{4} + \sin \frac{\pi}{4} k.$$

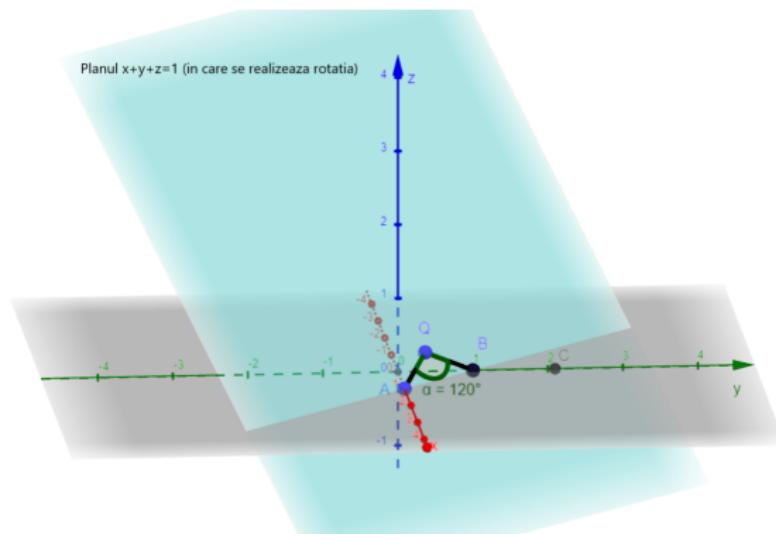
Atunci:

$$q = \frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2} k, \quad q^{-1} = \frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2} k.$$

(ii) Punctele $(1, 0, 0), (0, 1, 0), (0, 0, 1)$ corespund respectiv cuaternionilor i, j, k , deci $R_{\mathbf{u}, \theta}(1, 0, 0) = (0, 1, 0)$ se rescrie $R_{\mathbf{u}, \theta}(i) = j$. Se poate verifica faptul că are loc relația $(0, j) = q \cdot (0, i) \cdot q^{-1}$ (altfel spus $j = q \cdot i \cdot q^{-1}$, sau, echivalent, $j \cdot q = q \cdot i$). Aceasta este exact rescrierea din propoziția anterioară (cu $P = (1, 0, 0) \equiv i, P' = (0, 1, 0) \equiv j$).

3. Rotări 3D - reprezentare folosind cuaternioni

Exemplul 2. Considerăm rotația $R_{\mathbf{u}, \theta}$ dată de **vectorul** $\mathbf{u} = (\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}})$ și de unghi $\theta = \frac{2\pi}{3} (= 120^\circ)$. De exemplu, avem $R_{\mathbf{u}, \theta}(1, 0, 0) = (0, 1, 0)$ (în figură $A = (1, 0, 0), B = (0, 1, 0)$) - rotația de la A la B are loc în planul $x + y + z = 1$ (perpendicular pe axa \mathbf{u} a rotației). Punctul $Q = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ este fix. Practic are o rotație a lui QA către QB , unghiul fiind de 120° .



3. Rotații 3D - reprezentare folosind cuaternioni

Exemplul 2 (continuare). Considerăm rotația $R_{\mathbf{u}, \theta}$ dată de **vectorul $\mathbf{u} = (\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}})$** și **de unghi $\theta = \frac{2\pi}{3}$ ($= 120^\circ$)**. Avem $R_{\mathbf{u}, \theta}(1, 0, 0) = (0, 1, 0)$. Interpretarea acestei relații folosind cuaternioni este următoarea.

(i) Vectorul \mathbf{u} se scrie cu cuaternioni $\mathbf{u} = (\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}) \equiv \frac{1}{\sqrt{3}}i + \frac{1}{\sqrt{3}}j + \frac{1}{\sqrt{3}}k$. Conform propoziției anterioare, rotației $R_{\mathbf{u}, \theta}$ i se asociază cuaternionul

$$q = \cos \frac{\theta}{2} + \sin \frac{\theta}{2} \mathbf{u} = \cos \frac{2\pi}{6} + \sin \frac{2\pi}{6} \left(\frac{1}{\sqrt{3}}i + \frac{1}{\sqrt{3}}j + \frac{1}{\sqrt{3}}k \right) = \dots$$

Prin calcul, se deduce

$$q = \frac{1}{2}(1 + i + j + k), \quad q^{-1} = \frac{1}{2}(1 - i - j - k).$$

(ii) Punctele $(1, 0, 0), (0, 1, 0), (0, 0, 1)$ corespund respectiv cuaternionilor i, j, k , deci $R_{\mathbf{u}, \theta}(1, 0, 0) = (0, 1, 0)$ se rescrie $R_{\mathbf{u}, \theta}(i) = j$. 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>

<https://glm.g-truc.net/0.9.0/api/a00135.html>

Alte tehnici

Mihai-Sorin Stupariu

Sem. I, 2024 - 2025

Bump mapping. Normal mapping

Bump mapping. Normal mapping

Shadow mapping

Curbe și suprafețe Bézier

Problematizare

- ▶ Reprezentarea cât mai eficientă a unor suprafețe cu rugozitate mare.

Problematizare

- ▶ Reprezentarea cât mai eficientă a unor suprafețe cu rugozitate mare.
- ▶ Articol de referință: [Blinn, 1978]
Alte referințe: [Bruneton & Neyret, 2012], [Kautz et al., 2001],
[Heidrich & Seidel, 1999]

Problematizare

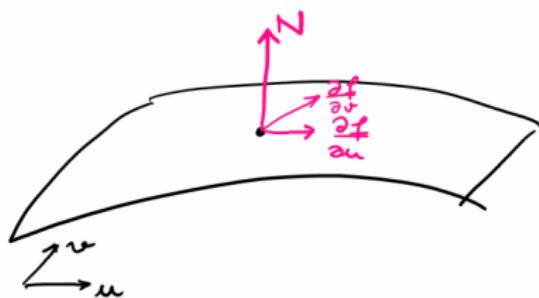
- ▶ Reprezentarea cât mai eficientă a unor suprafețe cu rugozitate mare.
- ▶ Articol de referință: [Blinn, 1978]
Alte referințe: [Bruneton & Neyret, 2012], [Kautz et al., 2001],
[Heidrich & Seidel, 1999]
- ▶ **Principiu: nu este necesar ca suprafețele propriu-zise să aibă o geometrie complicată, este suficient să fie controlat modul în care este reflectată lumina. În particular, este suficient să fie perturbați vectorii normali asociați vârfurilor.**

Problematizare

- ▶ Reprezentarea cât mai eficientă a unor suprafețe cu rugozitate mare.
- ▶ Articol de referință: [Blinn, 1978]
Alte referințe: [Bruneton & Neyret, 2012], [Kautz et al., 2001],
[Heidrich & Seidel, 1999]
- ▶ **Principiu: nu este necesar ca suprafețele propriu-zise să aibă o geometrie complicată, este suficient să fie controlat modul în care este reflectată lumina. În particular, este suficient să fie perturbați vectorii normali asociați vârfurilor.**
- ▶ Cum controlăm / implementăm modificarea vectorilor normali (teoretic / implementare)? Variante: (i) (*procedural*) *bump mapping*, (ii) *normal mapping*.

Bump mapping. Context

Fie $U \subset \mathbb{R}^2$ și $f : U \rightarrow \mathbb{R}^3$, $(u, v) \mapsto f(u, v)$ o suprafață parametrizată.



Conform teoriei, vectorul normal \mathbf{n} la suprafață într-un punct $f(u_0, v_0)$ se calculează

$$\mathbf{n} = \frac{\mathbf{N}}{\|\mathbf{N}\|}, \quad \mathbf{N} = \frac{\partial f}{\partial u}(u_0, v_0) \times \frac{\partial f}{\partial v}(u_0, v_0), \text{ pp. } \mathbf{N} \neq 0.$$

Vectorii $\frac{\partial f}{\partial u}(u_0, v_0)$ și $\frac{\partial f}{\partial v}(u_0, v_0)$ generează planul tangent la suprafață în punctul $f(u_0, v_0)$.

Bump mapping. Ideea de lucru

- 1. Introducerea unei funcții de perturbare.** Se consideră o funcție $\varphi : U \rightarrow \mathbb{R}$ care realizează o “distorsionare” cu “valori mici” ($\varphi \simeq 0$) în direcția normalei (în fiecare punct):

$$\tilde{f} = f + \varphi \mathbf{n}.$$

Bump mapping. Ideea de lucru

- Introducerea unei funcții de perturbare.** Se consideră o funcție $\varphi : U \rightarrow \mathbb{R}$ care realizează o "distorsionare" cu "valori mici" ($\varphi \simeq 0$) în direcția normalei (în fiecare punct):

$$\tilde{f} = f + \varphi \mathbf{n}.$$

- Estimarea vectorilor tangenți după distorsionare.**

$$\frac{\partial \tilde{f}}{\partial u} = \frac{\partial f}{\partial u} + \frac{\partial \varphi}{\partial u} \mathbf{n} + \varphi \frac{\partial \mathbf{n}}{\partial u} \underset{\varphi \simeq 0}{\approx} \frac{\partial f}{\partial u} + \frac{\partial \varphi}{\partial u} \mathbf{n}.$$

$$\frac{\partial \tilde{f}}{\partial v} = \frac{\partial f}{\partial v} + \frac{\partial \varphi}{\partial v} \mathbf{n} + \varphi \frac{\partial \mathbf{n}}{\partial v} \underset{\varphi \simeq 0}{\approx} \frac{\partial f}{\partial v} + \frac{\partial \varphi}{\partial v} \mathbf{n}.$$

Bump mapping. Ideea de lucru

- Introducerea unei funcții de perturbare.** Se consideră o funcție $\varphi : U \rightarrow \mathbb{R}$ care realizează o "distorsionare" cu "valori mici" ($\varphi \simeq 0$) în direcția normalei (în fiecare punct):

$$\tilde{f} = f + \varphi \mathbf{n}.$$

- Estimarea vectorilor tangenți după distorsionare.**

$$\frac{\partial \tilde{f}}{\partial u} = \frac{\partial f}{\partial u} + \frac{\partial \varphi}{\partial u} \mathbf{n} + \varphi \frac{\partial \mathbf{n}}{\partial u} \underset{\varphi \simeq 0}{\approx} \frac{\partial f}{\partial u} + \frac{\partial \varphi}{\partial u} \mathbf{n}.$$

$$\frac{\partial \tilde{f}}{\partial v} = \frac{\partial f}{\partial v} + \frac{\partial \varphi}{\partial v} \mathbf{n} + \varphi \frac{\partial \mathbf{n}}{\partial v} \underset{\varphi \simeq 0}{\approx} \frac{\partial f}{\partial v} + \frac{\partial \varphi}{\partial v} \mathbf{n}.$$

- Estimarea vectorului normal după distorsionare.**

$$\tilde{N} = \frac{\partial \tilde{f}}{\partial u} \times \frac{\partial \tilde{f}}{\partial v} \simeq \left(\frac{\partial f}{\partial u} + \frac{\partial \varphi}{\partial u} \mathbf{n} \right) \times \left(\frac{\partial f}{\partial v} + \frac{\partial \varphi}{\partial v} \mathbf{n} \right) =$$

$$= \frac{\partial f}{\partial u} \times \frac{\partial f}{\partial v} + \frac{\partial f}{\partial u} \times \frac{\partial \varphi}{\partial v} \mathbf{n} + \frac{\partial \varphi}{\partial u} \mathbf{n} \times \frac{\partial f}{\partial v} + \frac{\partial \varphi}{\partial u} \mathbf{n} \times \frac{\partial \varphi}{\partial v} \mathbf{n} = N + D.$$

Vectorul $D = \frac{\partial f}{\partial u} \times \frac{\partial \varphi}{\partial v} \mathbf{n} + \frac{\partial \varphi}{\partial u} \mathbf{n} \times \frac{\partial f}{\partial v}$ s.n. *displacement vector*.

De la teorie la implementare. *Normal mapping*

- ▶ **Inițial:** f (obiectul) și φ (*bump function*) definite pe același domeniu ([Blinn, 1978])

De la teorie la implementare. *Normal mapping*

- ▶ **Inițial:** f (obiectul) și φ (*bump function*) definite pe același domeniu ([Blinn, 1978])
- ▶ **Idee:** “separarea” *bump function* / a normalelor de suprafața pe care este randată

De la teorie la implementare. *Normal mapping*

- ▶ **Inițial:** f (obiectul) și φ (*bump function*) definite pe același domeniu ([Blinn, 1978])
- ▶ **Idee:** “separarea” *bump function* / a normalelor de suprafața pe care este randată
- ▶ **Practic:** cum se rețin normalele? - folosirea texturilor Tutoriale: [learnopengl](#), [opengl-tutorial](#)

Normal mapping. Principii

1. Un vector normal are trei componente, ca o imagine color de tip RGB. Astfel, o colecție de vectori normali pentru o primitivă poate fi stocată într-un fișier de tip imagine (*normal map*), care conține vectorii normali. Practic: pot fi utilizate texturile, transmise către shader folosind variabile de tip *uniform sampler*.

Normal mapping. Principii

1. Un vector normal are trei componente, ca o imagine color de tip RGB. Astfel, o colecție de vectori normali pentru o primitivă poate fi stocată într-un fișier de tip imagine (*normal map*), care conține vectorii normali. Practic: pot fi utilizate texturile, transmise către shader folosind variabile de tip *uniform sampler*.
2. Într-un fișier de tip *normal map* nu sunt reținute normalele propriu-zise, ci deviația acestora față de direcția "teoretică" a normalelor pentru primitiva randată. Pe componente x, y sunt variații față de verticală, iar pe componenta z este valoarea 1. Două consecințe: (i) aspect vizual cu tonuri de albastru, (ii) este necesară raportarea la **geometria primitivei**. Aceasta implică o **schimbare de coordinate**.

Normal mapping. 1. RGB \leftrightarrow Normal

Componentele unui cod RGB (de tip float) au valori în intervalul $[0.0, 1.0]$. Componentele unui vector normal au valori în intervalul $[-1.0, 1.0]$. Este necesară aplicarea unor transformări de corespondență între cele două intervale.

$$RGB = 0.5Normal + 0.5$$

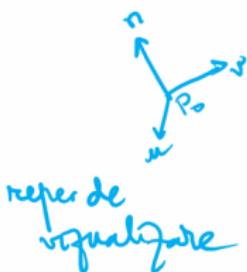
$$Normal_{aux} = 2RGB - 1, \quad Normal = \frac{Normal_{aux}}{\|Normal_{aux}\|}.$$

De exemplu, codul RGB $(0.5, 0.35, 0.3)$ conduce la vectorul $Normal_{aux} = (0.0, -0.3, -0.4)$, iar $Normal = (0.0, -0.6, -0.8)$.

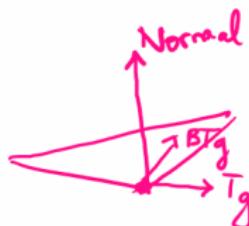
Normal mapping. 2. Schimbare de coordonate

Ideea de bază: normalele utilizate reprezintă "devierea" față de vectorul $(0, 0, 1)$, i.e. planul procesat la momentul respectiv este cel orizontal. Prin urmare, este necesară o schimbare de coordonate (un nou reper), relevante pentru vârfurile considerate.

Spatiul obiectelor



reper de vizualizare

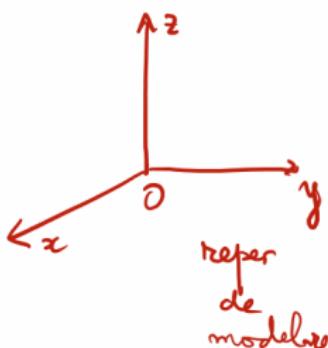


reper ON(T_g , B_tg , Normal)

T_g : tangent

B_tg : bitangent

Normal



reper de modelare

Normal mapping. 2. Schimbare de coordonate

Folosind vectorii tangent, bitangent, normal asociați unui vârf (se presupune că au normă 1 și sunt perpendiculari 2×2) se construiește o matrice ortogonală (matricea TBN, care are acești vectori pe linii), apoi se poate calcula normala “perturbată” pentru un vârf dat. Există diverse moduri în care poate fi utilizată această matrice.

Height mapping

Sunt utilizate *height maps* pentru a perturba pozițiile vârfurilor.

Practic: o imagine (textură) care conține elevații/altitudini (*heights*), de obicei o scară de gri, cu valori între 0 și 1.

Pot fi utile pentru generarea terenurilor.

Exemplu

Principiu și etape

- **Principiu:** Obiectele care nu pot fi văzute de sursa de lumină sunt în umbră. Informația referitoare la obiectele vizibile din poziția sursei de lumină este stocată în buffer-ul de adâncime (*depth buffer*).

Principiu și etape

- ▶ **Principiu:** Obiectele care nu pot fi văzute de sursa de lumină sunt în umbră. Informația referitoare la obiectele vizibile din poziția sursei de lumină este stocată în buffer-ul de adâncime (*depth buffer*).
- ▶ **Etapa 1.** Scena este “desenată” din poziția sursei de lumină. Buffer-ul de adâncime (*depth buffer*) conține, pentru fiecare pixel, distanța dintre sursa de lumină și cel mai apropiat obiect (pe baza *Hidden Surface Removal algorithm*). Informația este stocată într-un buffer dedicat sau într-o textură (*shadow buffer / shadow texture*).

Principiu și etape

- ▶ **Principiu:** Obiectele care nu pot fi văzute de sursa de lumină sunt în umbră. Informația referitoare la obiectele vizibile din poziția sursei de lumină este stocată în buffer-ul de adâncime (*depth buffer*).
- ▶ **Etapa 1.** Scena este “desenată” din poziția sursei de lumină. Buffer-ul de adâncime (*depth buffer*) conține, pentru fiecare pixel, distanța dintre sursa de lumină și cel mai apropiat obiect (pe baza *Hidden Surface Removal algorithm*). Informația este stocată într-un buffer dedicat sau într-o textură (*shadow buffer / shadow texture*).
- ▶ **Etapa 2.** Redarea propriu-zisă a scenei. Pentru fiecare pixel p se extrage valoarea z_p din buffer-ul/textura dedicat(ă) umbrei. Dacă pentru obiectul desenat (corespunzător pixelului) distanța până la sursa de lumină este mai mare decât z_p , atunci obiectul respectiv este în umbră și este desenat cu culoarea umbrei (eventual se aplică termenul ambiental al modelului de iluminare).

Shadow mapping - etapa 1

- Camera este mutată în poziția sursei de lumină - este necesară o matrice de vizualizare-proiecție adecvată (ShadowMVP). Poziția observatorului: sursa de lumină, este ales un punct de referință adecvat.

Shadow mapping - etapa 1

- ▶ Camera este mutată în poziția sursei de lumină - este necesară o matrice de vizualizare-proiecție adecvată (ShadowMVP). Poziția observatorului: sursa de lumină, este ales un punct de referință adecvat.
- ▶ Se are în vedere copierea buffer-ului de adâncime într-o textură (variante folosind `glCopyTexImage2D()` sau `glFrameBufferTexture()` - în acest din urmă caz nu este necesară copierea buffer-ului într-o textură, existând una atașată). Valorile din buffer-ul de adâncime sunt accesate în funcția `glTexImage2D`, folosind pentru format opțiunea `GL_DEPTH_COMPONENT`.

Shadow mapping - etapa 1

- ▶ Camera este mutată în poziția sursei de lumină - este necesară o matrice de vizualizare-proiecție adecvată (ShadowMVP). Poziția observatorului: sursa de lumină, este ales un punct de referință adecvat.
- ▶ Se are în vedere copierea buffer-ului de adâncime într-o textură (variante folosind `glCopyTexImage2D()` sau `glFrameBufferTexture()` - în acest din urmă caz nu este necesară copierea buffer-ului într-o textură, existând una atașată). Valorile din buffer-ul de adâncime sunt accesate în funcția `glTexImage2D`, folosind pentru format opțiunea `GL_DEPTH_COMPONENT`.
- ▶ Este apelată o funcție de desenare (e.g. `glDrawArrays()`), fiind activat testul de adâncime, dar fiind apelat `glDrawBuffer(GL_NONE)`. În shader-ul de vârfuri: aplicată transformarea de mai sus. În shader-ul de fragment: nimic. După desenare, se revine la varianta implicită pentru buffer-ul de cadru (funcții adecvate, de exemplu `glDrawBuffer(GL_FRONT)`).

Shadow mapping - etapa 2

- ▶ Pentru fiecare vârf procesat trebuie stabilit dacă este în umbră sau nu. În shader-ul de vârfuri: aplicată o transformare folosind o matrice adecvată (ShadowMVP2, obținută trecând de la $[-1, 1]$ la $[0, 1]$), rezultatul fiind transmis în shader-ul de fragment. În acesta din urmă, folosind funcția `textureProj` din `GLSL` se stabilește dacă un pixel este sau nu în umbră.

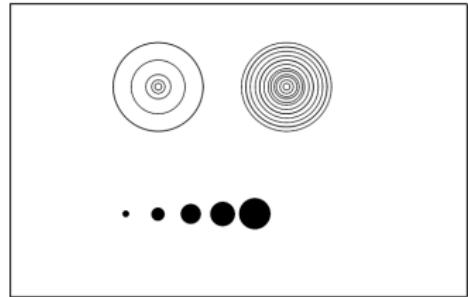
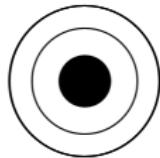
Shadow mapping - etapa 2

- ▶ Pentru fiecare vârf procesat trebuie stabilit dacă este în umbră sau nu. În shader-ul de vârfuri: aplicată o transformare folosind o matrice adecvată (ShadowMVP2, obținută trecând de la $[-1, 1]$ la $[0, 1]$), rezultatul fiind transmis în shader-ul de fragment. În acesta din urmă, folosind funcția `textureProj` din `GLSL` se stabilește dacă un pixel este sau nu în umbră.
- ▶ În programul principal sunt realizati pașii obișnuiți pentru desenare. În plus, este utilizată o variabilă uniformă de tip *sampler*, numită `sampler2DShadow`, care este atașată unei texturi de tip umbră.

Shadow mapping - etapa 2

- ▶ Pentru fiecare vârf procesat trebuie stabilit dacă este în umbră sau nu. În shader-ul de vârfuri: aplicată o transformare folosind o matrice adecvată (ShadowMVP2, obținută trecând de la $[-1, 1]$ la $[0, 1]$), rezultatul fiind transmis în shader-ul de fragment. În acesta din urmă, folosind funcția `textureProj` din `GLSL` se stabilește dacă un pixel este sau nu în umbră.
- ▶ În programul principal sunt realizati pașii obișnuiți pentru desenare. În plus, este utilizată o variabilă uniformă de tip *sampler*, numită `sampler2DShadow`, care este atașată unei texturi de tip umbră.
- ▶ *Shadow mapping* poate produce artefacte, există diverse `îmbunătățiri`

Motivație: cum reprezentăm elementele grafice?



Grafică vectorială și grafică rasterială

Comentarii:

- (i) **Fonturile** sunt de fapt elemente grafice.
- (ii) În proiectare este nevoie de forme cât mai variate, fie la nivel de **schiță**, fie într-un stadiu mai avansat de proiectare.

Scop: Cum generăm elementele de grafică vectorială? (**Curbe Bézier**).

Mecanism

Input: O mulțime de puncte (poligon de control)

Output: Curba reprezentată

Curbe de interpolare:

Exemplu

Curbe Bézier:

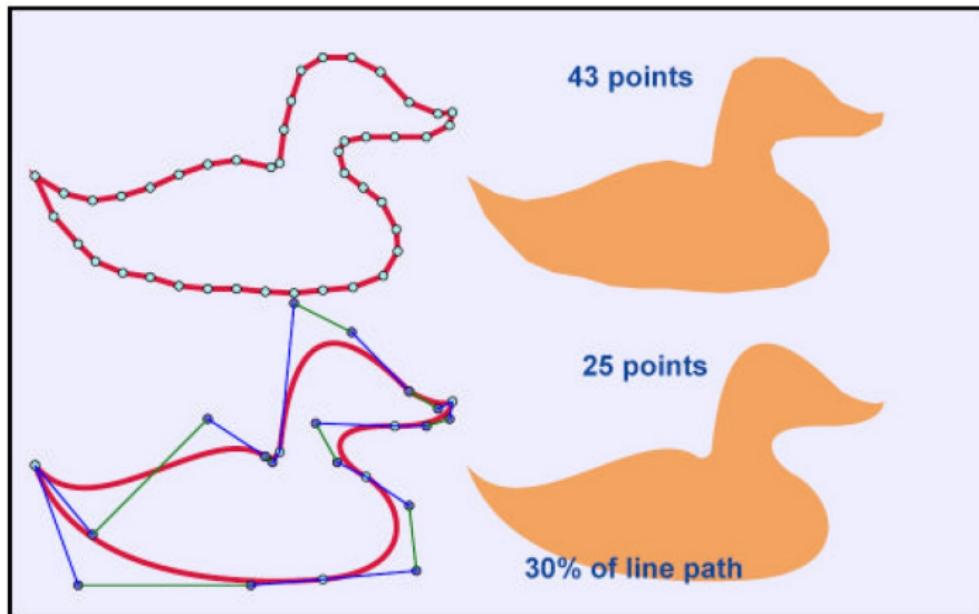
<https://javascript.info/bezier-curve>;

<https://www.jasondavies.com/animated-bezier/>

Mecanism

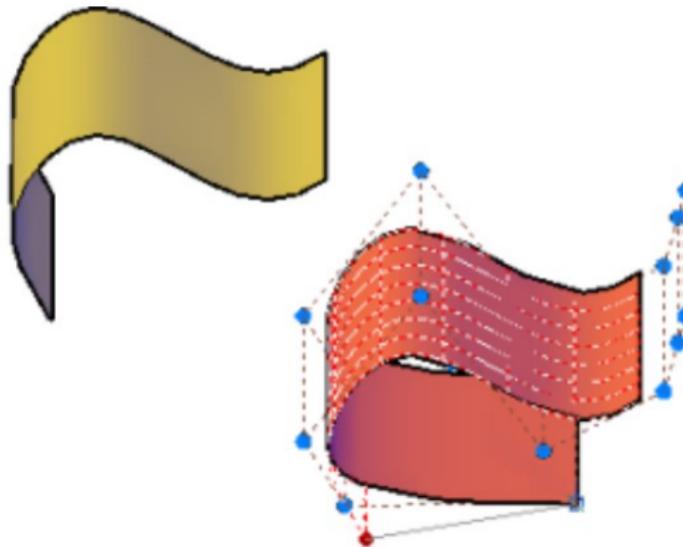
Input: O mulțime de puncte (poligon de control)

Output: Curba reprezentată



Sursa: Duce et al, SVG tutorial

Același principiu funcționează și pentru suprafețe



Sursa: [Knowledge Autodesk](#)

Generalități

- ▶ Dat un poligon de control $(\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_n)$, putem construi o curbă polinomială care să interpoleze aceste puncte.
- ▶ Unele proprietăți ale curbelor de interpolare (de exemplu, faptul că nu sunt incluse în acoperirea convexă a punctelor poligonului de control) fac ca acestea să nu fie practice în aplicații legate de grafica pe calculator.
- ▶ În anii '60, independent unul de celălalt, Paul de Casteljau și Pierre Bézier au investigat curbele asociate poligoanelor de control dintr-o altă perspectivă. Chiar dacă proprietatea de interpolare nu este verificată, sunt alte proprietăți geometrice remarcabile care s-au dovedit a fi foarte utile în inginerie și, ulterior, în CAGD: curbele Bézier (sau, mai precis, reprezentarea Bézier a curbelor polinomiale). La fel ca și curbele de interpolare, curbele Bézier pot fi construite folosind fie metode de natură geometrică (algoritmul de Casteljau), fie utilizând un aparat algebric (forma Bernstein).

Algoritmul de Casteljau pentru cazul $n = 2$

Fie \mathbf{b}_0 , \mathbf{b}_1 și \mathbf{b}_2 trei puncte necoliniare. Pentru $t \in \mathbb{R}$ se construiesc punctele

$$\mathbf{b}_0^1(t) = (1-t)\mathbf{b}_0 + t\mathbf{b}_1,$$

$$\mathbf{b}_1^1(t) = (1-t)\mathbf{b}_1 + t\mathbf{b}_2,$$

$$\mathbf{b}_0^2(t) = (1-t)\mathbf{b}_0^1(t) + t\mathbf{b}_1^1(t).$$

Punctul $\mathbf{b}_0^2(t)$ descrie, când t variază în \mathbb{R} , o parabolă, mai precis parabolă care trece prin punctele \mathbf{b}_0 și \mathbf{b}_2 și ale cărei tangente în aceste puncte sunt dreptele $\mathbf{b}_0\mathbf{b}_1$, respectiv $\mathbf{b}_2\mathbf{b}_1$. Pentru $t \in [0, 1]$ se obține arcul acestei parabole care unește punctele \mathbf{b}_0 și \mathbf{b}_2 .

Punctele intermediare pot fi scrise într-un tablou triunghiular, numit **schemă de Casteljau**. Considerăm, de exemplu, $n = 2$ și fixăm $t_0 \in [0, 1]$. Schema de Casteljau corespunzătoare are forma

$$\begin{array}{ll} \mathbf{b}_0 & \\ \mathbf{b}_1 & \mathbf{b}_0^1(t_0) \\ \mathbf{b}_2 & \mathbf{b}_1^1(t_0) \quad \mathbf{b}_0^2(t_0) \end{array} \quad (1)$$

Exemplu

Considerăm punctele

$$\mathbf{b}_0 = (0, 6), \quad \mathbf{b}_1 = (6, 6), \quad \mathbf{b}_2 = (6, 0).$$

Pentru $t = \frac{1}{3}$ avem

$$\mathbf{b}_0^1\left(\frac{1}{3}\right) = \frac{2}{3}\mathbf{b}_0 + \frac{1}{3}\mathbf{b}_1 = (2, 6),$$

$$\mathbf{b}_1^1\left(\frac{1}{3}\right) = \frac{2}{3}\mathbf{b}_1 + \frac{1}{3}\mathbf{b}_2 = (6, 4),$$

$$\mathbf{b}_0^2\left(\frac{1}{3}\right) = \frac{2}{3}\mathbf{b}_0^1 + \frac{1}{3}\mathbf{b}_1^1 = \left(\frac{10}{3}, \frac{16}{3}\right).$$

Schema de Casteljau asociată este

$$(0, 6)$$

$$(6, 6) \quad (2, 6)$$

$$(6, 0) \quad (6, 4) \quad \left(\frac{10}{3}, \frac{16}{3}\right).$$

Algoritmul de Casteljau, forma generală

Fie $\mathbf{b}_0, \mathbf{b}_1, \dots, \mathbf{b}_n \in \mathbb{R}^m$. Pentru $t \in \mathbb{R}$ se notează $\mathbf{b}_i^0(t) := \mathbf{b}_i$ ($i = 0, \dots, n$) și se definesc punctele

$$\mathbf{b}_i^r(t) := (1-t)\mathbf{b}_i^{r-1}(t) + t\mathbf{b}_{i+1}^{r-1}(t), \quad \begin{cases} r = 1, \dots, n \\ i = 0, \dots, n-r \end{cases} \quad (2)$$

Punctul $\mathbf{b}_0^n(t)$ descrie, când t variază, o curbă, notată cu \mathbf{b}^n . Punctele $\mathbf{b}_0, \mathbf{b}_1, \dots, \mathbf{b}_n$ se numesc **puncte de control** ale curbei \mathbf{b}^n , iar poligonul determinat de acestea se numește **poligon de control**.

Analog cazului $n = 2$, punctele intermediare pot fi scrise într-un tablou triunghiular, numit **schemă de Casteljau**.

Exemplu

Considerăm punctele

$$\mathbf{b}_0 = (1, -2), \quad \mathbf{b}_1 = (3, 2), \quad \mathbf{b}_2 = (3, -2), \quad \mathbf{b}_3 = (-3, -2).$$

Schema de Casteljau corespunzătoare acestor puncte și valorii $t_0 = \frac{1}{2}$ a parametrului este

$$(1, -2)$$

$$(3, 2) \qquad (2, 0)$$

$$(3, -2) \quad (3, 0) \quad \left(\frac{5}{2}, 0\right)$$

$$(-3, -2) \quad (0, -2) \quad \left(\frac{3}{2}, -1\right) \quad \left(2, -\frac{1}{2}\right).$$

Varianta algebrică ($n = 2$)

Date $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$, prin calcul direct se obține

$$\mathbf{b}_0^2(t) = (1-t)^2 \mathbf{b}_0 + 2t(1-t) \mathbf{b}_1 + t^2 \mathbf{b}_2.$$

Polinoamele care apar în această scriere sunt **polinoamele Bernstein de grad 2**

$$B_0^2(t) = (1-t)^2, \quad B_1^2(t) = 2t(1-t), \quad B_2^2(t) = t^2,$$

deci

$$\mathbf{b}_0^2(t) = \sum_{i=0}^2 B_i^2(t) \mathbf{b}_i.$$

Comentarii

- Polinoamele Bernstein de gradul 2 formează o **bază** în spațiul vectorial al polinoamelor de grad mai mic sau egal cu 2, deci **orice** curbă parametrizată polinomial (cu grad ≤ 2) poate fi scrisă folosind polinoame Bernstein.

Comentarii

- Polinoamele Bernstein de gradul 2 formează o **bază** în spațiul vectorial al polinoamelor de grad mai mic sau egal cu 2, deci **orice** curbă parametrizată polinomial (cu grad ≤ 2) poate fi scrisă folosind polinoame Bernstein.
- De fapt: curbele polinomiale de grad mai mic sau egal cu 2 sunt curbele construite folosind algoritmul de Casteljau (sau folosind forma Bernstein) pentru $n = 2$.

Forma algebrică a curbelor Bézier - cazul general

Pentru $n \in \mathbb{N}$ fixat, **polinoamele Bernstein de grad n** , $B_0^n(t), B_1^n(t), \dots, B_n^n(t)$ sunt definite prin

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

unde $C_n^i = \frac{n!}{i!(n-i)!}$. Prin convenție, se poate defini $B_i^n(t) = 0$, dacă $i \notin \{0, \dots, n\}$.

Fapt: Dat un poligon de control $(\mathbf{b}_0, \mathbf{b}_1, \dots, \mathbf{b}_n)$ și $\mathbf{b}_0^n(t)$ punctul contruit cu algoritmul de Casteljau pentru un $t \in \mathbb{R}$, are loc relația

$$\mathbf{b}_0^n(t) = \sum_{k=0}^n B_k^n(t) \mathbf{b}_k.$$

Aceasta este forma Bernstein a curbei Bézier asociate poligonului $(\mathbf{b}_0, \mathbf{b}_1, \dots, \mathbf{b}_n)$.

Proprietăți elementare

Fie $(\mathbf{b}_0, \dots, \mathbf{b}_n)$ un poligon de control din \mathbb{R}^m și $\mathbf{b} : [0, 1] \rightarrow \mathbb{R}^m$ curba Bézier asociată.

Proprietăți elementare

Fie $(\mathbf{b}_0, \dots, \mathbf{b}_n)$ un poligon de control din \mathbb{R}^m și $\mathbf{b} : [0, 1] \rightarrow \mathbb{R}^m$ curba Bézier asociată.

- (i) \mathbf{b} este o curbă polinomială, având gradul mai mic sau egal cu n ;
- (ii) curba \mathbf{b} interpolează extremitățile poligonului de control, i.e. $\mathbf{b}(0) = \mathbf{b}_0$, $\mathbf{b}(1) = \mathbf{b}_n$; în particular, dacă poligonul de control este închis, curba Bézier asociată este închisă;
- (iii) **proprietatea acoperirii convexe:** punctele curbei Bézier \mathbf{b} se află în acoperirea convexă a punctelor de control;

Proprietăți elementare

Fie $(\mathbf{b}_0, \dots, \mathbf{b}_n)$ un poligon de control din \mathbb{R}^m și $\mathbf{b} : [0, 1] \rightarrow \mathbb{R}^m$ curba Bézier asociată.

- (i) \mathbf{b} este o curbă polinomială, având gradul mai mic sau egal cu n ;
- (ii) curba \mathbf{b} interpolează extremitățile poligonului de control, i.e. $\mathbf{b}(0) = \mathbf{b}_0$, $\mathbf{b}(1) = \mathbf{b}_n$; în particular, dacă poligonul de control este închis, curba Bézier asociată este închisă;
- (iii) **proprietatea acoperirii convexe:** punctele curbei Bézier \mathbf{b} se află în acoperirea convexă a punctelor de control;
- (iv) **invarianță afină:** dacă $\tau : \mathbb{R}^m \rightarrow \mathbb{R}^m$ este o aplicație afină, atunci curba Bézier asociată poligonului de control date de $(\tau(\mathbf{b}_0), \dots, \tau(\mathbf{b}_n))$ este curba $\tau(\mathbf{b}^n)$;
- (v) **(Invarianță la combinații baricentrice):** fie $(\mathbf{b}_0, \dots, \mathbf{b}_n)$, respectiv $(\tilde{\mathbf{b}}_0, \dots, \tilde{\mathbf{b}}_n)$ două poligoane de control și \mathbf{b} , respectiv $\tilde{\mathbf{b}}$ curbele Bézier corespunzătoare. Pentru orice $\alpha \in \mathbb{R}$, curba Bézier asociată poligonului de control $((1 - \alpha)\mathbf{b}_0 + \alpha\tilde{\mathbf{b}}_0, \dots, (1 - \alpha)\mathbf{b}_n + \tilde{\mathbf{b}}_n)$ este curba $(1 - \alpha)\mathbf{b} + \alpha\tilde{\mathbf{b}}$.

De reținut!

- ▶ Orice curbă Bézier este definită/controlată de un poligon de control, acesta este "memorat/stocat" și determină geometria curbei.

De reținut!

- ▶ Orice curbă Bézier este definită/controlată de un poligon de control, acesta este "memorat/stocat" și determină geometria curbei.
- ▶ Pentru construcția/randarea curbelor Bézier este folosit algoritmul de Casteljau sau reprezentarea cu polinoame Bernstein.

De reținut!

- ▶ Orice curbă Bézier este definită/controlată de un poligon de control, acesta este "memorat/stocat" și determină geometria curbei.
- ▶ Pentru construcția/randarea curbelor Bézier este folosit algoritmul de Casteljau sau reprezentarea cu polinoame Bernstein.
- ▶ Implementare: folosind *tessellation shader*. [Exemplu](#)

Extinderi

Q: Curbele Bézier sunt, de fapt, **curbe polinomiale**. Cum putem genera curbe cât mai complexe?

Extinderi

Q: Curbele Bézier sunt, de fapt, **curbe polinomiale**. Cum putem genera curbe cât mai complexe?

A:

Extinderi

Q: Curbele Bézier sunt, de fapt, **curbe polinomiale**. Cum putem genera curbe cât mai complexe?

A:

- ▶ Folosind mai multe puncte de control (crește gradul curbei, deci calcule mai complexe).

Extinderi

Q: Curbele Bézier sunt, de fapt, **curbe polinomiale**. Cum putem genera curbe cât mai complexe?

A:

- ▶ Folosind mai multe puncte de control (crește gradul curbei, deci calcule mai complexe).
- ▶ Racordând (“punând cap la cap”) arce de curbă de grad mai mic. Exemplu (curbe de gradul I): graficul funcției modul $c : \mathbb{R} \rightarrow \mathbb{R}$, $c(t) = (t, |t|)$. **În practică: curbe de gradul 3 (cubice).** Se ajunge la **curbe polinomiale pe porțiuni (curbe spline)**.

Extinderi

Q: Curbele Bézier sunt, de fapt, **curbe polinomiale**. Cum putem genera curbe cât mai complexe?

A:

- ▶ Folosind mai multe puncte de control (crește gradul curbei, deci calcule mai complexe).
- ▶ Racordând ("punând cap la cap") arce de curbă de grad mai mic. Exemplu (curbe de gradul I): graficul funcției modul $c : \mathbb{R} \rightarrow \mathbb{R}$, $c(t) = (t, |t|)$. **În practică: curbe de gradul 3 (cubice).** Se ajunge la **curbe polinomiale pe porțiuni (curbe spline)**.
- ▶ Folosind fracții. Exemplu (curbă rațională):
 $c : \mathbb{B} \rightarrow \mathbb{R}^2$, $c(t) = \left(\frac{-t^2+1}{t^2+1}, \frac{2t}{t^2+1} \right)$. Imaginea geometrică a acestei curbe este cercul de centru O și rază 1 din care este eliminat punctul $(-1, 0)$. Se ajunge la **curbe Bézier raționale**.

Extinderi

Q: Curbele Bézier sunt, de fapt, **curbe polinomiale**. Cum putem genera curbe cât mai complexe?

A:

- ▶ Folosind mai multe puncte de control (crește gradul curbei, deci calcule mai complexe).
- ▶ Racordând ("punând cap la cap") arce de curbă de grad mai mic. Exemplu (curbe de gradul I): graficul funcției modul $c : \mathbb{R} \rightarrow \mathbb{R}$, $c(t) = (t, |t|)$. **În practică: curbe de gradul 3 (cubice)**. Se ajunge la **curbe polinomiale pe porțiuni (curbe spline)**.
- ▶ Folosind fracții. Exemplu (curbă rațională):
 $c : \mathbb{B} \rightarrow \mathbb{R}^2$, $c(t) = \left(\frac{-t^2+1}{t^2+1}, \frac{2t}{t^2+1} \right)$. Imaginea geometrică a acestei curbe este cercul de centru O și rază 1 din care este eliminat punctul $(-1, 0)$. Se ajunge la **curbe Bézier raționale**.
- ▶ Combinând cele două idei (spline + raționale) se ajunge la o categorie mai generală, **curbe NURBS - Non Uniform Rational B-Splines**.

Extinderi

Q: Curbele Bézier sunt, de fapt, **curbe polinomiale**. Cum putem genera curbe cât mai complexe?

A:

- ▶ Folosind mai multe puncte de control (crește gradul curbei, deci calcule mai complexe).
- ▶ Racordând ("punând cap la cap") arce de curbă de grad mai mic. Exemplu (curbe de gradul I): graficul funcției modul $c : \mathbb{R} \rightarrow \mathbb{R}$, $c(t) = (t, |t|)$. **În practică: curbe de gradul 3 (cubice)**. Se ajunge la **curbe polinomiale pe porțiuni (curbe spline)**.
- ▶ Folosind fracții. Exemplu (curbă rațională):
 $c : \mathbb{B} \rightarrow \mathbb{R}^2$, $c(t) = \left(\frac{-t^2+1}{t^2+1}, \frac{2t}{t^2+1} \right)$. Imaginea geometrică a acestei curbe este cercul de centru O și rază 1 din care este eliminat punctul $(-1, 0)$. Se ajunge la **curbe Bézier raționale**.
- ▶ Combinând cele două idei (spline + raționale) se ajunge la o categorie mai generală, **curbe NURBS - Non Uniform Rational B-Splines**.
- ▶ Prinципiile de mai sus pot fi extinse în cazul suprafețelor.

Despre testul scris

Mihai-Sorin Stupariu

Sem. I, 2024 - 2025

Informații generale

Modele de probleme

Organizare

- ▶ **Când / unde?** Lucrare scrisă (test): la ultimul curs
seriile 33, 34, 35: miercuri, 15.01.2025, 14:00-15:00, amf. Tițeica
seria 46: miercuri, 29.01.2025, 14:00-15:00, sala 220;

Organizare

- ▶ **Când / unde?** Lucrare scrisă (test): la ultimul curs seriile 33, 34, 35: miercuri, 15.01.2025, 14:00-15:00, amf. Tițeica seria 46: miercuri, 29.01.2025, 14:00-15:00, sala 220;
- ▶ **Este obligatoriu?** Nu. Cine dorește să promoveze doar pe baza punctajului din oficiu + laborator + bonus, va trimite un e-mail (stupariu@fmi.unibuc.ro) în care va preciza explicit acest lucru până **vineri, 10.01.2025** (inclusiv).

Organizare

- ▶ **Când / unde?** Lucrare scrisă (test): la ultimul curs
seriile 33, 34, 35: miercuri, 15.01.2025, 14:00-15:00, amf. Tițeica
seria 46: miercuri, 29.01.2025, 14:00-15:00, sala 220;
- ▶ **Este obligatoriu?** Nu. Cine dorește să promoveze doar pe baza punctajului din oficiu + laborator + bonus, va trimite un e-mail (stupariu@fmi.unibuc.ro) în care va preciza explicit acest lucru până **vineri, 10.01.2025** (inclusiv).

Organizare

- ▶ **Când / unde?** Lucrare scrisă (test): la ultimul curs
seriile 33, 34, 35: miercuri, 15.01.2025, 14:00-15:00, amf. Tițeica
seria 46: miercuri, 29.01.2025, 14:00-15:00, sala 220;
- ▶ **Este obligatoriu?** Nu. Cine dorește să promoveze doar pe baza punctajului din oficiu + laborator + bonus, va trimite un e-mail (stupariu@fmi.unibuc.ro) în care va preciza explicit acest lucru până **vineri, 10.01.2025** (inclusiv).
- ▶ **Formatul?** “Cu cărțile (inclusiv resurse electronice, dar fără ChatGPT sau alte instrumente similare bazate pe AI) pe masă.”

Organizare

- ▶ **Când / unde?** Lucrare scrisă (test): la ultimul curs
seriile 33, 34, 35: miercuri, 15.01.2025, 14:00-15:00, amf. Tițeica
seria 46: miercuri, 29.01.2025, 14:00-15:00, sala 220;
- ▶ **Este obligatoriu?** Nu. Cine dorește să promoveze doar pe baza punctajului din oficiu + laborator + bonus, va trimite un e-mail (stupariu@fmi.unibuc.ro) în care va preciza explicit acest lucru până **vineri, 10.01.2025** (inclusiv).
- ▶ **Formatul?** “Cu cărțile (inclusiv resurse electronice, dar fără ChatGPT sau alte instrumente similare bazate pe AI) pe masă.”
- ▶ **Precizări importante?** **NU COPIAȚI ȘI NU ÎI AJUTAȚI/LĂSATI PE COLEGI/COLEGE SĂ COPIEZE!**

Despre subiecte (generalități)

- Trei grupe de probleme (grilă, cu , cu redactare).

Despre subiecte (generalități)

- Trei grupe de probleme (grilă, cu , cu redactare).
- Tipuri de enunțuri: direct, dați exemple, alegeti valori, fragmente de cod sursă.

Despre subiecte (generalități)

- Trei grupe de probleme (grilă, cu , cu redactare).
- Tipuri de enunțuri: direct, dați exemple, alegeți valori, fragmente de cod sursă.
- Conținuturi: atât referitoare la aspectele teoretice, cât și la partea aplicativă (OpenGL, GLSL).

Despre subiecte (generalități)

- Trei grupe de probleme (grilă, cu , cu redactare).
- Tipuri de enunțuri: direct, dați exemple, alegeți valori, fragmente de cod sursă.
- Conținuturi: atât referitoare la aspectele teoretice, cât și la partea aplicativă (OpenGL, GLSL).
- Detalii și modele de probleme: în continuare.

I. Indicați răspunsul corect. - 10 subiecte a 1 punct

Exemple:

Care dintre codurile RGB de mai jos generează culoarea galben pentru o primitivă grafică?

- a) (1.0, 0.0, 0.0)
- b) (0.0, 1.0, 0.0)
- c) (1.0, 1.0, 0.0)

Se utilizează `glm::lookAt(1,2,4,2,1,4,0,0,1)`. Punctul de referință este:

- a) (1, 2, 4)
- b) (0, 0, 1)
- c) (2, 1, 4)

Se presupune că am generat o textură reprezentând o tablă de șah 8x8 și că aceasta este apelată folosind coordonatele de texturare (0.0, 0.0), (3.0, 0.0), (3.0, 3.0), (0.0, 3.0) și opțiunea GL_REPEAT. Câte pătrățele albe apar? (fondul este negru)

- a) 144
- b) 288
- c) 96

II. Completați răspunsul corect - 10 subiecte a 2 puncte

Exemple:

Dacă se apelează `glDrawArrays(GL_LINES, a, b)` (alegeți $a > 0$, $b > 10$), vor fi desenate segmente.

La apelarea funcției `glm::translatef(5, 6, 7)`, matricea 4×4 generată are suma elementelor egală cu

În funcția `glDrawArrays()` poate fi utilizată constanta simbolică, având ca efect desenarea

Indicați două caracteristici (prezentate la curs) referitoare la fața poligoanelor

Indicați două diferențe dintre sursele de lumină direcționale și cele punctuale

Se consideră un vârf de coordonate $(2, 4, 3)$ cu proprietatea de material $(0.4, 0.0, 0.9)$, normala la suprafață în vârful respectiv este $\mathbf{s} = (0, 0, 1)$ și sursa de lumină, cu `GL_DIFFUSE` dată de $(0.1, 0.2, 0.3)$, este situată în punctul $(2, 4, 7)$. Valoarea termenului difuz (*diffuse term*) este deoarece

III. Rezolvați complet problemele - 2 subiecte a 5 puncte

Exemple:

1. (Slides 07_transformari_partea_3_vizualizare, codul sursă 06_01_poligoane3D.cpp)

Fie punctele

$A_1 = (5, -5, 5)$, $A_2 = (-5, -5, 5)$, $A_3 = (-5, 5, 5)$, $A_4 = (5, 5, 5)$. Stabiliți poziția punctului $O = (0, 0, 0)$ față de poligonul $A_1A_2A_3A_4$.

2. (Slides 07_transformari_partea_3_vizualizare, codul sursă 06_02_poligoane3D_exemplu2.cpp)

Fie punctele $P_1 = (6, 2, 0)$, $P_2 = (-4, 4, 8)$, $P_3 = (0, 0, 8)$ (toate trei situate în planul de ecuație $x + y + z = 8$).

- a) Să se aleagă P_4 astfel ca patrulaterul $P_1P_2P_3P_4$ să fie concav.
- b) Să se aleagă P_5 astfel ca patrulaterul $P_1P_2P_3P_5$ să fie convex.
- c) Să se aleagă puncte O_1 și O_2 astfel ca poligonul $P_1P_2P_3P_5$ să fie văzut din față, respectiv din spate.

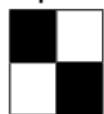
III. Rezolvați complet problemele - 2 subiecte a 5 puncte

Exemple:

3. Sunt indicate vârfurile $(0,0), (2,0), (2,2), (0,2)$. Este apelată secvența
`m1 = glm::scale (0.5, 2.0, 0.0);`
`m2 = glm::translate (20.0, 10.0, 0.0);`

matricea transmisă în shader fiind `m1 * m2`.

- a) Care sunt coordonatele vârfului desenat în dreapta sus?
b) Aplicăm dreptunghiului rezultat în urma transformărilor textura



; coordonatele de texturare asociate vârfurilor sunt $(0,0)$ (stânga jos), $(4,0)$ (dreapta jos), $(4,2)$ (dreapta sus), $(0,2)$ (stânga sus), iar fundalul este roșu. Stabiliți care este raportul dintre aria colorată cu alb și cea colorată cu negru, știind că este utilizată opțiunea `GL_CLAMP`.

III. Rezolvați complet problemele - 2 subiecte a 5 puncte

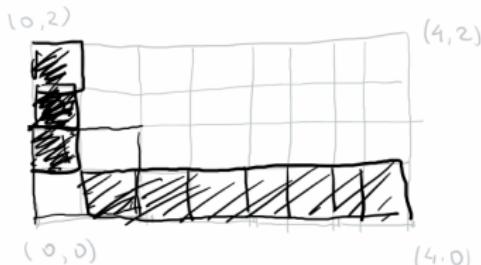
Soluția problemei 3:

3. a) Dreptunghiu original: vf. dreapta sus: (2,2)

$$(2, 2) \xrightarrow{\text{translație}} (22, 12) \xrightarrow{\text{scalare}} (11, 24)$$

(corect în 4 coordonate)

b)



Pe orizontală: de 4 ori
verticale: de 2 ori

In total: $8 \times 4 = 32$
de celule
(pătratele)

Negru: 10
Alb: 22

$$\Rightarrow raportul: \frac{22}{10} = \frac{11}{5} = 2,2$$

III. Rezolvați complet problemele - 2 subiecte a 5 puncte

Exemple:

4. Se aplică funcția `glm::lookAt(3,5,7,1,5,7,0,0,1)`. Este desenat triunghiul determinat de vârfurile $A(0, 3, 7)$, $B(0, 7, 7)$, $C(0, 4, 9)$. Se presupune că se aplică o proiecție ortogonală cu parametri adecvați (adică, după aplicarea acesteia, triunghiul este desenat complet). Să se arate că în randare triunghiul are o latură orizontală și să se stabilească dacă cel de-al treilea vârf este reprezentat deasupra sau dedesubtul acestei laturi.

III. Rezolvați complet problemele - 2 subiecte a 5 puncte

Soluția problemei 4:

4. - "Interpretăm" funcția glmr:: lookAt() :

- observatorul: $P_o = (3, 5, 7)$

- pct. de referință: $P_{ref} = (1, 5, 7)$

- verticala "proprie": $V = (0, 0, 1)$

- Reperul de vizualizare:

$$\begin{aligned} V \times n &= (0, 0, 1) \times (1, 0, 0) : \\ &= \begin{vmatrix} 0 & 1 & e_1 \\ 0 & 0 & e_2 \\ 1 & 0 & e_3 \end{vmatrix} = \\ &= 0 \cdot e_1 - (-1) e_2 + \\ &\quad + 0 \cdot e_3 = (0, 1, 0) \end{aligned}$$

- $N = P_o - P_{ref} = (2, 0, 0) \Rightarrow \underline{n = (1, 0, 0)}$

- orizontală din planul de vizualizare: $u = \frac{V \times n}{\|V\|} = (0, 1, 0)$

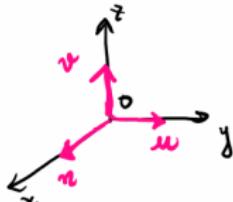
- verticală din planul de vizualizare: $v = n \times u = (0, 0, 1)$

(obs. în acest exemplu v coincide cu V)

III. Rezolvați complet problemele - 2 subiecte a 5 puncte

Soluția problemei 4 (continuare):

-Întelegerea contextului geometric



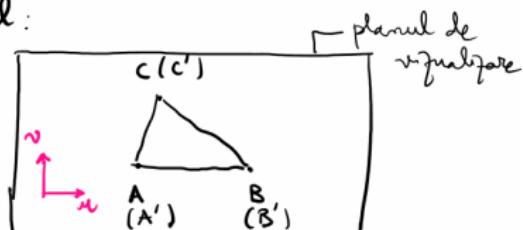
Planul de vizualizare (pe care sunt proiectate obiectele) este generat de $u = (0, 1, 0)$ și $v = (0, 0, 1)$, deci este paralel cu Oyz . Proiecția este orizontală și este realizată paralel cu axa Ox .

Cele trei vârfuri sunt proiectate astfel:

$$A = (0, 3, 7) \mapsto A' \equiv (3, 7)$$

$$B = (0, 7, 7) \mapsto B' \equiv (7, 7)$$

$$C = (0, 4, 9) \mapsto C' \equiv (4, 9)$$



în raportare: $[AB]$ orizontal
C deasupra lui $[AB]$

III. Rezolvați complet problemele - 2 subiecte a 5 puncte

Exemple:

5. În funcția createVBO sunt indicate vârfurile

```
GLfloat Vertices[] =  
{  
    // coordonate           // culori  
    -2.0f, 3.0f, -2.0f, 1.0f, 0.0f, 0.0f, 1.0f,  
    2.0f, 3.0f, -2.0f, 1.0f, 0.0f, 0.0f, 1.0f,  
    2.0f, -2.0f, -8.0f, 1.0f, 0.0f, 0.0f, 1.0f,  
    -2.0f, -2.0f, -8.0f, 1.0f, 0.0f, 0.0f, 1.0f  
};
```

În funcția de desenare se apelează

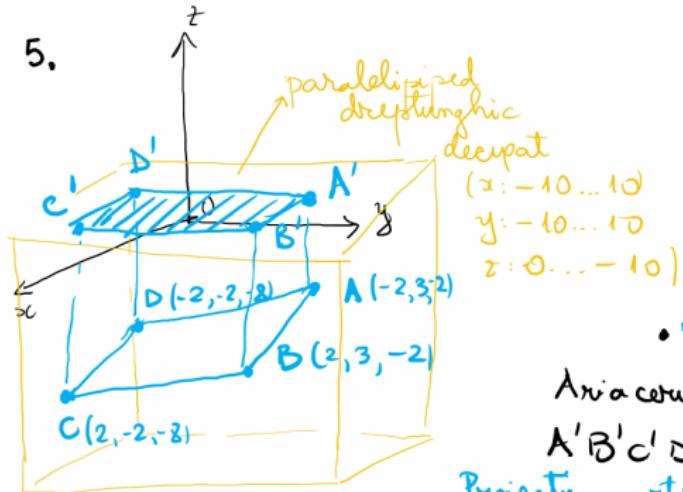
```
glm::ortho(-10,10,-10,10,0,10);  
glDrawArrays(GL_QUADS, 0, 4);
```

Ce arie va avea figura desenată cu albastru (prin raportare la sistemul de coordonate inițial, fără a efectua scalarea la paralelipipedul normalizat)?

III. Rezolvați complet problemele - 2 subiecte a 5 puncte

Soluția problemei 5:

5.



- Cele patru vârfuri sunt situate în interiorul paralelipipedului dreptunghic decupat.

• Vă fi acordat $A'B'C'D'$

Aria cerută este aria dreptunghiului $A'B'C'D'$.

Proiecția ortogonală: de-a lungul axei Oz

$$A = (-2, 3, -2) \mapsto A' \equiv (-2, 3)$$

$$B = (2, 3, -2) \mapsto B' \equiv (2, 3)$$

$$C = (2, -2, 8) \mapsto C' \equiv (2, -2)$$

$$\Rightarrow \text{aria } A'B'C'D' = A'B \cdot B'C = 4 \cdot 5 = 20 \quad (\text{aria cerută})$$

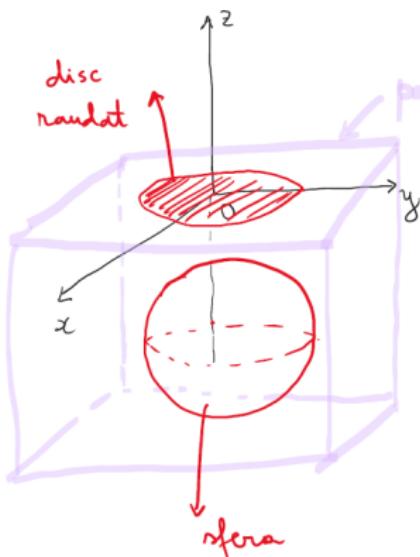
III. Rezolvați complet problemele - 2 subiecte a 5 puncte

Exemple:

- 6.** Se aplică `glm::ortho(-10,10,-10,10,0,10)`, nu este apelată funcția `glm::lookAt()`. În funcția `createVBO()` sunt indicate vârfurile unei sfere de centru $(0, 0, a)$ și de rază 3.0, toate având culoarea roșie. Ce arie va avea figura randată cu roșu dacă (i) $a=-5.0$; (ii) $a=-12.0$? - raportarea se face la sistemul de coordonate inițial, se presupune că nu s-a efectuat scalarea la paralelipipedul normalizat.

III. Rezolvați complet problemele - 2 subiecte a 5 puncte

Soluția problemei 6:



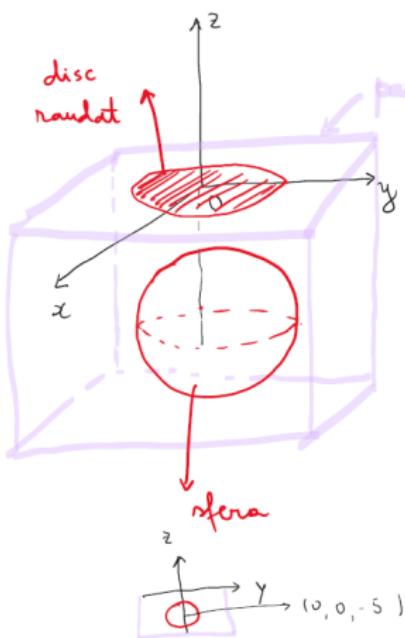
paralelipiped dreptunghic decupat
(fara sferă)
(fata superioară în planul $z=0$ și fata inferioară în planul $z=-10$)

Prin proiecție va fi randat un disc (cerc + interiorul său).

Raza sa depinde de modul în care este decupată sfera.

III. Rezolvați complet problemele - 2 subiecte a 5 puncte

Soluția problemei 6 (continuare):



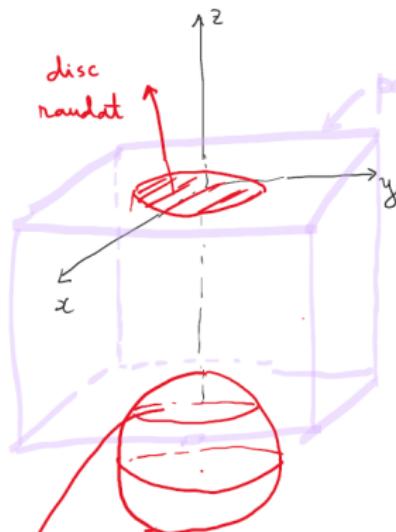
paralelipiped
 dreptunghic decupat
 (față superioară
 în planul $z = 0$
 și față inferioară
 în planul $z = -10$)

a) $a = -5.0$. Centrul sferei este $(0, 0, -5)$,
 iar raza sferei este 3.
 Cel mai de "sus" punct al sferei $(0, 0, -2)$
 și cel mai de "jos" $\rightarrow (0, 0, -8)$
 \Rightarrow întreaga sferă este inclusă în
 paralelipipedul decupat \rightarrow
 raza maximă posibilă a
 unui cerc este 3, deci
 discul răndat va avea aria

$$\underline{\underline{9\pi}}$$

III. Rezolvați complet problemele - 2 subiecte a 5 puncte

Soluția problemei 6 (continuare):

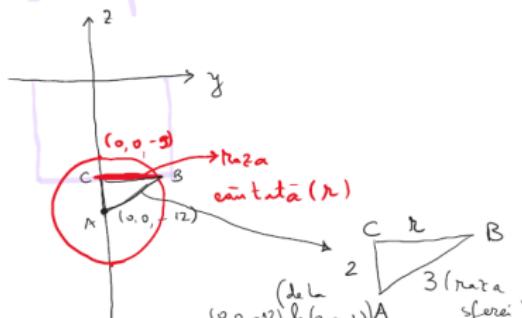


Raza cercului cerc (\cap dintre sfere și pl.
este raza discului rulant

paralelipiped
dreptunghic
deschis

b) $a = -12,0$. Centrul sferei este $(0, 0, -12)$,
iar raza sferei este 3.

Cel mai de "sus" punct al sferei $(0, 0, -9)$
— — "jos" — — — : $(0, 0, -15)$



Rezultă $r = \sqrt{5}$,
deci aria cîntăță
este egală cu 5π .