10/20/2012

FACULTATEA DE AUTOMATICA SI CALCULATOARE

ELEMENTE DE GRAFICA PE CALCULATOR

Laborator 3

OpenGL & GLUT

Introducere in OpenGL

OpenGL este un API (Application Programming Interface) standard pentru grafica 3D. Inainte de aparitia sa fiecare producator de hardware avea propria sa biblioteca grafica. Nu-i greu de imaginat cat de costisitor era de produs o aplicatie grafica pentru mai multe platforme.

Ce poate face? In primul rand scuteste efortul de a transpune scena 3D pe ecran. Acest lucru se face automat.

- Implementeaza transformari: rotatie, translatie, scalare.
- Ofera o stiva de matrici in care putem pastra transformarile aplicate anterior.
- Poate desena diverse primitive cu diverse proprietati de material (obiectele pot sa para metalice, de plastic etc.).
- Texturile (imagini aplicate pe obiecte) pot face o scena simpla sa arate mult mai realist.
- Pot fi create obiecte transparente.
- OpenGL ofera diverse modele de iluminare si mai multe surse de lumina. Totusi iluminarea este destul de rudimentara pentru ca tine cont doar de pozitia obiectului fata de sursa de lumina nu si de celelalte obiecte din scena.

Insa facilitatile oferite de OpenGL micsoreaza efortul necesar pentru a produce umbrele. Acest document nu descrie totul. Scopul lui este sa ofere minimul necesar pentru a scrie o aplicatie OpenGL si nu descrie gestiunea ferestrelor intr-o anumita implementare.

Conventii de numire a functiilor

Desi la prima vedere numele functiilor par neintuitive, ele urmeaza o regula simpla. Majoritatea sunt de forma:

gl{nume functie}{numar de parametri}{sufix care arata tipul parametrilor}{v (daca parametrul este un vector} (.....)

Fiind independent de platform, OpenGL prevede cateva tipuri de date. Pentru a obtine o aplicatie portabila este recomandat sa le folositi pe acestea si nu altele de aceasi marime caracteristice sistemului de operare folosit:

Nume	Descriere	Sufix
GLbyte	intreg pe 8 biti	b
GLshort	intreg pe 16 biti	s
GLint, GLsizei	intreg pe 32 de biti	i
GLfloat, GLclampf	reprezentare in virgula mobila pe 32 de biti	f
GLdouble, GLclampd	reprezentare in virgula mobila pe 64 de biti	d
GLubyte, GLboolean	intreg fara semn pe 8 biti	ub
GLushort	intreg fara semn pe 16 biti	us
GLuint, GLenum,GLbitfield	intreg fara semn pe 32 de biti	ui

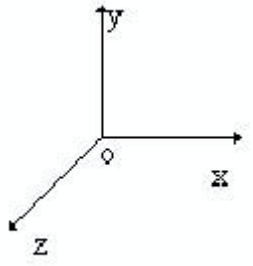
Cateva exemple:

- ***glVertex3d*** -primeste 3 parametri de tip GLdouble
- ***glColor3f*** -3 parametri de tip GLfloat

Totusi exista si cateva exceptii in care lipsesc numarul si tipul parametrilor (de exemplu ***glBegin***) sau care incep cu un alt prefix (***gluLookAt***).

Sistem de coordonate

OpenGL foloseste un sistem de coordonate dreapta. Avem ordinea axelor xyz. Un sistem de coordonate este dreapta, daca privind de-a lungul unei axe dinspre $+\infty$ spre origine, o rotatie in sens trigonometric va aduce o axa pozitiva, peste axa pozitiva urmatoare. De exemplu: (ox peste oy) sau (oy peste oz) sau (oz peste ox).

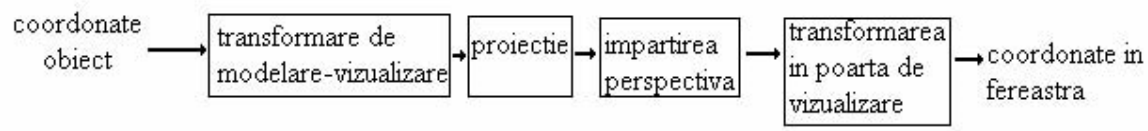


Transformari ale obiectelor 3D in OpenGL

Orice secventa de transformari, inclusiv proiectia se poate reprezenta printr-o matrice. OpenGL reprezinta intern punctele ca vectori coloana. Din acest motiv o secventa de doua transformari T1 apoi T2 aplicate punctului p produc rezultatul:

$$p' = T2\ T1\ p$$

Pentru a aduce scena 3D pe ecran OpenGL combina mai multe transformari:



OpenGL foloseste constantele: ***GL_MODELVIEW***, ***GL_PROJECTION*** pentru a identifica matricea de modelare-vizualizare, respectiv matricea de proiectie. Functiile OpenGL care reprezinta transformari modifica matricea curenta. Trebuie sa aveti grija ca inainte de a le aplica sa apelati ***glMatrixMode(constanta_care_identifica_matricea);*** pentru a va asigura ca transformarile vor produce efectul dorit.

Transformarea de modelare pozitioneaza un obiect undeva in lume. Puteti folosi rotatii, translatii, scalari. Transformarea de vizualizare pozitioneaza observatorul. Acelasi efect poate fi obtinut in ambele moduri, de aceea OpenGL le pastreaza ca o singura matrice: ***GL_MODELVIEW*.**

Pentru a stabili pozitia observatorului puteti folosi functia:

gluLookAt (GLdouble ex, GLdouble ey, GLdouble ez,GLdouble cx, GLdouble cy, GLdouble cz,GLdouble upx, GLdouble upy, GLdouble upz);

- (ex, ey, ez) reprezinta noua pozitie a observatorului
- (cx, cy, cz) va arata directia in care priveste acesta
- (upx, upy, upz) reprezinta directia sus din planul de vizualizare

Implicit observatorul e situat in origine, priveste in directia negativa a axei oz, iar directia sus a planului de vizualizare este directia pozitiva a axei oy. In acest moment matricea de modelare-vizualizare este matricea identitate. Cand apelati ***gluLookAt*** transformarea descrisa de aceasta este compusa cu cea deja existenta. De aceea daca o apelati de doua ori, al doilea apel s-ar putea sa produca alte rezultate decat cele pe care le asteptati. Inainte de ***gluLookAt*** puteti apela:

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

pentru a incarca matricea identitate in matricea de modelare-vizualizare.

Obiectele pot fi asezate/transformate 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);

Singura a carei semnificatie nu e evidenta este glRotatef. Aceasta functie roteste in sens trigonometric in jurul axei x,y,z cu unghiul unghi. O astfel de functie modifica matricea curenta, astfel incat transformarea se va aplica tuturor obiectelor pe care le vom desena de acum incolo, lasandu-le neschimbate pe cele deja desenate. Insa comportamentul lor este diferit de ceea ce ne-am astepta. Transformarile sunt aplicate obiectului in ordinea inversa apelurilor.

Pana acum am vazut cum stabilim pozitia si orientarea observatorului in raport cu obiectele din lume. Insa pentru a face scena sa apara pe ecran trebuie sa delimitam spatiul vizibil: 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 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:

void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble near, 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]

Transformarea in poarta de vizualizare se defineste folosind functia:

glViewport (GLint px, GLint py, GLint pz, GLsizei width, GLsizei height);

- 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.

Setarea culorii

Inainte de a face vreun desen trebuie sa setam o culoare de fond:

glClearColor(GLclampf R, GLclampf G, GLclampf B, GLclampf A);

A reprezinta opacitatea.Apoi vom sterge imaginea:

glClear(GL_COLOR_BUFFER_BIT);

Constanta data ca parametru arata ce buffer vrem sa stergem. OpenGL foloseste mai multe buffere in diverse scopuri. Putem sterge mai multe cu acelasi apel astfel:

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

In acest caz am sters imaginea si bufferul folosit pentru a determina care figuri sunt vizibile (folosit de algoritmul z-buffer). Acum putem seta o culoare care va fi folosita pentru urmatoarele desenari (pana o schimbam din nou):

glColor3f(GLfloat R, GLfloat G, GLfloat B);

Desenarea diverselor primitive

Exista mai multe tipuri de primitive pe care le putem desena folosind OpenGL. Pentru inceput insa nu le voi prezenta pe toate.

Putem desena ceva pe ecran folosind functia:

glVertex3f(GLfloat x, GLfloat y, GLfloat z;)

Pentru a avea vreun efect apelurile acestei functii trebuiesc incadrate intre apelurile ***glBegin(GLenum tip_primitiva)*** si ***glEnd( )*** . Tipul primitivei este identificat printr-una din constantele:

- ***GL_POINTS*** - deseneaza puncte
- ***GL_LINES*** - deseneaza segmente. Doua puncte definesc un segment, urmatoarele doua alt segment etc.
- ***GL_POLYGON*** - deseneaza un poligon
- ***GL_TRIANGLES*** - fiecare grup de 3 varfuri defineste un triunghi
- ***GL_QUADS*** - patrulatere
- ***GL_LINESTRIP*** - doar primul segment e definit prin doua puncte. Urmatoarele au un capat comun cu cel anterior
- ***GL_LINE_LOOP*** - Se deseneaza un segment si intre primul si ultimul punct
- ***GL_TRIANGLE_STRIP*** - banda de triunghiuri. Toate in afara de primul (care e definit prin 3 varfuri) au primele doua varfuri egale cu ultimele doua ale triunghiului anterior
- ***GL_TRIANGLE_FAN*** - toate triunghiurile au primul varf comun. Doua triunghiuri succesive au o latura comuna
- ***GL_QUAD_STRIP*** - asemanator cu triangle strip, dar sunt patrulatere

Este recomandat sa folositi ***GL_TRIANGLE_STRIP*** sau ***GL_TRIANGLE_FAN*** in loc de ***GL_TRIANGLES*** oricand este posibil si sa introduceti cat mai multe varfuri intre un singur apel ***glBegin*** si ***glEnd*** pentru ca placile grafice sunt optimizate pentru desenarea in acest mod si trebuie sa comunicam mai putine date placii.

Intre ***glBegin*** si ***glEnd*** puteti apela si ***glColor*** pentru a stabili culoarea varfurilor urmatoare.

Modul in care sunt afisate poligoanele se poate stabili (inainte de ***glBegin***) folosind:

glPolygonMode (GLenum face, GLenum mode);

- face poate fi ***GL_FRONT_AND_BACK*** sau ***GL_FRONT*** sau ***GL_BACK***. Aceste constante identifica fetele fata sau spate (in mod implicit fetele care sunt definite
- prin varfuri parcurse in sens trigonometric sunt considerate fete fata (front faces) mode poate fi ***GL_POINT***, ***GL_LINE*** sau ***GL_FILL***

Stive de matrici

Stiti deja ca putem stabili o transformare care sa fie aplicata tuturor varfurilor de acum incolo. Insa uneori avem nevoie sa definim o transformare nu fata de cea dinaintea ei, ci fata de una mai veche. De exemplu avem nevoie sa desenam o masina. Desenam masina. Aplicam o translatie si desenam prima roata. Ne este mai usor sa calculam pozitia celorlalte roti fata de

masina, nu fata de prima roata. OpenGL ne ofera o posibilitate de a face acest lucru. Exista o stiva de matrici. Cea din varful stivei este matricea curenta. In momentul in care apelam ***glPushMatrix( )*** matricea curenta este salvata in stiva. Putem sa modificam si sa folosim matricea curenta. Apoi cand vrem sa ne intoarcem la transformarea pastrata in stiva apelam ***glPopMatrix( ).*** Singurul lucru la care trebuie sa fim atenti este sa nu apelam ***glPopMatrix*** daca nu avem nici o matrice in stiva si sa nu depasim capacitatea stivei. Desi exista implementari care ofera stive foarte mari sau chiar nelimitate, standardul prevede o stiva de 32 de matrici de modelare-vizualizare si o stiva de 2 matrici de proiectie. Stiva pe care lucreaza ***glPushMatrix*** si ***glPopMatrix*** este stabilita cu ajutorul functiei ***glMatrixMode***.

Gestiunea ferestrelor folosind biblioteca glut

Pe langa standardul OpenGL, glut ofera si functii care usureaza realizarea unei interfete cu utilizatorul. Vom enumera cateva dintre ele (pentru mai multe informatii va puteti uita in indrumarul de laborator):

- ***glutInit(int* argc, char** argv)*** initializeaza variabilele interne si prelucreaza argumentele din linia de comanda. Trebuie apelata inaintea oricarei alte functii glut.
- ***glutInitDisplayMode(unsigned int mode)*** initializeaza modul de afisare. Mode specifica:
  - modelul de culoare folosit. Se recomanda ***GLUT_RGB***
  - folosirea unei ferestre cu buffer simplu sau dublu: ***GLUT_SINGLE*** sau ***GLUT_DOUBLE***
  - folosirea bufferului de adancime pentru algoritmul z-buffer: ***GLUT_DEPTH*** Pentru a obtine valoarea lui mode putem aplica | intre valorile care ne intereseaza.
- ***glutInitWindowPosition(int x, int y)*** pozitioneaza fereastra fata de coltul stanga sus al ecranului.
- ***glutInitWindowSize(int width, int height)*** stabileste dimensiunea ferestrei
- ***int glutCreateWindow(char* nume)*** creaza o fereastra cu un context OpenGL
- ***int glutDestroyWindow(int window)*** distruge fereastra window impreuna cu toate subferestrele ei.

Functii pentru controlul evenimentelor de intrare

- ***glutReshapeFunc(void (*f) (int width, int height))*** primeste ca parametru un pointer la o functie care trebuie apelata de fiecare data cand fereastra e redimensionata.
- ***glutKeyboardFunc(void (*f) (unsigned char key, int x, int y))*** primeste ca parametru o functie care trebuie apelata de fiecare data cand se apasa/elibereaza o tasta. Key este valoarea ASCII. x si y reprezinta pozitia mouse-ului la apasarea tastei.
- ***glutSpecialFunc(void (*f) (int key, int x, int y))*** primeste ca parametru o functie care se apeleaza de fiecare data cand se apasa/elibereaza o tasta speciala. Tastele speciale si valorile lor se gasesc la http://www.opengl.org/resources/libraries/glut/spec3/node54.html.
- ***glutMouseFunc(void (*f) (int buton, int stare, int x, int y))*** . *f va fi apelata la apasarea sau eliberarea unui buton de mouse. ***buton*** poate fi: ***GLUT_LEFT_BUTTON***, ***GLUT_MIDDLE_BUTTON***, ***GLUT_RIGHT_BUTTON***. Parametrul state poate fi***: GLUT_UP*** sau ***GLUT_DOWN***. Parametrii x si y reprezinta pozitia mouse-ului la aparitia evenimentului

- ***glutMotionFunc(void (*f) (int x, int y)*** . *f va fi apelata la deplasarea mouse-ului in timp ce un buton este apasat. Parametrii x si y reprezinta pozitia mouse-ului in momentul apasarii.

Functii pentru gestiunea meniurilor

Meniurile create cu GLUT sunt meniuri simple de tip pop-up.

- ***int glutCreateMenu(void (*f) (int value)*** primeste ca parametru un pointer la o functie care reprezinta functia de callback pentru meniul respectiv iar parametrul *value* reprezinta optiunea din meniu care a fost selectata. Functia va intoarce identificatorul meniului care este unic.
- ***void glutAddMenuEntry(char* name, int value)*** Adauga o noua optiune meniului curent. *name* reprezinta textul optiunii iar *value* va fi valoarea transmisa functiei callback asociata optiunii atunci cand aceasta este selectata.
- ***void glutAddSubMenu(char* name, int menu)*** Functia adauga un submeniu la meniul curent. *name* reprezinta textul noului submeniu iar *menu* reprezinta identificatorul meniului
- ***void glutAttachMenu(int button)*** Functia ataseaza meniul curent butonului mouseului specificat de *button*. Se pot folosi constantele ***GLUT_RIGHT_BUTTON***, ***GLUT_LEFT_BUTTON*** , etc.

Executia aplicatiei

- ***glutDisplayFunc(void (*f)(void))*** stabileste functia de afisare. Functia de afisare va fi apelata automat de fiecare data cand este necesara redesenarea continutului ferestrei. Putem cere explicit acest lucru apeland: ***glutPostRedisplay( )***. In cazul in care folosim un buffer dublu, la sfarsitul lui *f trebuie sa cerem schimbarea bufferelor prin ***glutSwapBuffers( )*** .
- ***glutIdleFunc(void (*f) (void))*** . Parametrul este o functie care va fi executata in perioadele in care nu exista evenimente in curs de tratare. NULL inseamna ca functia idle e dezactivata. Poate fi folosita pentru crearea unei animatii.
- ***glutMainLoop(void)*** este ultima functie care se apeleaza in main. Contine o bucla in care aplicatia asteapta evenimente.

Afisare de obiecte 3D predefinite

Biblioteca GLUT contine functii pentru afisarea urmatoarelor obiecte 3D:

- cub
  - o Desenare cub wireframe de latura size : ***void glutWireCube(GLdouble size);***
  - o Desenare cub solid de latura size : ***void glutSolidCube(GLdouble size);***
- sfera
  - o Desenare sfera wireframe de raza radius : ***void glutWireSphere(GLdouble radius, GLint slices, GLint stacks);***
  - o Desenare sfera solida de raza radius : ***void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);***
- tor
  - o Desenare tor wireframe : ***void glutWireTorus(GLdouble innerRadius, GLdouble outerRadius,GLint nsides, GLint rings);***

- Desenare tor solid : ***void glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius,GLint nsides, GLint rings);***

- icosaedru
  - Desenare icosaedru wireframe : ***void glutWireIcosahedron(void);***
  - Desenare icosaedru solid : ***void glutSolidIcosahedron(void);***
- octaedru
  - Desenare octaedru wireframe : ***void glutWireOctahedron(void);***
  - Desenare octaedru solid : ***void glutSolidOctahedron(void);***
- tetraedru
  - Desenare tetraedru wireframe : ***void glutWireTetrahedron(void);***
  - Desenare tertaedru solid : ***void glutSolidTetrahedron(void);***
- dodecaedru
  - Desenare dodecaedru wireframe : ***void glutWireDodecahedron(void);***
  - Desenare doedecaedru solid : ***void glutSolidDodecahedron(void);***
- con
  - Desenare con wireframe : ***void glutWireCone(GLdouble radius, GLdouble height, GLint slices, GLint stacks);***
  - Desenare con solid : ***void glutSolidCone(GLdouble radius, GLdouble height, GLint slices, GLint stacks);***
- ceainic
  - Desenare ceainic wireframe : ***void glutWireTeapot(GLdouble size);***
  - Desenare ceainic solid : ***void glutSolidTeapot(GLdouble size);***

Toate aceste obiecte sunt desenate centrate in originea sistemului de coordonate real.

La laborator se va folosi biblioteca freeglut (http://freeglut.sourceforge.net/), o alternativa Open Source pentru biblioteca GLUT.