Curs 2

OpenGL

- OpenGL este o specificație standard care definește o aplicatie cross-platform API
- a fost dezvoltat de Silicon Graphics Inc. (SGI) in 1992 de Mark Segalsi si Kurt Akeley (bazat pe IrisGL)

versiuni OpenGL -> 1.0, 1.1, 1.2, 1.2.1, 1.3, 1.4, 1.5, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3, 4.0, 4.1, 4.2....4.5 , 4.6

OpenGL este condus de un consortiu tehnologic non-profit, Khronos Grup.

https://vulkan-tutorial.com/Drawing_a_triangle/Presentation/Window_surface

OpenGL 4.6 integrează funcționalitatea numeroaselor extensii ARB și EXT create de membrii Khronos AMD, Intel și NVIDIA în nucleu, inclusiv capacitatea de a ingera umbrere SPIR-V ™. SPIR-V este un limbaj intermediar standard definit de Khronos pentru calcul paralel și grafică, care permite creatorilor de conținut să își simplifice canalele de autorizare și gestionare a shader-urilor oferind în același timp o flexibilitate semnificativă a limbajului de umbrire a sursei. OpenGL 4.6 adaugă suport pentru ingestia de umbrere SPIR-V la specificațiile de bază, garantând că umbrele SPIR-V vor fi susținute pe scară largă de implementările OpenGL. OpenGL 4.6 adaugă funcționalitatea acestor extensii ARB la specificațiile de bază ale OpenGL

https://community.khronos.org/t/official-feedback-on-opengl-4-6-thread/76018

- OpenGl este un API (Aplication Programming Interface - Interfaţa de programe de aplicaţii) foarte utilizat pentru programarea componentelor grafice 2D si 3D ale programelor de calculator.

La elaborarea specificatiei OpenGL au stat la baza urmatoarele principii:

- ✓independenta fata de platforma *hardware*
- ✓independenta fata de sistemul de operare.

https://www.opengl.org//

OpenGL vs Vulkan.

OpenGL este un API multiplataforma în care API se referă la interfața de programare a aplicației și se concentrează pe redarea 2D, precum și a graficelor vectoriale 3D cu rezultate eficiente. Pentru redarea hardware accelerată, interacționează cu unitatea de procesare grafică (GPU).

Vulkan este, de asemenea, software multiplataforma care funcționează ca grafică 3D, precum și software de calcul nu numai acest lucru, ci se ocupă și cu programarea jocurilor video și a elementelor multimedia. Ambele programe au, de asemenea, cele mai multe funcții, chiar dacă sunt diferite între ele.

Vulkan este noua generație API deschisă standard pentru acces de înaltă eficiență la grafică și calcul pe GPU-uri moderne. Acest design bazat, denumit anterior Inițiativa OpenGL de generație următoare, oferă aplicațiilor control direct asupra accelerației GPU pentru performanță maximă și predictibilitate.

- OpenGL nu este un mediu de programare; nu este o interfata grafica; nu este orientat obiect.

Biblioteca OpenGl contine:

- primitive geometrice → puncte, linii si poligoane;
- primitive rastru;
- mod de lucru RGB/A;
- modelarea si vizualizarea transformarilor de geometrie;
- eliminarea muchiilor ascunse (invizibile);
- maparea texturilor - aplicarea de texturi 2D pe obiect 3D;
- efecte speciale - fum, ceata si dizolvari;
- operatii cu pixeli;
- lucrul cu portiuni de ecran;
- lucrul simultan cu doua randuri de imagini, una care este afisata si una care este pregatita pentru afisare.

Biblioteca grafică OpenGL conţine funcţii de redare a primitivelor geometrice independente de sistemul de operare, de orice sistem Windows sau de platforma hardware. Ea nu conţine nici o funcţie pentru a crea sau administra ferestre de afişare pe display (windows) şi nici funcţii de citire a evenimentelor de intrare (mouse sau tastatură).

Pentru crearea şi administarea ferestrelor de afişare şi a evenimentelor de intrare se pot aborda mai multe soluţii: utilizarea directă a interfeţei Windows (Win32 API), folosirea compilatoarelor sub Windows, care conţin funcţii de acces la ferestre şi evenimente sau folosirea altor instrumente (utilitare) care înglobează interfaţa OpenGL în sistemul de operare respectiv.

Un astfel de utilitar este GLUT, care se compilează şi instalează pentru fiecare tip de sistem Windows. Header-ul glut.h trebuie să fie inclus în fişierele aplicaţiei, iar biblioteca glut.lib trebuie legată (linkată) cu programul de aplicaţie. Pentru Visual Studio 2019 exista situatie speciala (vezi laborator).

În GLUT sunt predefinite câteva tipuri de funcţii callback; acestea sunt scrise în aplicaţie şi pointerii lor sunt transmişi la înregistrare sistemului Windows, care le apelează (prin intermediul pointerul primit) în momentele necesare ale execuţiei.

Funcţii de control ale ferestrei de afişare

```
void glutInit(int* argc, char** argv);
```

Această funcţie iniţializează GLUT folosind argumentele din linia de comandă; ea trebuie să fie apelată înaintea oricăror alte funcţii GLUT sau OpenGL.

void glutInitDisplayMode(unsigned int mode);

Specifică caracteristicile de afişare a culorilor şi a bufferului de adâncime şi numărul de buffere de imagine. Parametrul mode se obţine prin SAU logic între valorile fiecărei opţiuni.

Exemplu

- glutInitDisplayMode(GLUT_SINGLE | GLUT_DOUBLE | GLUT_RGB | GLUT_RGBA)

iniţializează afişarea culorilor în modul RGB, cu două buffere → de imagine şi buffer de adâncime. Alte valori ale parametrului mode sunt: GLUT_SINGLE (un singur buffer de imagine), GLUT_RGBA (modelul RGBA al culorii), GLUT_STENCIL (validare buffer şablon) si GLUT_ACCUM (validare buffer de acumulare).

void glutInitWindowPosition(int x, int y);

Specifică locaţia iniţială pe ecran a colţului stânga sus al ferestrei de afişare.

void glutInitWindowSize(int width, int heigth);

Defineşte dimensiunea iniţială a ferestrei de afişare în număr de pixeli pe lăţime (width) şi înălţime (heigth) .

int glutCreateWindow(char* string);

Creează fereastra în care se afişează contextul de redare OpenGL şi returnează identificatorul ferestrei.

Observatie:

Această fereastră este afişată numai după apelul funcţiei glutMainLoop();

Funcţii callback.

Funcţiile callback se definesc în program şi se înregistrează în sistem prin intermediul unor funcţii speciale de tip GLUT.

Ele sunt apelate de sistemul de operare atunci când este necesar, în funcţie de evenimentele apărute.

glutDisplayFunc(void(*Functie_utilizator)(void));

Această funcţie înregistrează funcţia Functie_utilizator în care se calculează şi se afişează imaginea. Argumentul funcţiei este un pointer la o funcţie fără argumente care nu returnează nici o valoare.

Atentie: Funcţia Display (a aplicaţiei) este apelată oridecâte ori este necesară desenarea ferestrei: la iniţializare, la modificarea dimensiunilor ferestrei.

glutReshapeFunc(void(*Reshape)(int w, int h));

Înregistrează funcţia callback Reshape care este apelată ori de câte ori se modifică dimensiunea ferestrei de afişare.

Argumentul este un pointer la funcţia cu numele Reshape cu două argumente de tip întreg şi care nu returnează nici o valoare. În această funcţie, programul de aplicaţie trebuie să refacă transformarea fereastră-poartă, dat fiind că fereastra de afişare şi-a modificat dimensiunile.

glutKeyboardFunc(void(*Keyboard)(unsigned int key, int x, int y);

Înregistrează funcţia callback Keyboard care este apelată atunci când se acţionează o tastă. Parametrul key este codul tastei, iar x şi y sunt coordonatele (relativ la fereastra de afişare) a mouse-ului în momentul acţionării tastei.

glutMouseFunc(void(*MouseFunc)(unsigned int button, int state, int x, int y);

Înregistrează funcţia callback MouseFunc care este apelată atunci când este apăsat sau eliberat un buton al mouse-ului.

Parametrul button este codul butonului si poate avea una din constantele GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON sau GLUT_RIGHT_BUTTON) . Parametrul state indică apăsarea (GLUT_DOWN) sau eliberarea (GLUT_UP) al unui buton al mouse-ului. Parametrii x şi y sunt coordonatele relativ la fereastra de afişare a mouse-ului în momentul evenimentului.

Exemplu:

```
void mouse(int buton, int stare, int x, int y)
{
        switch(buton)
        {
        case GLUT_LEFT_BUTTON:
                if(stare == GLUT_DOWN)
                        glutIdleFunc(animatieDisplay);
                break;
        case GLUT_RIGHT_BUTTON:
                if(stare== GLUT_DOWN)
                        glutIdleFunc(NULL);
                break;
        }
}
```

Poarta de afişare OpenGL

Poarta de afişare mai este numită context de redare (*rendering context*), şi este asociată unei ferestre din sistemul Windws. Dacă se programează folosind biblioteca GLUT, corelarea dintre fereastra Windows şi portul OpenGL este asigurată de funcţii ale acestei biblioteci. Dacă nu se foloseşte GLUT, atunci funcţiile bibliotecilor de extensie XGL, WGL sau PGL permit atribuirea unui context de afişare Windows pentru contextul de redare OpenGL şi accesul la contextul de afişare Windows (*device context*). Funcţia OpenGL care defineşte transformarea fereastră-poartă este:

```
void glViewport(GLint x, GLint y,   GLsizei width, GLsizei height);
```

unde x şi y specifică poziţia colţului stânga-jos al dreptunghiului porţii în fereastra de afişare (window) şi au valorile implicite 0, 0. Parametrii width şi heigth specifică lăţimea, respectiv înălţimea, porţii de afişare, dată ca număr de pixeli.

Exemplu: glViewport (0, 0, (GLsizei) w, (GLsizei) h);

ATENTIE: transformarea ferestră-poartă este componentă a transformării din sistemul de referinţă normalizat în sistemul de referinţă ecran 3D

Un pixel este reprezentat în OpenGL printr-un descriptor care defineşte mai mulţi parametri:

- numărul de biţi/pixel pentru memorarea culorii
- numărul de biţi/pixel pentru memorarea adâncimii
- numărul de buffere de imagine

Operaţiile de bază OpenGL

OpenGL desenează primitive geometrice (puncte, linii şi poligoane) în diferite moduri selectabile. Primitivele sunt definite printr-un grup de unul sau mai multe vârfuri (*vertices*). Un vârf defineşte un punct, capătul unei linii sau vârful unui poligon. Fiecare vârf are asociat un set de date:

- coordonate,
- culoare,
- coordonate de textură.

OpenGL execută secvenţa de operaţii grafice asupra fiecărei primitive geometrice, definită printr-o mulţime de vârfuri şi tipul acesteia. Coordonatele unui vârf sunt transmise către OpenGL prin apelul unei funcţii glVertex#(coordonate). Exemple:

- void glVertex2d(GLdouble x, GLdouble y);
- void glVertex3d(GLdouble x, GLdouble y, GLdouble z);
- void glVertex3f(GLfloat x, GLfloat y, GLfloat z);

- Vârfurile pot fi specificate în plan si în spaţiu folosind apelul funcţiei corespunzătoare.
- O primitivă geometrică se defineşte printr-o mulţime de vârfuri (care dau descrierea geometrică a primitivei) şi printr-unul din tipurile prestabilite, care indică topologia, adică modul în care sunt conectate vârfurile între ele.
- Mulţimea de vârfuri este delimitată între funcţiile glBegin(argument) şi glEnd(). Aceeaşi mulţime de vârfuri (v_0, v_1, v_2,….v_{n-1}) poate fi tratată ca puncte izolate, linii, poligon, etc, în funcţie de tipul primitivei, care este transmis ca argument al funcţiei glBegin():

void glBegin(GLenum mode);

Valorile argumentului mode şi, deci, tipurile de primitive acceptate de OpenGL sunt prezentate în tabelul 1.

void **glFlush**(void);

În cazul în care nu există nici un client şi toate comenzile sunt executate pe server functia **glFlush** nu va avea nici un efect. Pentru ca un program să funcţioneze corect atât în reţea cât şi pe o singură maşina, se va include apelul functieie **glFlush** la sfârşitul fiecărei scene. Funcţia **glFlush** nu aşteaptă terminarea desenării, ci doar forţează începerea execuţiei desenării.

Desenarea primitivelor in OpenGL

- Exista mai multe tipuri de primitive pe care le putem desena folosind OpenGL. 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(),** unde tipul primitivei este identificat printr-una din constantele:
- **GL_POINTS** - pentru desenare puncte
- **GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP** - pentru segmente. **GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN** -triunghiuri
- **GL_QUADS, GL_QUAD_STRIP** - pentru patrulatere
- **GL_POLYGON** - pentru poligoane

Argument	Primitivă geometrică
GL_POINTS	Desenează n puncte
GL_LINES	Desenează segmentele de dreaptă izolate (v_0,v_1), (v_2,v_3),… ş.a.m.d. Dacă n este impar ultimul vârf este ignorat
GL_LINE_STRIP	Desenează linia poligonală formată din segmentele (v_0,v_1), (v_1,v_2), .. . …(v_{n-2}, v_{n-1})
GL_LINE_LOOP	La fel ca primitiva GL_LINE_STRIP, dar se mai desenează segmentul (v_n, v_0) care închide o buclă.
GL_TRIANGLES	Desenează o serie de triunghiuri folosind vârfurile (v_0,v_1,v_2), (v_3,v_4,v_5), ş.a.m.d. Dacă n nu este multiplu de 3, atunci ultimele 1 sau 2 vârfuri sunt ignorate.
GL_TRIANGLE_STRIP	Desenează o serie de triunghiuri folosind vârfurile (v_0,v_1,v_2), (v_2,v_1,v_3),..ş.a.m.d. Ordinea este aleasă astfel ca triunghiurile să aibă aceeaşi orientare, deci să poată forma o suprafaţă închisă.
GL_TRIANGLE_FAN	Desenează triunghiurile (v_0,v_1,v_2), (v_0,v_2,v_3), ş.a.m.d.
GL_QUADS	Desenează o serie de patrulatere (v_0,v_1,v_2,v_3), (v_4,v_5,v_6,v_7), ş.a.m.d. Dacă n nu este multiplu de 4, ultimele 1, 2 sau 3 vârfuri sunt ignorate.
GL_QUADS_STRIP	Desenează o serie de patrulatere (v_0,v_1,v_3,v_2), (v_3,v_2,v_5,v_4), ş.a.m.d Dacă n < 4, nu se desenază nimic. Dacă n este impar, ultimul vârf este ignorat.
GL_POLYGON	Desenează un poligon cu n vârfuri, (v_0,v_1,…v_{n-1}). Dacă poligonul nu este convex, rezultatul este

- void **glPointSize**(GLfloat size);

Funcţia setează lăţimea în pixeli a punctelor ce vor fi afişate. Parametrul *size* reprezintă dimensiunea punctului exprimată în pixeli ecran. Ea trebuie să fie mai mare ca 0.0, iar valoarea sa implicită este 1.0.

- void **glLineWidth**(GLfloat width);

Funcţia setează lăţimea în pixeli a liniilor ce vor fi afişate; *width* trebuie să fie mai mare ca 0.0, iar valoarea implicită este 1.0.

Reprezentarea culorilor în OpenGL

În biblioteca OpenGL sunt definite două modele de culori: modelul de culori RGBA şi modelul de culori indexate. În modelul RGBA sunt memorate componentele de culoare R, G, B şi transparenţa A pentru fiecare primitivă geometrică sau pixel al imaginii.

```
void glColor3f(GLfloat red, GLfloat green, GLfloat blue);
void glColor4d(GLdouble red, GLdouble green, GLdouble blue, GLdouble alpha);
```

Fiecare componentă a culorii curente este memorată ca un număr în virgulă flotantă cuprins în intervalul [0,1].

Valorile argumentelor de tip întreg fără semn (unsigned int, unsigned char, etc.) sunt convertite liniar în numere în virgulă flotantă, astfel încât valoarea 0 este transformată în 0.0, iar valoarea maximă reprezentabilă este transformată în 1.0 (intensitate maximă).

```
#include "stdafx.h"
#include <gl/freeglut.h>
//puncte.cpp
void init()
{
glClearColor(0.0, 0.0, 0.0, 0.0);
//glPointSize(40.0);
}
void display()
{
glColor3f(1.0, 1.0, 0.0);
glBegin(GL_POLYGON);//initializare desen poligon
glVertex2f(0.0, 00.0); //stabilire coordonate triunghi
glVertex2f(200.0, 200.0);
glVertex2f(0.0, 200.0);
glEnd();
glFlush();
glPointSize(40.0);
glColor3f(1.0, 0.0, 0.0);
glBegin(GL_POINTS);
glVertex2i(300, 300);glVertex2i(20, 20);
glEnd();   glFlush();
}
```

```
void reshape(int w, int h)//functia redesenare
{
glViewport(0, 0, (GLsizei)w, (GLsizei)h);//stabilirea viewportului la dimensiunea
ferestrei
glMatrixMode(GL_PROJECTION);//specificare matrice modificabila la valoare
argumentului de modificare
glLoadIdentity();//initializarea sistemului de coordonate
gluOrtho2D(0.0, (GLdouble)w, 0.0, (GLdouble)h);//stabileste volumul de vedere
folosind o proiectie ortografica
}
void main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(400, 400);
glutInitWindowPosition(150, 150);
glutCreateWindow("puncte");
init();
glutDisplayFunc(display);
 glutReshapeFunc(reshape);
glutMainLoop();
}
```

```
glBegin(GL_POINTS);
glPointSize(40.0);
for (int dist = 0, i = 1; i <= 3; i++)
{
dist += 40;
glVertex2i(40 * i + dist, 40);//y
}
```

```
glBegin(GL_LINES);
for(int i=0;i<=4;++i)
{
glVertex3f(0,0,0); //x,y,z
glVertex3f(1-i/4.0, i/4.0, 0);//1,0,0 ; i=0;
//i=4;//0,1,0
}
glEnd();
```