10/29/2012

FACULTATEA DE AUTOMATICA SI CALCULATOARE

ELEMENTE DE GRAFICA PE CALCULATOR

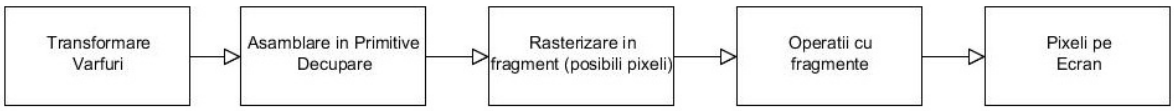
Laborator 4

Nota: demonstratiile sunt la sfarsit sub forma de anexe.

1. Banda grafica pe scurt

Transformarea descrierii scenei 3D intr-o imagine afisata pe ecran este realizata automat de API-urile 3D, printr-un lant de prelucrari **fix**. Acest lucru este valabil pentru versiunea de OpenGL 1.0, care este predata la laborator. Versiunea curenta de OpenGL este 4.3 si este foarte diferita fata de 1.0, dar conceptele din laborator nu sunt influentate in vreun fel de versiunea aleasa, 1.0 avand avantajul de a include modul de desenare imediat. (glVertex3f..) ce este semnificativ mai usor de folosit pentru incepatori.

Lantul de prelucrari consta din etapele redate in figura urmatoare:

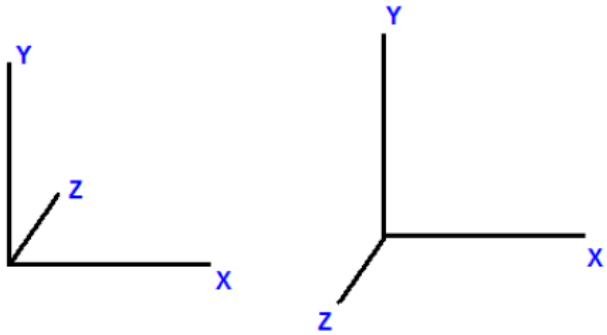


Deoarece fiecare etapa primeste intrari de la etapele anterioare si transmite rezultate pentru etapele urmatoare, secventa de operatii este asemanata cu o "banda de asamblare" intr-un proces de fabricatie. Deoarece obiectul fabricat in acest caz este imaginea unei scene 3D, banda este denumita „banda grafica" („graphics pipeline"). Operatiile din banda grafica fixa, redata in figura de mai sus, sunt aceleasi pentru toate primitivele unei scene 3D. Semnificatia fiecarei etape, foarte pe scurt:

- Transformare varfuri – se ia fiecare varf(vertex) din scena si se transforma in parelel, pe placa grafica, pana cand ajunge in pozitia sa finala (dupa proiectie)
- Asamblare in Primitive si Decupare – cu vertecsii de la etapa precedenta si cu informatia ce defineste legarea lor se creeaza triunghiuri. Triunghiurile ce sunt in afara volumului vizual sunt decupate.
- Rasterizare in fragmente – se iau primitivele de la etapa precedenta si se transforma in patrate de dimensiunea pixelilor ce se numesc fragmente.
- Operatii cu fragmente – se iau fragmentele de la etapa precedenta si se fac operatii pe ele: colorare, iluminare, etc.
- Pixeli pe ecran – fragmentele ce au trecut de etapa precedenta sunt procesate alaturi de pixelii deja existenti.

2. Sisteme de axe si operatii

Inainte de a putea trece la operatiile efective trebuie inteles modul in care OpenGL lucreaza cu datele. Toate datele sunt prelucrate prin operatii vectoriale/matriceale, de aceea este nevoie de o intelegere clara a acestor operatii cat si a spatiului in care sunt facute. OpenGL foloseste un sistem de coordonate dreapta, deci se respecta regula mainii drepte, nu regula mainii stangi. Practic in OpenGL axa Z iese din ecran iar intr-un sistem de coordonate stanga (Direct3D) axa Z intra in ecran.
In figura urmatoare se poate vedea diferenta dintre un sistem de coordonate dreapta si un sistem de coordonate stanga.



Left-handed Right-handed

Trebuie mentionat faptul ca se folosesc matrici si vectori in forma **coloana**. Ca o scurta recapitulare:

- Forma Linie: [x' , y' , z'] = [x, y, z] * M1 * M2 * M3
- Forma Coloana $[x', y', z']_T$ = M3 * M2 * M1 * $[x, y, z]^T$ unde T reprezinta operatorul de transpunere.

Deci in OpenGL daca se doreste mai intai rotatie , apoi translatia si la sfarsit scalarea unui set de coordonate rezultatul final este: $[x', y', z']_T$ = S * T * R * $[x, y, z]_T$

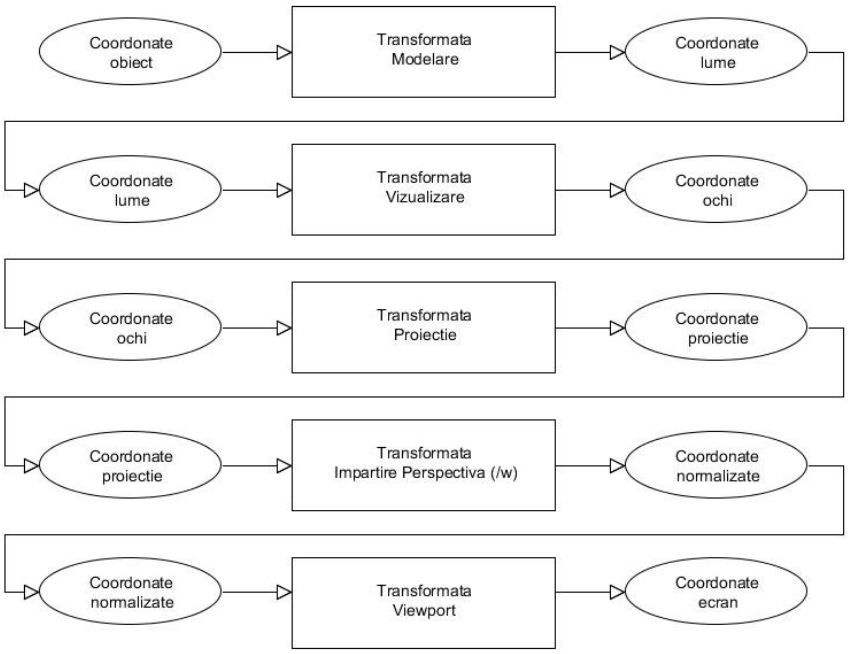
OpenGL lucreaza cu matrici de dimensiunea 4x4, iar aceastea pot fi obtinute sub forma liniara, deci un vector de 16 float-uri. Coeficientii unei matrici sunt numerotati astfel (ordinea lor poate creea usor confuzii!):

m0 m4 m8 m12
m1 m5 m9 m13
m2 m6 m10 m14
m3 m7 m11 m15

Deci primele 4 valori din matrice in forma liniara reprezinta prima coloana nu prima linie!

3. Lantul de transformari

Lantul de transformari pe care il urmeaza un varf este impartit in mai multe etape, dar inainte ca aceastea sa fie explicate este indicat sa fie facuta o prezentare de ansamblu a intregului proces.



Dupa cum se observa in imagine un varf trece printr-o serie de ***spatii***, fiecare avand o semnificatie specifica.

- In spatiul obiect sunt modelate coordonatele varfurilor unui obiect, in spatiul lume este pozitionat acest obiect intr-o scena,
- in spatiul ochi sunt reprezentate obiectele asa cum sunt ele vazute de observator, in spatiul proiectie sunt pozitionate obiectele intr-un volum de vizualizare,
- in spatiul NDC(normalized device coordinates) sunt pozitionate obiectele intr-un volum canonic(cub de latura 2 centrat in origine) de vizualizare,
- in spatiul ecran obiectele sunt pozitionate relativ la fereastra de afisare.

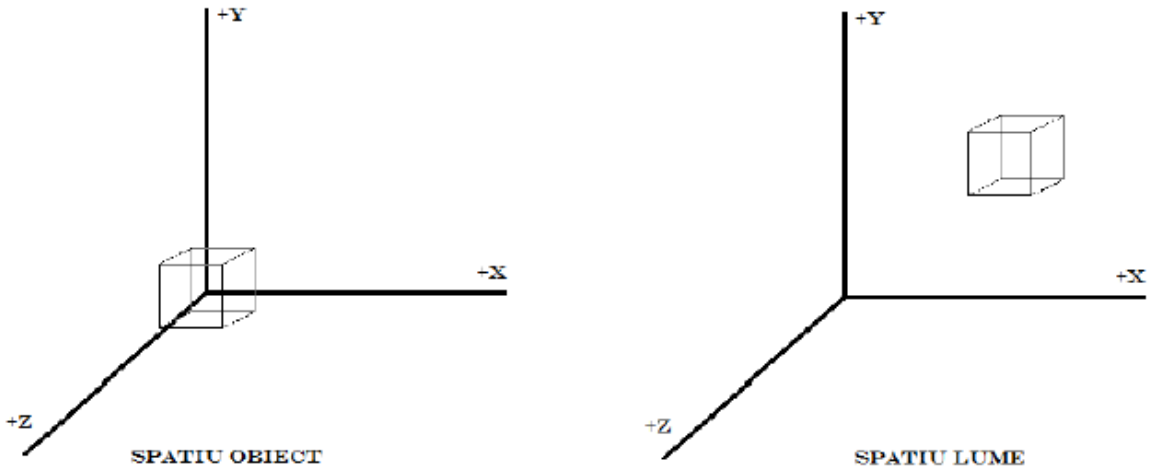
In cazul OpenGL, matricile de modelare si de vizualizare sunt grupate intr-o singura matrice numita GL_MODELVIEW. Matricea de proiectie este numita GL_PROJECTION. Matricea cu care lucram poate fi specificata prin comanda glMatrixMode(matrice). Astfel, pentru a putea desena o scena in OpenGL este necesar sa specificam 3 transformari: modelarevizualizare, proiectie si viewport.

4. Specificarea Transformatelor

Transformarea de MODELARE pozitioneaza un obiect undeva in lume.Puteti folosi rotatii, translatii, scalari. Obiectele le puteti aseza/transforma in scena folosind:

- **glScalef(GLfloat x, GLfloat y, GLfloat z)**
- **glTranslatef(GLfloat x, GLfloat y, GLfloat z)**
- **glRotatef (GLfloat unghi, GLfloat x, GLfloat y, GLfloat z)**

Matricea de transformare din coordonate obiect in coordonate lume va fi deci dependenta doar de dimensiunea, orientarea si pozitia obiectului, dupa cum vedem in imagine:



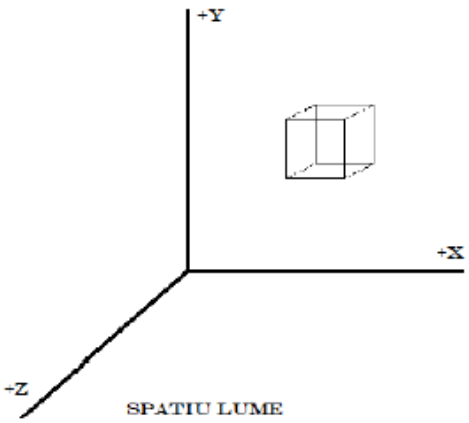
Transformarea de VIZUALIZARE pozitioneaza observatorul si astfel perceptia utilizatorului asupra spatiului observat. Pentru a stabili transformata de vizualizare se foloseste functia:

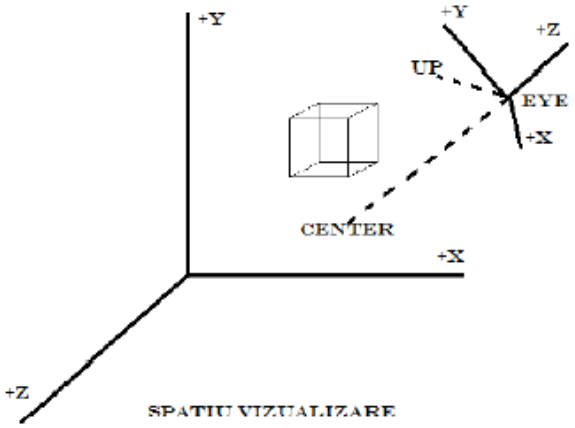
gluLookAt (GLdouble eye_x, GLdouble eye_y, GLdouble eye_z,
GLdouble center_x, GLdouble center_y, GLdouble center_z,
GLdouble up_x, GLdouble up_y, Gldouble up_z)

unde:

- (eye_x, eye_y, eye_z) reprezinta pozitia observatorului
- (center_x, center_y, center_z) arata directia in care priveste acesta
- (up_x, up_y, up_z) reprezinta directia sus din planul de vizualizare

O imagine in care se prezinta transformarea de vizualizare:





Transformata de PROIECTIE face scena sa treaca dintr-un spatiu bidimensional intr-un spatiu tridimensional. Pentru a face acest lucru trebuie sa delimitam spatiul vizibil, numit **volumul de vizualizare**. OpenGL permite mai multe moduri de a face acest lucru.

Proiectia ortografica se realizeaza cu ajutorul functiei:

• glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)

Volumul de vizualizare este un paralelipiped dreptunghic. Nu uitati insa ca pozitia observatorul este implicit in origine si priveste in directia negativa a axei oz. Deci doar obiectele cu z intre -near si -far vor fi vizibile.

Proiectia perspectiva se poate realiza in doua moduri:

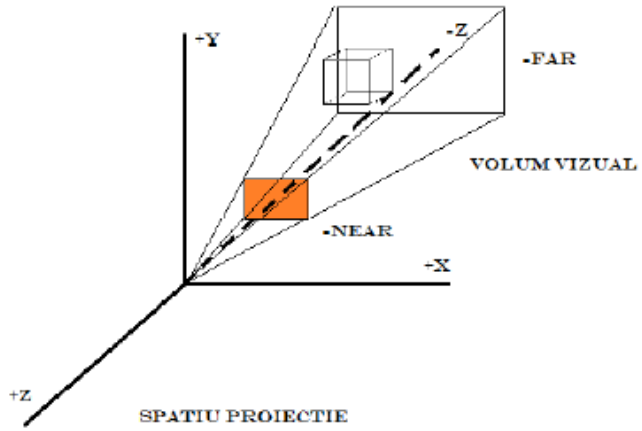
• glFrustum(GLdouble left, GLdouble right, GLdouble bottom,GLdouble top, GLdouble near, GLdouble far)

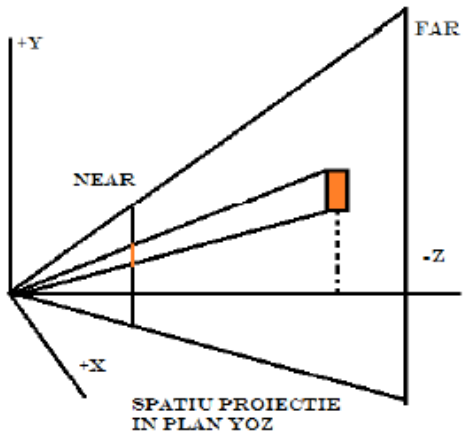
left, right, bottom, top se refera in acest caz la fereastra din planul apropiat. Cum stim pozitia observatorului (e cea implicita sau cea setata de noi cu gluLookAt) putem calcula volumul de vizualizare. Distantele near si far trebuie sa fie pozitive.

Un alt mod de a specifica acest volum de vizualizare este cu ajutorul functiei:

• gluPerspective(GLdouble fovy, GLdouble aspect, GLdoublenear, GLdouble far)

Ati recunoscut deja parametrii near si far. Parametrul aspect reprezinta raportul dintre lungimea si inaltimea ferestrei in planul de aproape. Pentru a obtine o imagine nedistorsionata acest raport trebuie sa corespunda cu cel al ferestrei de afisare. Parametrul fovy este unghiul de vizualizare in planul xOz si trebuie sa fie in intervalul [0.0, 180.0].





Transformata VIEWPORT se defineste folosind functia:

• glViewport (GLint px, GLint py, GLsizei width, Glsizei height)

unde : px, py reprezinta coordonatele in fereastra ale coltului stanga jos al portii. Implicit sunt 0,0. width, height sunt latimea si inaltimea portii. Valorile implicite sunt date de latimea si inaltimea ferestrei in care se afiseaza. Este o transformare fereastra poarta cu mentiunea ca z este in intervalul (0,1).

ANEXA 1 – gluLookAt

http://www.opengl.org/sdk/docs/man/xhtml/gluLookAt.xml

```
Vector3 forward = center – eye;
f.normalize();
up.normalize();
Vector3 right = forward X up;              //cross product!
Vector3 newup = right X forward;           //cross product!

m[0]=right.x;     m[4]=right.y;     m[8]=right.z;
m[1]=newup.x;     m[5]=newup.y;     m[9]=newup.z;
m[2]=-forward.x;  m[6]=-forward.y;  m[10]=-forward.z;
m[15]=1;

translate(-eye.x, -eye.y, -eye.z)
```

Demonstratie: se considera 2 sisteme de ecuatii liniare: unul ce reprezinta pozitia punctului transformat in coordonate lume si altul ce reprezinta pozitia punctului transformat in coordonate camera. Daca pozitia in primul sistem de coordonate o obtinem usor (translate/rotate/scale cu informatii cunoscute a priori), pozitia in cel de-al doilea sistem de coordonate e obtinuta prin ortogonalizarea vectorilor dati ca argument la gluLookAt si apoi obtinerea din acestia a unor versori. Avem un sistem de ecuatii iar matricea oferita de aceasta soltie este chiar matricea din referinta de mai sus.

ANEXA2 – gluPerspective

http://www.opengl.org/sdk/docs/man/xhtml/gluPerspective.xml

```
float f=1.0f/tan(fovy);                    //radians
m[0]=f/aspect;
m[5]=f;
m[10]=(zfar+znear)/(znear-zfar);
m[11]=-1;
m[14]=2*zfar*znear/(znear-zfar);
```

Demonstatie: http://www.songho.ca/opengl/gl_projectionmatrix.html