

# 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\_0LD.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 între 0 și 800, coordonata y între 0 și 600 - “dreptunghi decupat”

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

Funcții specifice pentru deplasare (translație, rotație) - atenție la ordinea în 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

# Funcții pentru transformări în OpenGL

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

# Funcții pentru transformări în OpenGL

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

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

# 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  $\sigma_s$  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  $T_t$  de vector  $t = (t_1, t_2, t_3)$

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

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

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

# Funcții pentru transformări în OpenGL

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

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

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

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

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

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

# Funcții pentru transformări în OpenGL

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

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

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

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

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

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

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

# Aspecte teoretice

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

# Aspecte teoretice

- ▶ **Motivație:** Transformările pot fi modelate cu ajutorul matricelor, **dar** este necesar un cadru în care transformările să fie reprezentate în mod uniform și compunerea lor să fie ușor de descris: **folosind “coordonate omogene” și considerând 4 coordonate.**
- ▶ **De reținut!**
  - (i) orice vârf este reprezentat (intern) ca având 4 coordonate
  - (ii) orice transformare este reprezentată (intern) folosind o matrice  $4 \times 4$
  - (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

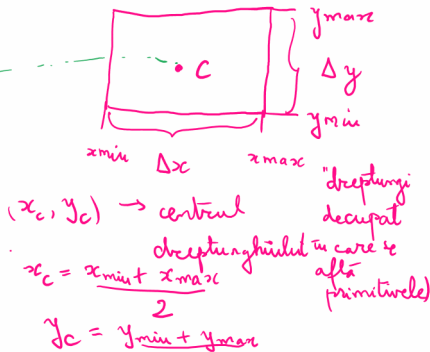
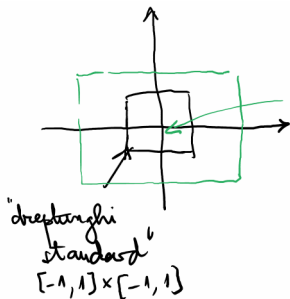
## Din punctul de vedere al implementării...

- 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

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

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

# Compunerea transformărilor; despre glm::ortho - Codul sursă 03\_03\_resize.cpp



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

( $\Rightarrow$  dreptunghi cu centrul în origine)

- (2) scalăm: scalare  $\sigma$  cu factori  $\frac{2}{\Delta x}$ ,  $\frac{2}{\Delta y} \Rightarrow$  matrice  $M_\sigma$   
 $\Rightarrow$  matricea  $M_\sigma * M_T$

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

- fie  $C$  centrul rotației
  - aplicăm translația de vector  $-\vec{OC} = \overset{\text{matrice}}{\underset{-OC}{T}} (M_1)$
  - aplicăm rotația / scalarea ( $M_2$ )
  - aplicăm translația de vector  $\vec{OC} = \overset{\text{matrice}}{\underset{\vec{OC}}{T}} (M_3)$

$\Rightarrow$  matricea folosită este  $\underline{M_3 * M_2 * M_1}$