1. Introducere în OpenGL

1.1 Suport teoretic

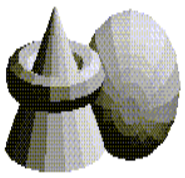
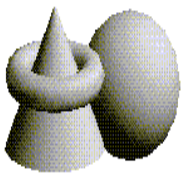
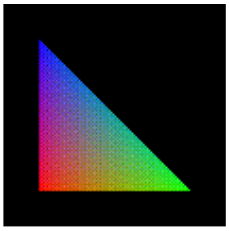
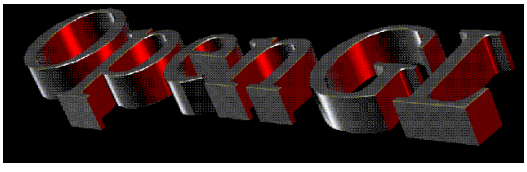
Una dintre cele mai impresionante demonstraţii care pot fi efectuate pe un calculator este vizualizarea aplicaţiilor grafice 3D. O dată cu introducerea standardului OpenGL ca bibliotecă de funcţii grafice 3D, aproape fiecare platformă are un mediu de creare a graficii profesionale.

OpenGL este o API (interfaţă pentru programarea aplicaţiilor grafice) care poate fi implementată atât hardware cât şi software. Varianta hardware presupune executarea funcţiilor OpenGL dintr-o aplicaţie de către acceleratorul grafic OpenGL existent pe placa video. Varianta hardware presupune executarea de către microprocessor a procedurilor corespunzătoare fiecărui apel OpenGL. Interfaţa OpenGL este scalabilă astfel că a fost îmbunătăţită continuu (versiunile 1.1, 1.2, 1.3, 1.4) sub conducerea colectivului ARB (Architecture Rview Board).

OpenGL are o arhitectură client-server astfel că se poate vizualiza o scenă grafică de pe un calculator care nu este computerul care rulează programul graficii respective. OpenGL este o interfaţă independentă de hardware. Independenţa este realizată prin neincluderea în ea a funcţiilor dependente de hardware. Aceasta presupune că sistemele de operare sunt cele care au fost adaptate pentru a suporta OpenGL. Biblioteca OpenGL este implementată pe diverse platforme hardware (Intel, Macintosh) şi pe diverse sisteme de operare (Windows, Unix), singura condiţie fiind de a rula din fereastră. Funcţiile OpenGL pot fi apelate din aproape orice mediu şi limbaj de programare (Visual C, Borland C, Java, Fortran, Basic, etc.). OpenGL stă la baza unor aplicaţii grafice mai complexe pentru modelarea scenelor 3D (spre exemplu 3D Studio Max).

OpenGL nu conţine comenzi de nivel înalt pentru descrierea obiectelor 3D. Asemenea comenzi ar permite specificarea unor forme relativ complicate cum ar fi: automobile, părţi ale corpului omenesc, avioane sau molecule. Cu OpenGL trebuie să construieşti modelul dorit folosind un mic set de primitive geometrice: puncte, linii şi poligoane. (O librărie sofisticată pentru asemenea şabloane este Open Inventor).

Exemple de scene simple modelate în OpenGL:

Pentru OpenGL s-au creat biblioteci suplimentare care permit crearea într-un mod simplu a ferestrelor şi tratarea evenimentelor de la tastatură (bibliotecile GLUT şi GLAUX). Aceste biblioteci nu permit realizarea unor aplicaţii cu interfeţe complicate (meniuri, butoane, cutii de dialog) dar au avantajul de a fi simplu de apelat şi de a permite afişarea scenelor OpenGL. În cadrul laboaratoarelor de OpenGL vom utiliza biblioteca GLAUX. Aplicaţia *pătrat.c* este o aplicaţie simplă pe care o putem considera un template pentru aplicaţiile care vor urma.

1.2 Desfăşurarea lucrării

Pentru exemplificare se foloseşte sursa *pătrat.c* – template pentru aplicaţiile GLAUX. Programul afişează un pătrat pe care îl translatează pe axa x la apăsarea săgeţilor stânga, dreapta.

1.2.1 Fazele obţinerii executabilului:

- se compilează sursa:

```
Build
      └→ Compile
```

- se adaugă bibliotecile `opengl32.lib, glu32.lib, glaux.lib` astfel:

```
Project
   └→ Settings
         └→ Link
               └→ librăriile
                  de mai sus
```

1.2.2. Comentarea codului sursă

Funcţia **main(...)** iniţializează şi deschide o fereastră pe ecran cu ajutorul următoarelor rutine GLAUX (rutinele GLAUX încep cu **aux**):

auxInitDisplayMode() - creează o fereastră în modelul de culoare RGB cu un singur (AUX_SINGLE) buffer de culoare (AUX_RGB) asociat (OpenGL conţine un buffer de cadru care la rândul său conţine buffer-ul (ele) de culoare, buffer-ul şablon, buffer-ul de acumulare, buffer-ul de adâncime). Deoarece dimensiunea buffer-elor depinde de dimensiunea ferestrei este normal ca la definirea ferestrei să se specifice ce conţine buffer-ul de cadru.

auxInitPosition() – setează poziţia ferestrei pe ecran şi dimensiunea ferestrei;

auxInitWindow() – deschide fereastra pe ecran, setează butonul ESC pentru ieşirea din program şi asociază un şir pentru titlul ferestrei;

Pentru tratarea evenimentelor de la tastatură sunt apelate următoarele funcţii GLAUX:

auxKeyFunc() – asociază tastelor AUX_LEFT (←) şi AUX_RIGHT(→) funcţiile de tip callback `MutaStanga, MutaDreapta` definite în codul sursă. De asemenea înregistrează în sistemul de operare aceste funcţii. Sistemul de operare le va apela ori de câte ori sunt apăsate tastele respective.

auxReshapeFunc() – înregistrează în sistemul de operare funcţia de tip callback myReshape care indică ce acţiuni să aibă loc în cazul în care fereastra este redimensionată. Funcţia apelată – myReshape- va fi discutată ulterior pentru o mai bună înţelegere.

auxMainLoop() - înregistrează în sistemul de operare funcţia `display` care va fi apelată ori de câte ori se va redesena conţinutul ferestrei şi apoi o apelează.

Funcţia `display` (folosită pentru desenarea scenei) efectuează următoarele: şterge bufferul de culoare cu ajutorul rutinei `glClear(GL_COLOR_BUFFER_BIT).`

Funcţia `glLoadIdentity ()`- iniţializează matricea curentă de modelare-vizualizare care este setată ca fiind matricea identitate. Acest pas este necesar deoarece toate comenzile de transformare multiplică matricea curentă cu o anumită matrice şi apoi rezultatele se trec în matricea curentă. Dacă nu este iniţializată matricea de modelare-vizualizare cu matricea identitate se riscă pornirea cu o matrice de modelare-vizualizare al cărei conţinut nu este controlat.

În continuare, se definesc culorile şi coordonatele vârfurilor pătratului cu ajutorul rutinelor `glColor3f()`, `glVertex2f()` cuprinse între funcţiile `glBegin()` şi `glEnd()`.

Funcţia `glFlush()` are efect atunci când există o reţea, forţează clientul să trimită pachetul în reţea. Deasemenea, rutina nu aşteaptă ca desenul să fie complet, ci forţează execuţia funcţiilor OpenGL anterioare.

Aplicaţia 1. Să se adauge aplicaţiei translaţia SUS-JOS a pătratului cu ajutorul săgeţilor sus-jos ale tastaturii.

Aplicaţia 2. Să se rotească obiectul folosind butoanele mouse-ului. (Se va utiliza rutina `glRotatef(alfa, 0, 0, 1)`, unde „alfa” specifică unghiul de rotaţie, iar „1” indică faptul că rotaţia se face în jurul axei z.)
Funcţia GLAUX care înregistrează în sistemul de operare funcţia apelată la acţionarea mouse-ului va fi:

```
auxMouseFunc (AUX_LEFTBUTTON, AUX_MOUSEDOWN,
rot_z_up);        // înregistrează funcţia callback rot_z_up
auxMouseFunc (AUX_RIGHTBUTTON, AUX_MOUSEDOWN,
rot_z_down);      // înregistrează funcţia callback rot_z_down
```

Se vor programa corespuzător funcţiile `rot_z_up` şi `rot_z_down`.

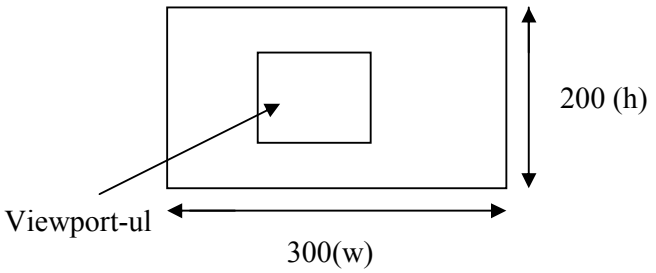
1.2.3. Funcţia **myReshape()**

Funcţia **myReshape()** are un rol foarte important în cazul în care fereastra este redimensionată. Ea apare în aplicaţie în două forme, pentru transformarea anizotropică (comentată), respectiv transformarea izotropică. În această funcţie se defineşte volumul de vizualizare şi viewport-ul. Deasemenea se stabilesc dimensiunile volumului de vizualizare în unităţi logice. Unităţile folosite la desenare vor fi unităţi logice. Dacă funcţia **myReshape()** lipseşte din aplicaţie se va lucra implicit în pixeli. Transformarea de viewport controlează conversia coordonatelor logice în coordonate ale ecranului (pixeli). **Transformarea izotropică** presupune că maparea de la unităţi logice la unităţi de dispozitiv se realizează astfel încât dimensiunea în pixeli a unităţilor logice este egală pe cele două axe (x şi y). În cazul transformării anizotropice dimensiunea în pixeli a unităţilor logice este diferită pe cele două axe. Acesta este motivul pentru care un pătrat, spre exemplu, care are laturile egale în unităţi logice va apare pe ecran ca dreptunghi.

Funcţia **myReshape()** din această aplicaţie conţine următoarele rutine:

- **glViewport(0, 0, w, h) – desemnează dimensiunea viewportului (**indică originea, înălţimea şi lăţimea (în pixeli) a zonei din fereastră în care se afişează scena).

Exemplu:

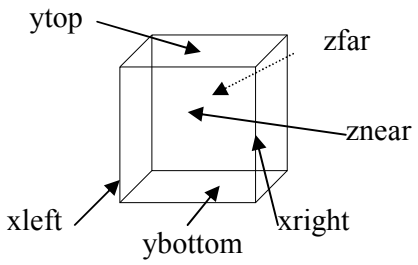


Dacă fereastra îşi măreşte dimensiunea, viewportul trebuie să-şi schimbe dimensiunea în mod corespunzător. În aplicaţia din laborator viewport-ul ocupă întreaga fereastră.

- **glMatrixMode(GL_PROJECTION)** – se comută pe matricea curentă de proiecţie;

- **glMatrixMode(GL_MODELVIEW) –** indică faptul că transformările următoare vor afecta matricea curentă de modelare-vizualizare şi nu matricea de proiecţie;

- **glOrtho() –** delimitează volumul de vizualizare corespunzător proiecţiei paralele ortografice, corespunzător tranformării anizotropice, respectiv izotropice. Parametrii funcţiei sunt `xleft`, `xright`, `ybottom`, `ytop`, `znear`, `zfar`.



Funcţia **myReshape()** comentată obţine o transformare viewport anizotropică, cealaltă funcţie **myReshape()** obţine transformarea izotropică.
Pentru a obţine transformarea izotropică se compară `w` cu `h`.
Dacă w>h, stabilim ca bază înălţimea şi stabilim numărul unităţilor logice pe orizontală în funcţie de dimensiunea unităţii logice pe verticală.

$$1 \text{ UL (pe înălţime)} = \frac{h}{320} \text{ pixeli;}$$

Numărul unităţilor logice pe lăţime = $w/(\frac{h}{320}) = 320 * \frac{w}{h}$ unităţi logice.

Aplicaţia 3. Să se ruleze aplicaţia cu fiecare din cele două funcţii `myReshape()` precum şi fără funcţia `myReshape()`.

Aplicaţia 4. Să se fixeze un vârf al patrulaterului în punctul de coordonate (0, 0). Să se rescrie funcţia myReshape() astfel încât punctul de coordonate logice (0, 0) să nu se mai afle în centrul ferestrei ci în colţul stânga-jos al ferestrei. Dimensiunea ferestrei să fie tot de 320 unităţi logice iar transformarea să fie tot izotropică.

De reţinut:

Rolul funcţiei `myReshape();`

1. Stabilirea dimensiunii volumului de vizualizare;
2. Stabilirea tipului transformării: – izotropică
   – anizotropică;
3. Stabilirea dimensiunii unităţilor logice;
4. Stabilirea poziţiei originii sistemului de coordonate logic.