

10/12/2013

FACULTATEA
DE
AUTOMATICA SI
CALCULATOARE

ELEMENTE DE GRAFICA PE CALCULATOR

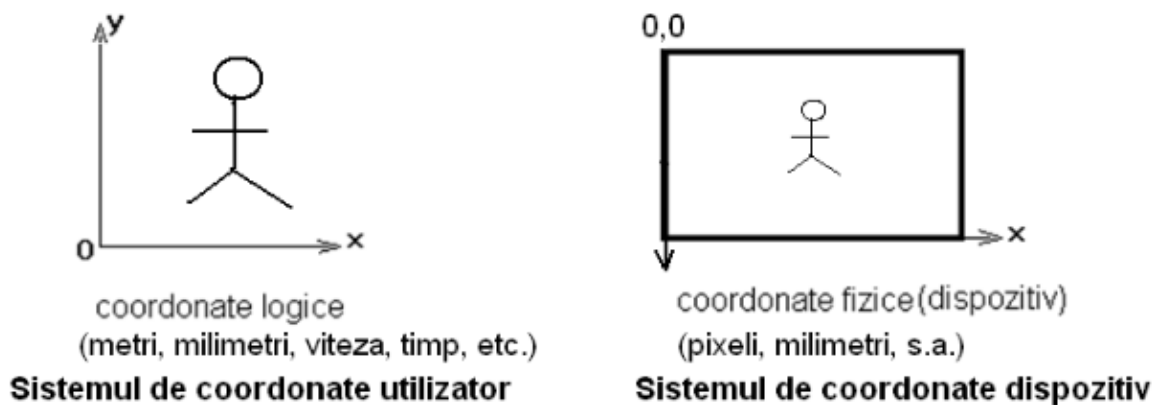


Laborator 2

Transformarea de vizualizare 2D (transformarea fereastră – poarta)

1. Teorie

Desenele reprezentate intr-un program de aplicatie grafica 2D sunt, de regula, raportate la un sistem de coordonate diferit de cel al suprafetei de afisare.



Definitia matematica:



Transformarea este definita prin 2 dreptunghiuri, in cele doua sisteme de coordonate, numite *fereastră (de vizualizare)* si *poarta (de afisare)*.

Transformarea se mai numeste *fereastră-poarta*.

F: un punct din fereastră

P: punctul in care se transforma F prin transformarea de vizualizare

Pozitia relativa a lui P in poarta de afisare trebuie sa fie aceeași cu pozitia relativa a lui F in fereastră.

$$sx = \frac{xp_{max} - xp_{min}}{xf_{max} - xf_{min}}$$

$$sy = \frac{yp_{max} - yp_{min}}{yf_{max} - yf_{min}}$$

- sx, sy depind de dimensiunile celor doua ferestre
- tx, ty depind de pozitiile celor doua ferestre fata de originea sistemului de coordonate in care sunt definite.

$$tx = xp_{min} - sx * xf_{min} \quad ty = yp_{min} - sy * yf_{min}$$

$$xp = xf * sx + tx$$

$$yp = yf * sy + ty$$

Consideram o aceeași orientare a axelor celor două sisteme de coordonate.

Efectele transformării:

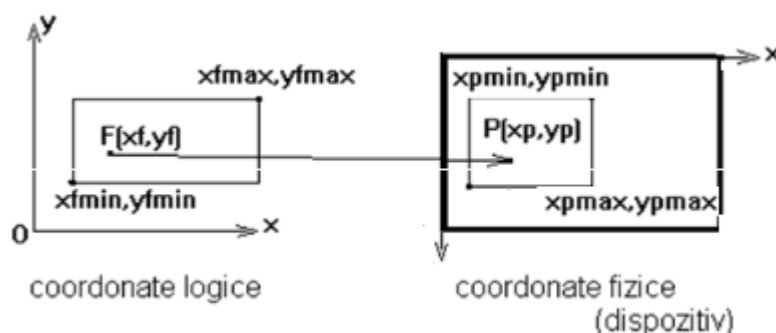
- marire/micsorare, in functie de dimensiunile ferestrei si ale portii
- deformare: fereastra si poarta nu sunt dreptunghiuri asemenea
- scalare uniforma: $s = \min(sx, sy)$
- afisare centrata in poarta: translatie suplimentara pe axa Ox sau pe axa Oy:

$$Tsx = (xp_{max} - xp_{min} - s * (xf_{max} - xf_{min})) / 2$$

$$Tsy = (yp_{max} - yp_{min} - s * (yf_{max} - yf_{min})) / 2$$

- decuparea primitivelor aflate in afara ferestrei vizuale

Corectia coordonatei yp:



$$yp' = -yp + yp_{max} + yp_{min}$$

2. Clasa Visual2D

- metode importante:

- constructor – creeaza un context vizual:
`Visual2D(float xf1,float yf1,float xf2,float yf2,int xp1,int yp1,int xp2,int yp2);`
 - xf1, yf1, xf2, yf2 – colturi stanga jos si dreapta sus pentru fereastra de vizualizare
 - xp1, yp1, xp2, yp2 – colturi stanga sus si dreapta jos pentru poarta de afisare
- `void tipTran(bool tip);`
 - stabileste daca transformarea fereastra-poarta este una uniforma
- `void fereastra(float x1,float y1,float x2,float y2);`
 - stabileste fereastra contextului vizual
- `void poarta(int x1,int y1,int x2,int y2);`
 - stabileste poarta contextului vizual
- `void cadruFereastra(Color color);`
 - deseneaza un cadru pentru fereastra, in culoarea color
- `void cadruPoarta(Color color);`
 - deseneaza un cadru pentru poarta in culoarea color
- `float xLog(int xp);`
 - intoarce coordonata x in sistemul de coordonate logice (reale)
- `float yLog(int yp);`
 - intoarce coordonata y in sistemul de coordonate logice (reale)

Pentru a adauga un context vizual la **DrawingWindow**, trebuie apelata functia

`static void addVisual2D(Visual2D *v);`

3. Clasa DrawingWindow

Pe langa containerul de obiecte *objects2D* din clasa *DrawingWindow*, care contine obiectele care vor fi desenate indiferent daca definim un context vizual sau nu, si care vor fi desenate in toate contextele vizuale, fiecare obiect *Visual2D* poate avea unele obiecte care sunt continute numai in acel context vizual. Metoda care permite adaugarea unui obiect intr-un context vizual:

`static void addObject2D_to_Visual2D(Object2D *o, Visual2D *v);`

4. Main

In plus fata de laboratorul 1, in clasa fisierul *main.cpp* mai aveti o functie care se apeleaza la fiecare redimensionare a ferestrei de afisare (a portii):

`onReshape(int width,int height)`

- *width* si *height* sunt noua latime si inaltime a ferestrei de afisare.
- daca se doreste ca poarta unui context vizual sa tina cont de dimensiunile curente ale ferestrei de afisare, atunci poarta contextului ar trebui redefinita la fiecare redimensionare (la fiecare apelare a functiei *onReshape()*)

5. In plus fata de laboratorul 1

In plus fata de laboratorul 1, s-au adaugat urmatoarele elemente/explicatii:

a) **clasa Text**, care permite afisarea de text pe ecran

- pentru a scrie un text pe ecran, acesta trebuie intai creat, si apoi adaugat la containerul de obiecte text din **DrawingWindow**:

Text(string text)

- constructor: creeaza un obiect **Text**, ce contine textul *text*, care e pozitionat in 0,0, de culoare neagra

Text(string text, Point2D pos, Color color, void font)*

- constructor: creeaza un obiect **Text**, care e pozitionat in *pos* (coordonate fereastra de vizualizare), de culoare *color*, si fontul *font*. Pentru variabila *font* exista mai multe constante definite in clasa **Text**:

```
#define BITMAP_8_BY_13 GLUT_BITMAP_8_BY_13
#define BITMAP_9_BY_15 GLUT_BITMAP_9_BY_15
#define BITMAP_TIMES_ROMAN_10 GLUT_BITMAP_TIMES_ROMAN_10
#define BITMAP_TIMES_ROMAN_24 GLUT_BITMAP_TIMES_ROMAN_24
#define BITMAP_HELVETICA_10 GLUT_BITMAP_HELVETICA_10
#define BITMAP_HELVETICA_12 GLUT_BITMAP_HELVETICA_12
#define BITMAP_HELVETICA_18 GLUT_BITMAP_HELVETICA_18
```

*static void addText(Text *text);*

- functie din **DrawingWindow** care permite adaugarea de text la containerul de texte ce vor fi afisate pe ecran

*static void addText_to_Visual2D(Text *text, Visual2D *v);*

- functie din **DrawingWindow** care permite adaugarea de text la contextual vizual *v*.

b) **Functii de stergere obiecte, context vizuale, text:**

```
static void removeObject2D(Object2D *o);
static void removeVisual2D(Visual2D *v);
static void removeObject2D_from_Visual2D(Object2D *o, Visual2D *v);
static void removeText(Text *text);
static void removeText_from_Visual2D(Text *text, Visual2D *v);
```

c) **Alte explicatii:**

void DrawingWindow::onKey(unsigned char key)

- se apeleaza la apasarea unei tastaturi
- daca doriti sa folositi taste speciale (exemplu: sageata stanga, sageata dreapta, etc), puteti folosi constantele definite in `DrawingWindow.h`:

```
#define KEY_DOWN GLUT_KEY_DOWN
#define KEY_LEFT GLUT_KEY_LEFT
#define KEY_UP GLUT_KEY_UP
#define KEY_RIGHT GLUT_KEY_RIGHT
#define KEY_ESC 27
#define KEY_SPACE 32
```

`void DrawingWindow::onMouse(int button, int state, int x, int y)`

- se apeleaza la click pe mouse
- `x` si `y` sunt coordonatele in poarta de afisare, unde s-a dat click pe mouse
- daca doriti sa aflati coordonatele logice (din fereastra de vizualizare), puteti folosi functiile `xLog` si `yLog` din **Visual2D**.

3D

Intre 2D si 3D exista foarte multe asemanari. Din acest motiv, nu se va pune accent pe elementele similare din 2D.

1. Obiecte3D – clasa **Object3D**

Un obiect 3D este definit printr-un vector de varfuri, *vertices*, un vector de varfuri transformate, *transf_vertices*, un vector de fete, *faces*, o culoare *color* (de tip **Color**) si un atribut *fill* (false => obiect wireframe; true => obiect solid).

O fata (clasa **Face**) contine un vector de indecsi, numit *coutour*. Elementele vectorului *contour* reprezinta indecsii varfurilor, din vectorul de *vertices*, ce formeaza conturul fetei respective.


Exemplu de definire a unui obiect 3D:

```
vector <Point3D*> vertices;
vector <Face*> faces;
//varfurile de jos
vertices.push_back(new Point3D(0,0,0));
vertices.push_back(new Point3D(n,0,0));
vertices.push_back(new Point3D(n,0,n));
vertices.push_back(new Point3D(0,0,n));
//varfurile de sus
vertices.push_back(new Point3D(0,n,0));
vertices.push_back(new Point3D(n,n,0));
vertices.push_back(new Point3D(n,n,n));
vertices.push_back(new Point3D(0,n,n));

vector <int> contour;
```

```
//cele 6 fete
//fata jos
contour.clear();
contour.push_back(0);
contour.push_back(1);
contour.push_back(2);
contour.push_back(3);
faces.push_back(new Face(contour));
//fata sus
contour.clear();
contour.push_back(4);
contour.push_back(5);
contour.push_back(6);
contour.push_back(7);
faces.push_back(new Face(contour));
//fata fata
contour.clear();
contour.push_back(0);
contour.push_back(1);
contour.push_back(5);
contour.push_back(4);
faces.push_back(new Face(contour));
//fata dreapta
contour.clear();
contour.push_back(1);
contour.push_back(2);
contour.push_back(6);
contour.push_back(5);
faces.push_back(new Face(contour));
//fata spate
contour.clear();
contour.push_back(2);
contour.push_back(3);
contour.push_back(7);
contour.push_back(6);
faces.push_back(new Face(contour));
//fata stanga
contour.clear();
contour.push_back(3);
contour.push_back(0);
contour.push_back(4);
contour.push_back(7);
faces.push_back(new Face(contour));

Object3D *cube = new Object3D(vertices,faces,Color(1,0,0),false);
addObject3D(cube);
```



2. Transformari 3D – clasa Transform3D

Daca in laboratorul trecut am discutat despre transformarile 2D, rotatiile, scalarile si translatiile, la acest laborator vom trata aceleasi transformari, dar in 3D. Functiile care definesc matricile de translatie, rotatie si scalare sunt definite in clasa **Transform3D**. Acestea vor trebui implementate la laborator. Deoarece clasa **Transform3D** are aceeaasi structura si mod de functionare ca **Transform2D**, nu vom repeta explicatiile din laboratorul trecut (sau de la Visual2D).

Translatia:

$$T = \begin{pmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Scalarea:

$$S = \begin{pmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotatia fata de axa Ox:

$$R(Ox) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(u) & -\sin(u) & 0 \\ 0 & \sin(u) & \cos(u) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotatia fata de axa Oy:

$$R(Oy) = \begin{pmatrix} \cos(u) & 0 & -\sin(u) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(u) & 0 & \cos(u) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotatia fata de axa Oz:

$$R(Oz) = \begin{pmatrix} \cos(u) & -\sin(u) & 0 & 0 \\ \sin(u) & \cos(u) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Responsabil laborator: Anca Morar