Laboratorul 03

Video Laborator 3: https://youtu.be/G-rR8QFZVuI
Autor: Stefania Cristea

Transformări 2D

Obiectele 2D sunt definite într-un sistem de coordonate carteziene 2D, de exemplu, XOY, XOZ sau YOZ. În cadrul acestui laborator vom implementa diferite tipuri de transformări ce pot fi aplicate obiectelor definite în planul XOY: translații, rotații și scalări. Acestea sunt definite în format matriceal, în coordonate omgene, așa cum ați învățat deja la curs. Matricile acestor transformări sunt următoarele:

Translația

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotația

Rotația față de origine

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} cos(u) & -sin(u) & 0 \\ sin(u) & cos(u) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotația față de un punct oarecare

Rotația relativă la un punct oarecare se rezolvă în cel mai simplu mod prin:

1. translatarea atât a punctului asupra căruia se aplică rotația cât și a punctului în jurul căruia se face rotația a.î. cel din urmă să fie originea sistemului de coordonate.
2. rotația normală (în jurul originii),
3. translatarea rezultatului a.î. punctul în jurul căruia s-a făcut rotația să ajungă în poziția sa inițială

Scalarea

Scalarea față de origine

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Dacă $sx = sy$ atunci avem scalare uniformă, altfel avem scalare neuniformă.

Scalarea față de un punct oarecare

Scalarea relativă la un punct oarecare se rezolvă similar cu rotația relativă la un punct oarecare.

Utilizarea bibliotecii GLM

În cadrul laboratorului folosim biblioteca GLM, care este o bibliotecă implementată cu matrici în formă coloană, exact același format ca OpenGL. Forma coloană diferă de forma linie prin ordinea de stocare a elementelor matricei în memorie, Matricea de translație arată în modul următor în memorie:

```
glm::mat3 Translate(float tx, float ty)
{
        return glm::mat3(
                 1,  0, 0,     // coloana 1 in memorie
                 0,  1, 0,     // coloana 2 in memorie
                tx, ty, 1);    // coloana 3 in memorie

}
```

Din această cauză, este convenabil ca matricile să fie scrise manual în forma aceasta:

```
glm::mat3 Translate(float tx, float ty)
{
        return glm::transpose(
                glm::mat3( 1, 0, tx,
                           0, 1, ty,
                           0, 0, 1)
        );
}
```

În cadrul framework-ului de laborator, în fișierul `Transform2D.h` sunt definite funcțiile pentru calculul matricilor de translație, rotație și scalare. În momentul acesta toate funcțiile întorc matricea identitate. În cadrul laboratorului va trebui să modificați codul pentru a calcula matricile respective.

Transformări compuse

De ce sunt necesare matricile? Pentru a reprezenta printr-o singură matrice de transformări o secvență de transformări elementare, în locul aplicării unei secvențe de transformări elementare pe un anume obiect.

Deci, dacă dorim să aplicăm o rotație, o scalare și o translație pe un obiect, nu facem rotația obiectului, scalarea obiectului urmată de translația lui, ci calculăm o matrice care reprezintă transformarea compusă (de rotație, scalare și translație), după care aplicăm această transformare compusă pe obiectul care se dorește a fi transformat.

Astfel, dacă dorim să aplicăm o rotație (cu matricea de rotație R), urmată de o scalare (S), urmată de o translație (T) pe un punct (x,y), punctul transformat (x',y') se va calcula astfel:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(u) & -sin(u) & 0 \\ sin(u) & cos(u) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Deci, matricea de transformări compuse M este $M = T * S * R$.

În cadrul laboratorului, în fișierul `Laborator3.cpp`, există o serie de obiecte (pătrate) pentru care, în funcția `Update()`, înainte de desenare, se definesc matricile de transformări. Comanda de desenare se dă prin funcția `RenderMesh2D()`.

```
modelMatrix = glm::mat3(1);
modelMatrix *= Transform2D::Translate(150, 250);
RenderMesh2D(meshes["square1"], shaders["VertexColor"], modelMatrix);
```

Pentru exemplul anterior, matricea de translație creată va avea ca efect translatarea pătratului curent cu (150, 250). Pentru efecte de animație continuă, pașii de translație ar trebui să se modifice în timp.

Exemplu:

```
tx += deltaTimeSeconds * 100;
ty += deltaTimeSeconds * 100;
model_matrix *= Transform2D::Translate(tx, ty);
```

Rețineți: dacă la animație nu țineți cont de timpul de rulare al unui frame (`deltaTimeSeconds`), veți crea animații dependente de platformă.

Exemplu: dacă la fiecare frame creșteți pe tx cu un pas constant (ex: `tx += 0.01`), atunci animația se va comporta diferit pe un calculator care merge mai repede față de unul care merge mai încet. Pe un calculator care rulează la 50 FPS, obiectul se va deplasa 0.01 * 50 = 0.5 unități în dreapta într-o secundă. În schimb, pe un calculator mai încet, care rulează la 10 FPS, obiectul se va deplasa 0.01 * 10 = 0.1 unități în dreapta într-o secundă, deci animația va fi de 5 ori mai lentă.

Din acest motiv este bine să țineți cont de viteza de rulare a fiecărui calculator (dată prin `deltaTimeSeconds`, care reprezintă timpul de rulare al frame-ului anterior) și să modificați pașii de translație, unghiurile de rotație și factorii de scalare în funcție de această variabilă.

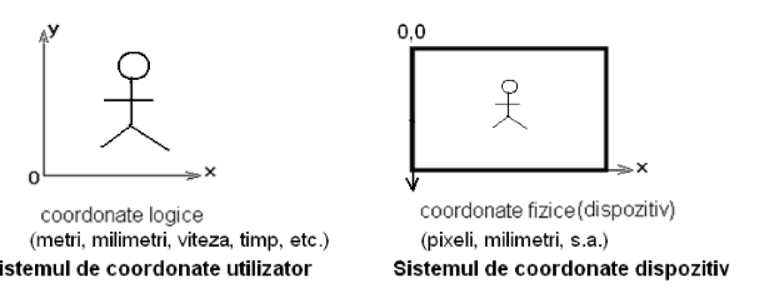
Transformarea fereastra-poartă

Desenele reprezentate într-un program de aplicație grafică (2D sau 3D) sunt, de regulă, raportate la un sistem de coordonate diferit de cel al suprafeței de afișare.

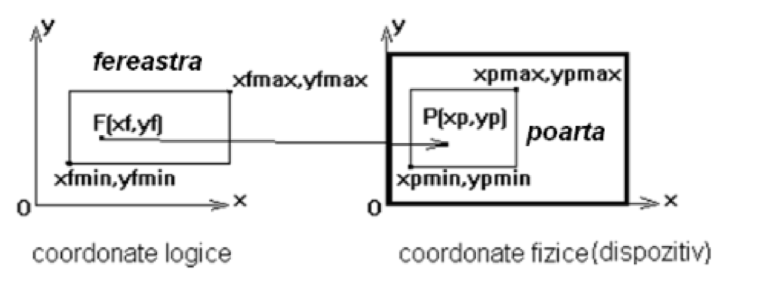
În exercițiile anterioare din acest laborator, coordonatele obiectelor au fost raportate la dimensiunea ferestrei definită prin `glViewport()`.

Exemplu: Dacă viewport-ul meu are colțul din stânga jos (0, 0) și are lățimea 1280 și înălțimea 720, atunci toate obiectele ar trebui desenate în acest interval, dacă vreau să fie vizibile. Acest lucru mă condiționează să îmi gândesc toată scena în (0, 0) - (1280, 720). Dacă vreau să scap de această limitare, pot să îmi gândesc scena într-un spațiu logic (de exemplu îmi creez toate obiectele în spațiul (-1, -1) - (1, 1) și apoi să le desenez în poarta de afișare, dar aplicând ceea ce se numește **transformarea fereastră poartă**.

În cele ce urmează vedem ce presupune această transformare și cum pot să imi gândesc scena fără să fiu limitat de dimensiunea viewport-ului.



Definiția matematică:



$$\frac{xp - xpmin}{xpmax - xpmin} = \frac{xf - xfmin}{xfmax - xfmin}$$

$$\frac{yp - ypmin}{ypmax - ypmin} = \frac{yf - yfmin}{yfmax - yfmin}$$

Transformarea este definită prin 2 dreptunghiuri, în cele două sisteme de coordonate, numite fereastră sau spațiul logic și poartă sau spațiul de afișare. De aici numele de transformarea fereastră-poartă sau transformarea de vizualizare 2D.

F: un punct din fereastră

P: punctul în care se transformă F prin transformarea de vizualizare

Poziția relativă a lui P în poarta de afișare trebuie să fie aceeași cu poziția relativă a lui F în fereastră.

$$sx = \frac{xpmax - xpmin}{xfmax - xfmin}$$

$$sy = \frac{ypmax - ypmin}{yfmax - yfmin}$$

- sx, sy depind de dimensiunile celor două ferestre
- tx, ty depind de pozițiile celor două ferestre față de originea sistemului de coordonate în care sunt definite

$$tx = xpmin - sx * xfmin$$

$$ty = ypmin - sy * yfmin$$

În final, transformarea fereastră-poartă are următoarele ecuații:

$$xp = xf * sx + tx$$

$$yp = yf * sy + ty$$

Considerăm o aceeași orientare a axelor celor două sisteme de coordonate. Dacă acestea au orientări diferite (ca în prima imagine), trebuie aplicată o transformare suplimentară de corecție a coordonatei y.

Efectele transformării

- mărire/micșorare, în funcție de dimensiunile ferestrei și ale porții
- deformare dacă fereastra și poarta nu sunt dreptunghiuri asemenea
- pentru scalare uniformă, $s = min(sx, sy)$, afișarea centrată în poartă presupune o translație suplimentară pe axa Ox sau pe axa Oy:

$$Tsx = (xpmax - xpmin - s * (xfmax - xfmin))/2$$

$$Tsy = (ypmax - ypmin - s * (yfmax - yfmin))/2$$

- decuparea primitivelor aflate în afara ferestrei vizuale

Matricea transformării fereastră-poartă

De reținut este că transformarea fereastră-poartă presupune o scalare și o translație. Ea are următoarea expresie, cu formulele de calcul pentru sx, sy, tx, ty prezentate anterior:

$$\begin{bmatrix} xp \\ yp \\ 1 \end{bmatrix} = \begin{bmatrix} sx & 0 & tx \\ 0 & sy & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} xf \\ yf \\ 1 \end{bmatrix}$$

Transformarea de vizualizare este deja implementată în clasa `Laborator3_Vis2D`:

```
//2D vizualization matrix
glm::mat3 Laborator3_Vis2D::VisualizationTransf2D(const LogicSpace & logicSpace, const Vie
{
        float sx, sy, tx, ty;
        sx = viewSpace.width / logicSpace.width;
        sy = viewSpace.height / logicSpace.height;
        tx = viewSpace.x - sx * logicSpace.x;
        ty = viewSpace.y - sy * logicSpace.y;

        return glm::transpose(glm::mat3(
                sx, 0.0f, tx,
                0.0f, sy, ty,
                0.0f, 0.0f, 1.0f));
}
```

În cadrul laboratorului, în clasa `Laborator3_Vis2D`, este creat un pătrat, în spațiul logic (0,0) - (4,4). De reținut este faptul că acum nu mai trebuie să raportăm coordonatele pătratului la spațiul de vizualizare (cum se intamplă în exercițiile anterioare), ci la spațiul logic pe care l-am definit noi.

```
logicSpace.x = 0;       // logic x
logicSpace.y = 0;       // logic y
logicSpace.width = 4;   // logic width
logicSpace.height = 4;  // logic height

glm::vec3 corner = glm::vec3(0.001, 0.001, 0);
length = 0.99f;

Mesh* square1 = Object2D::CreateSquare("square1", corner, length, glm::vec3(1, 0, 0));
AddMeshToList(square1);
```

În funcția `Update()` se desenează același pătrat creat anterior, de 5 ori: patru pătrate în cele patru colțuri și un pătrat în mijlocul spațiului logic. Se definesc 2 viewport-uri, ambele conținând aceleași obiecte. Primul viewport este definit în jumătatea din stânga a ferestrei de afișare, iar al doilea, în jumătatea din dreapta. Pentru primul viewport se definește transformarea fereastră-poartă default și pentru al doilea viewport, cea uniformă. Observați că în al doilea viewport pătratele rămân întotdeauna pătrate, pe când în primul viewport se văd ca dreptunghiuri (adică sunt deformate), dacă spațiul logic și spațiul de vizualizare nu sunt reprezentate prin dreptunghiuri asemenea.

Utilizare

Unde se poate folosi această transformare fereastră-poartă? De exemplu, într-un joc 2D cu mașini de curse, se dorește în dreapta-jos a ecranului vizualizarea mașinii proprii, într-un minimap. Acest lucru se face prin desenarea scenei de două ori.

Dacă de exemplu toată scena (traseul și toate mașinile) este gândită în spațiul logic (-10,-10) - (10,10) (care are dimensiunea 20×20) și spațiul de afișare este (0,0) - (1280, 720), prima dată se desenează toată scena cu parametrii funcției fereastră-poartă anterior menționați:

```
LogicSpace logic_space = LogicSpace(-10, -10, 20, 20);
ViewportSpace view_space = ViewportSpace(0, 0, 1280, 720);
vis_matrix *= VisualizationTransf2D(logic_space, view_space);
```

Dacă la un moment dat mașina proprie este în spațiul (2,2) - (5,5), adică de dimensiune 3×3 și vreau să creez un minimap în colțul din dreapta jos al ecranului de rezoluție 280×220, pot desena din nou aceeași scenă, dar cu urmatoarea transformare fereastră-poartă:

```
LogicSpace logic_space = LogicSpace(2, 2, 3, 3);
ViewportSpace view_space = ViewportSpace(1000, 500, 280, 220);
vis_matrix *= VisualizationTransf2D(logic_space, view_space);
```

Laboratorul 3

Descriere laborator

În cadrul acestui laborator aveți de programat în două clase:

- `Laborator3.cpp`, pentru familiarizarea cu transformările 2D de translație, rotație și scalare
- `Laborator3_Vis2D.cpp`, pentru familiarizarea cu transformarea fereastră-poartă

Din clasa `Main` puteți să alegeți ce laborator rulați:

```
World *world = new Laborator3();
```

sau

```
World *world = new Laborator3_Vis2D();
```

OpenGL este un API 3D. Desenarea obiectelor 2D și aplicarea transformărilor 2D sunt simulate prin faptul că facem abstracție de coordonata z.

Transformarea fereastră-poartă este și ea simulată pentru acest framework, dar veți învăța pe parcurs că ea este de fapt inclusă în lanțul de transformări OpenGL și că nu trebuie definită explicit.

Cerințe laborator

1. Descarcăți framework-ul de laborator
2. Completați funcțiile de translație, rotație și scalare din `/Laborator3/Transform2D.h`
3. Să se modifice pașii de translație, rotație și scalare pentru cele trei pătrate ca să se creeze animații.
4. Cu tastele W, A, S, D să se translateze fereastra logică `Laborator3_Vis2D`. Cu tastele Z și X să se facă zoom in și zoom out pe fereastra logică.

Info curs
- Elemente de Grafică pe Calculator
- Infographie

Cataloage EGC
- TBA

Laboratoare
- Laboratorul 01
- Laboratorul 02
- Laboratorul 03
- Laboratorul 04
- Laboratorul 05
- Laboratorul 06
- Laboratorul 07
- Prezentare Tema 1
- Laboratorul 08
- Laboratorul 09
- Vacanță
- Prezentare Tema 2
- Recuperări laborator
- Prezentare Tema 3
- Resurse: Redare text

Teme
- Regulament General
- Tema 1 - Bow and Arrow
- Tema 2 - Skyroads
- Tema 3 - Stylised Runner

Resurse
- Resurse Utile
- Notare

Table of Contents