Transformări

Mihai-Sorin Stupariu

Sem. I, 2021 - 2022

Prezentare generală

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 $t = (t_1, t_2, t_3)$

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

Funcții pentru transformări în OpenGL

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

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

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

Funcții pentru transformări în OpenGL

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

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

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

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

Funcții pentru transformări în OpenGL

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

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

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

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

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

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

Aspecte teoretice

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

Aspecte teoretice

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

Din punctul de vedere al implementării...

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

Din punctul de vedere al implementării...

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

Din punctul de vedere al implementării...

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

  Q: unde/cum indicăm matricele pentru transformări?

Din punctul de vedere al implementării...

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

Din punctul de vedere al implementării...

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

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

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

Din punctul de vedere al implementării...

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

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

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

Despre compunerea transformărilor; cerința (i) L4

Codul sursă 04_03_transformari_glm.cpp.

avem w, h (în 04_03 sunt w = 400, h = 300)

aplicând glm::resize (1/w, 1/h)

transformăm $[-w, w] \times [-h, h]$ în $[-1, 1] \times [-1, 1]$,

apoi se realizează desenarea.

cum procedăm să reprezentăm dreptunghiul $[0, 800] \times [0, 600]$?

translatăm pe axa Ox cu -400

axa Oy cu -300

$$[0,800] \times [0,600] \xrightarrow{T_{(-400,-300)}} [-400,400] \times [-300,300] \xrightarrow{\text{Resize}} (-1,1] \times [-1,1]$$

Despre compunerea transformărilor; cerințele (iii) si (iv) L4

Codul sursă `04_04_transformari_glm_compunere.cpp`.

Obs. importantă. Rotațiile și scalările au ca punct fix originea.

Dacă dorim să realizăm o rotație sau o scalare având centrul C (diferit de $O=(0,0,0)$):

ordinea:

- aplicăm translația de vector $-\overrightarrow{OC}$: $T_{-\overrightarrow{OC}}$ ($-\overrightarrow{OC} \equiv -C$ în coordonate)
- aplicăm rotația / scalarea : Rot/Scal ($\overrightarrow{OC} \equiv C$ în coordonate)
- aplicăm translația de vector $\overrightarrow{OC}$: $T_{\overrightarrow{OC}}$

$\Rightarrow$ matricea este: $\boxed{T_{\overrightarrow{OC}} * (Rot/Scal) * T_{-\overrightarrow{OC}}}$

(v. cod sursă 04_04, linia 226)