Programfejlesztés az OpenGL segítségével (1. rész)

A 3D programozás alapjai

Az OpenGL napjaink egyik legsokoldalúbb fejlesztői könyvtára melynek segítségével a komplex tervezőrendszerektől, a játékok megjelenítéséig bármit megvalósíthatunk ami csak kapcsolatban áll a számítógépes grafikával és a harmadik dimenzióval, ráadásul platformfüggetlen.

© Kiskapu Kft. Minden jog fenntartva

Mielőtt belevágnék az ***OpenGL*** ismertetésébe engedjék meg, hogy kicsit elkalandozzak. 1999-ben találkoztam először a ***Linuxszal*** az egyik évfolyam- társam jóvoltából. Nagyon megtetszett a rendszer, de rettenetesen zavart, hogy a felesleges időmet csak ***PacMan*** színvonalú játékokkal tudom elütni. Félreértés ne essék a ***PacMan*** remek játék, de windowsos ismerőseim akkor már a legújabb játékokkal játszottak: ***Kingpin***, ***Planescape***: ***Torment***, stb.
Elkezdtem nyomozni, hogy egyáltalán van-e lehetőségem arra, hogy használható játékot telepítsek a gépemre. Az eredmény nagyon elkeserítő volt: a videokártyák támogatottsága egy ***C-64***-gyel vetekedett és minden ezredik játékot portoltak nagy kegyesen a játékfejlesztők ***Linuxra***.
Mostanra már lényegesen javult ez az állapot: az ***nVidia*** és ***ATI*** kártyák meghajtó programjai rohamos fejlődésnek indultak és egyre több vállalkozó szellemű fiatalember vagy cég akad akik a játékokat átültetik a mi rendszerünkre is. Nem titkolt szándékom ezzel a cikkel, hogy egy kicsit megpróbáljam „felkavarni" az állóvizet kicsiny hazánkban. Nem olyan nagy ördöngösség ***OpenGL***-t programozni és ha az ember nem rögtön a ***Quake 6*** megírásával kezdené a dolgot (ami jó eséllyel csak kedvét szegi) akkor érdekes, izgalmas játékok születhetnének.
A ➲ ***www.happypenguin.org*** oldalon van néhány kiemelkedő színvonalú munka amiket az emberek szabadidejükben fejlesztettek vagy fejlesztenek még mindig.

Mi is az OpenGL?

Az ***OpenGL***-t a ***Silicon Graphics*** cég fejlesztette ki a saját munkaállomásai számára. A cél egy erősen optimalizált 3 dimenziós műveletek elvégzésére alkalmas függvénykönyvtár volt a cég nagy teljesítményű grafikai eszközei számára amelyeket modellezésre és különböző szimulációk lefuttatására használtak. Nem sokkal később más cégek is érdeklődni kezdtek az új – ekkor még ***IrisGL*** névre hallgató – szabvány iránt. 1992. július 1-én mutatta be az ***SGI*** az átdefiniált ***IrisGL*** szabványt ami az ***OpenGL*** nevet kapta. Pár nappal később a legelső Microsoft fejlesztői konferencián demonstrálták a rendszer működését és innentől kezdve az ***SGI*** és a ***Microsoft*** együtt fejlesztette tovább az ***OpenGL***-t.
Maga az ***OpenGL*** egy sok száz eljárásból álló függvénykönyvtár mely lehetővé teszi 2 és 3 dimenziós grafikai objektumok definiálását és módosítását. Egy fontos félreértést még az elején tisztázni szeretnék: az ***OpenGL*** nem rendelkezik saját ablakozó rendszerrel. Ezt a programozónak kell megoldani. Ebből adódik, hogy az ***OpenGL*** nem képes kezelni a beviteli eszközöket sem. Későbbiekben egy teljes cikk sorozatot szeretnék szentelni az ***OpenGL-SDL*** párosításnak ami a legtöbb linuxos játék magját képezi és igen hatékonyan oldhatunk meg benne különböző multimédiás feladatokat.

De én nem tudok X-et programozni!

Ahhoz, hogy hatékonyan tudjunk ***OpenGL***-es alkalmazásokat fejleszteni, nem kell értenünk az X programozásához. Ilyenkor hívhatjuk segítségül a ***GLUT*** névre hallgató függvénykönyvtárat ami az ***OpenGL*** ablak- és eszközkezelési hiányosságainak a kiküszöbölésére készült. Erre még egy kicsit később visszatérünk, most csak azért említettem meg, hogy azok akiknek a fenti pár bekezdés elvette volna a kedvét – mert úgy gondolták, hogy mégsem olyan egyszerű a dolog mint említettem – azok ne csapjanak fel virágárusnak, hanem olvassanak inkább tovább.

Matematikai gyorstalpaló

A 3D grafika – sajnálatos vagy nem sajnálatos ezt mindenki döntse el maga – velejárója a matematika középiskolásnál kicsit magasabb szintű ismerete. Nem kell megijedni, itt csak néhány igen specifikus dologra gondolok. Érdemes egy kicsit érteni a vektorokhoz, a koordináta rendszerekhez és a mátrixokhoz. A most következő pár bekezdést szeretném ezeknek a matematikai „finomságoknak" és a hozzájuk kapcsolódó ***OpenGL*** objektumoknak szentelni.

1. ábra Ezen a képen egy objektum alapvető felépítését láthatjuk (Planet Quake)

Az OpenGL alap objektumai

Mielőtt nekiesnénk a szorzásnak-osztásnak, nézzük meg, hogyan is épülnek fel a testek. Minden modell ami a számítógépes grafikában megjelenik pontokból, a pontokat összekötő élekből, és az élek által alkotott felületekből épül fel. Ahhoz, hogy bármi megjelenhessen a képernyőn ezekre szükség van. Ezekből az alapelemekből építkezik az ***OpenGL*** is. A pontokat vertex-nek, az éleket face-nek a felületeket pedig poligonnak nevezzük.
Hogyan kerül át egy modell a „modellező asztalról" a játékba? Egyszerűen! A Modellezők valamilyen program felhasználásával (***Maya***, ***Blender***, ***3D Studio Max***) elkészítik az objektumot és exportálják egy megfelelő szerkezetű fájlba ami leírja, hogy a pontok, az élek és a felületek, hol és hogyan helyezkednek el a térben. Erre a legjobb példa talán az egyik legelterjedtebb szöveges felépítésű 3D-s fájl az ***ASC***, vagy a nagy testvére az ***ASE***. Ezeket a fájlokat „szabad szemmel" is képesek vagyunk olvasni és nagyon jó kiindulási pontot jelentenek egy saját fájl hierarchia megtervezésénél.
Most lássuk, mi kell ahhoz, hogy térben tudjunk pontokat definiálni.

Koordinátarendszerek

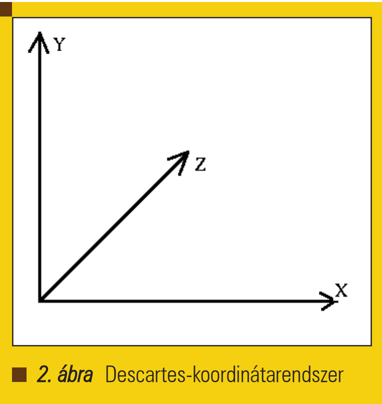
Az ***OpenGL*** az úgynevezett ***Descartes*** koordináta-rendszert használja a primitívek leírására.
Ez nem más mint a „hétköznapi" koordináta rendszer amelyet három egymásra merőleges vektor határoz meg (*2. ábra*).
Minden pont (vertex) egy rendezett számpárral van definiálva. Például ***(12,4,3)***. Ez azt jelenti, hogy van egy pont aminek az X koordinátája 12, az Y koordinátája 4, a Z koordinátája 3. Ezekkel a koordinátákkal egyértelműen meghatározható a 3 dimenziós tér egy pontja. Ha csak 2 dimenziós síkon szeretnénk mozogni akkor értelemszerűen a számunkra felesleges koordinátát 0 -ra kell állítani, vagy el kell hagyni.

Vektor és normálvektor

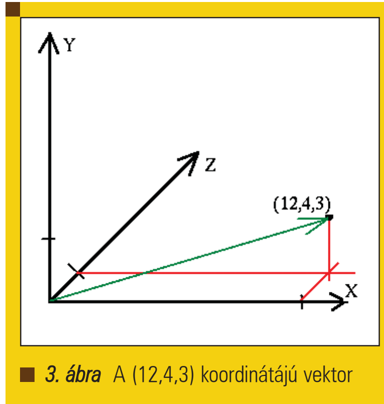
A vektor egy szakasz amely két tulajdonsággal rendelkezik: van iránya és hossza. Ez nekünk miért fontos? Azért, mert ha feltételezzük, hogy térben minden vektor egy középpontból (az origóból) indul ki akkor máris láthatjuk, hogy vektorok segítségével definiálhatunk pontokat a térben. Az ***OpenGL*** pontosan erre építkezik. A térnek van egy fix pontja ami az „univerzumunk" középpontját fogja jelenteni. Ehhez a ponthoz viszonyítunk mindent. Ennek a pontnak a koordinátái ***OpenGL***-ben ***(0,0,0)***. A *3. ábrán* a ***(12,4,3)*** végpontú vektort próbáltam ábrázolni.
A vektorok ismeretének fényében már igazán nem lehet problémánk a normálvektorok megértésével.
A normálvektor egy egységnyi hosszúságú (azaz a hossza 1) egy felületre merőlegesen mutató vektor. A feladata a felület irányának meghatározása. Egy normálvektor mindig merőleges azzal a felülettel amihez tartozik. Erre leginkább a fények, árnyalások és a megvilágítás létrehozásakor lesz szükségünk.
A *4. ábrán* megpróbáltam szemléltetni a normálvektor működését.
A zölddel jelölt vektor legyen a normálvektor a háromszög pedig a felület amire merőlegesen mutat.
A tengelyeken nincsenek bejelölve egységek ugyhogy most tekintsük úgy mintha a zölddel jelölt vektor 1 egység hosszú lenne.

© Kiskapu Kft. Minden jog fenntartva

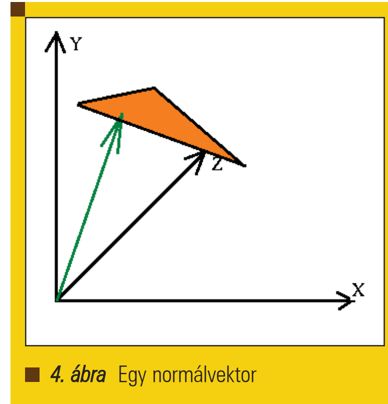
© Kiskapu Kft. Minden jog fenntartva



2. ábra Descartes-koordinátarendszer



3. ábra A (12,4,3) koordinátájú vektor



4. ábra Egy normálvektor

Mátrixok és a koordináta transzformációk

Azok akik még sosem találkoztak mátrixokkal – nem a *Neo* félével – vagy nem értik, hogy miről is lesz szó a most következő bekezdésben, ne keseredjenek el, inkább olvassák el még egyszer ezt a részt és/vagy nézzenek utána az interneten vagy egy könyvtárban.

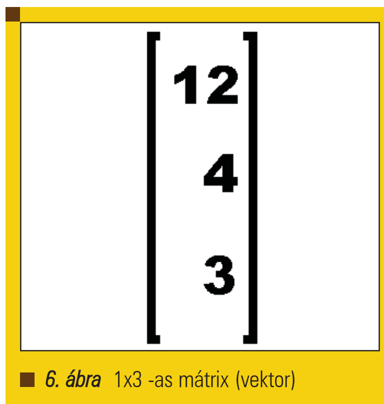
A mátrixok a matematikában nagyon fontos szerepet töltenek be. Segítségükkel egyszerűen oldhatunk meg nagyon bonyolult egyenletrendszereket vagy akár végezhetünk velük geometriai műveleteket. Egy mátrixnak sorai és oszlopai vannak, valójában nem más mint egy táblázat. Két mátrix elemein elvégezhetjük az összes műveletet amit a valós számkörben ismerünk, csak egy-két szabályra oda kell figyelnünk. Az *OpenGL* használatakor nekünk csak a mátrixok szorzására lesz szükségünk, ugyanis az *OpenGL* ezeknek a mátrixoknak a segítségével írja le a tér pontjainak elmozdulását, átméretezését és elforgatását. Akik még sosem láttak mátrixot, azok kedvéért bemutatok egyet az *5. ábrán*.

5. ábra 4x3 -as mátrix

Ez tehát egy számtáblázat, aminek 4 oszlopa és 3 sora van. Vannak speciális mátrixok, amelyeknek vagy csak egy oszlopa, vagy csak egy sora van (*6. ábra*).

Talán már meg is fordult a Kedves Olvasó fejében, hogy egy ilyen mátrix arra is alkalmas lehetne, hogy egy vertex koordinátáit tároljuk el benne. Ez remek ötlet!

Hogy miért? Azért, mert a mátrixszal képesek vagyunk leírni egy koordináta transzformációt. Koordináta transzformációnak azt nevezzük amikor egy művelet hatására egy test koordinátái valamilyen jól definiált rendszer szerint megváltoznak. Ilyen transzformáció például egy test elforgatása. Ha egy mátrix megfelelő elemeit behelyettesítjük szögfüggvényekkel és beszorozzuk vele a test pontjait tartalmazó másik mátrixot akkor a pontok „elfordulnak”. Szerintem nem szükséges a transzformációs mátrixok lelkivilágát tovább boncolgatni, ennyi bőven elég (még sok is) lesz ahhoz, hogy megbirkózzunk az *OpenGL* transzformációival, akiket pedig ennél jobban érdekel a dolog azok remek szakirodalomra bukkanhatnak ha beszerzik ***Hajós György Bevezetés a geometriába*** című könyvét.



6. ábra 1x3 -as mátrix (vektor)

Az *OpenGL* nem követeli meg tőlünk, hogy éjszakákon át tartó egyenletrendszer levezetésekbe és mátrix felírásokba bonyolódjunk. A *SGI* programozói előrelátóak voltak és lényegesen leegyszerűsítették a transzformációk használatát.

Forgatás: glRotatef(D, X, Y, Z)

Létrehoz egy transzformációs mátrixot és a kapott paramétereket a mátrix megfelelő helyeire írja be. D paraméter az elforgatás szögét határozza meg, az X, Y és Z paraméterek pedig azt, hogy melyik tengelyen történjen az elfordulás. A megfelelő paramétert 1 -re kell állítanunk.

Példa: glRotatef(45,0,0,1). A modelltér elfordul Z tengely mentén 45 fokkal.

Eltolás: glTranslatef(X, Y, Z)

Itt egy az elforgatás helyett egy elmozdulási mátrix jön létre amelynek a paraméterei az adott tengelyen történő elmozdulást adják meg.

Példa: glTranslatef(10,15,20). Ekkor az összes modelltérben lévő objektum elmozdul X tengelyen 10, Y tengelyen 15, Z tengelyen 20 egységgel.

Méretezés: glScalef(X, Y, Z)

Az átvett paraméterek az adott tengelyen történő skálázás mértékét adják.

Példa: glScalef(0.5, 0.5, 0.5). A modelltér minden objektumát a felére kicsinyíti.

Most akkor lássunk egy kis kódrészletet, hogy a fent vázoltak értelmet nyerjenek (*1. kód*).
Hát ez azért nem annyira szörnyű mint ahogy beharangoztam. Ugye? Ennek a kis kódrészletnek a láttán szörnyű kétely foghatja el az olvasót: mi az a `glVertex`?
A `glVertex` egyike az *OpenGL* legalapvetőbb utasításainak. Ennek a függvénynek a segítségével hozhatunk létre egy pontot a térben.
A pontok létrehozása előtt az *OpenGL*-t fel kell készítenünk arra, hogy most pontok adatai fogja kapni, ezért írjuk a `glVertex` függvényt `glBegin` és `glEnd` közé. A `glBegin` a fenti példában a `GL_TRIANGLES` kifejezéssel került meghívásra, de nem ez az egyetlen lehetőségünk. Ha `GL_QUAD`-ot adunk át `GL_TRIANGLES` helyett akkor az *OpenGL* nem három hanem négy koordinátát fog várni. Ezek természetesen egy téglalapot írnak le a térben. A `glVertex3f` utasítás három `float` típusú számot vár paraméterként melyek rendre a pont X, Y és Z koordinátáját határozzák meg.

Példa: glVertex3f(0.0, 0.0, 0.0) utasítás az origóba helyez egy pontot.

Nagyon fontos megemlíteni, hogy az egymást követően – egy `glBegin` és `glEnd` páron belül – definiált pontok egy felületet fognak képezni, vagyis az *OpenGL* „összeköti" őket.

Színezni is lehet?

Természetesen. Ha szeretnénk akkor a pontokhoz színeket rendelhetünk. Ha egy háromszög mindhárom csúcsához különböző színeket rendelünk akkor az *OpenGL* átmenetet képez közöttük!

Próbáljuk ki az alábbi kódrészletet:

```
glBegin(GL_TRIANGLES);
 glColor3f(1.0, 0.0 ,0.0);
 glVertex3f(0.0, 0.0, 0.0);

 glColor3f(0.0, 0.0, 1.0);
 glVertex3f(1.0, 0.0, 0.0);

 glColor3f(0.0, 0.0, 1.0);
 glVertex3f(0.5, 1.0, 0.0);
glEnd();
```

1. kód

```
...
glRotatef(45, 0.0, 0.0, 1.0); //elforgatjuk a képet 45 fokkal
                              //Z tengely mentén
glBegin(GL_TRIANGLES); //felkészítjük az OpenGL-t, hogy egy
                       //háromszög   adatait fogjuk megadni.
glVertex3f(0.0, 0.0, 0.0); //első pont koordinátája
glVertex3f(0.0, 1.0, 0.0); //második pont koordinátája
glVertex3f(1.0, 1.0, 0.0); //harmadik pont koordinátája
glEnd(); //rajzolás vége
...
```

2. kód

```
#include<GL\glut.h>
#include<stdlib.h>

void init(void)
{
	glClearColor(0.0, 0.0, 0.0, 0.0);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}
void display(void)
{
	glClear(GL_COLOR_BUFFER_BIT);

	// IDE A KIRAJZOLÁST glBegin ÉS glEnd közé!

	glFlush();
}

void keyboard(unsigned char key, int x, int y)
{
	switch(key)
	{
		case 27:
			exit(0);
			break;
	}
}

int main(void)
{
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
	glutInitWindowSize(200, 200);
	glutInitWindowPosition(100, 100);
	glutCreateWindow("elso OpenGL programom");
	init();
	glutDisplayFunc(display);
	glutKeyboardFunc(keyboard);
	glutMainLoop();
	return 0;
}
```

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva

A színek megadása itt ***RGB*** komponensek megadásával történik, vagyis a `glColor3f` függvény első paramétere a piros, a második a zöld, a harmadik a kék színkomponens telítettségét adja meg. Ha mindegyik 0 akkor fekete színt kapunk, ha mindegyik 1 akkor fehéret.
Most már tisztában vagyunk azzal, hogyan rajzolhatunk ki testeket és hogyan színezgethetjük őket, de mégis hova rajzoljuk őket? Ilyenkor fordulunk a cikk elején emlegetett ***GLUT***-hoz!

GLUT avagy Graphics Library Utility Toolkit

Mint már említettem a ***GLUT*** az ***OpenGL*** kibővítése amely rengeteg olyan műveletet elvégez helyettünk ami amúgy sok időnket venné el és gyakran meg kell csinálni. A ***GLUT*** leegyszerűsíti az ablakok esemény kezelését, hogy az eseményekhez ***callback eseményeket*** rendelhetünk amelyek az eseményhurokban (***event loop***) meghívódnak ha az őket kiváltó feltétel teljesül. Ilyen kiváltó ok lehet egy billentyű lenyomása, vagy az ablak elmozgatás, stb.
Első nekifutásra ennyi talán elég is lesz az áttekintésből. Mindenkinek javaslom, hogy látogasson el a ➲ ***www.opengl.org*** oldalra és nézelődjön. Egy újságcikkbe sajnos lehetetlen belezsúfolni az ***OpenGL*** legalapvetőbb képességeit és működését, ezért voltam ilyen felszínes. Talán nem is baj ha a száraz elméleti részt nem erőltetjük tovább és rátérünk a jóval izgalmasabb gyakorlati részekre.

Helló Világ?

Egy „egyszerű” példaprogram forráskódja a ➲ http://nehe.gamedev.net/data/lessons/linux/lesson04.tar.gz címről tölthető le és minden benne van amit a cikkben tárgyaltunk.
Érdemes nézelődni ezen a webhelyen, mert érdekes és jól dokumentált forráskódok találhatók itt úgy 15-20 programozási nyelvre!
Azok kedvéért akik szeretnének saját maguk is ilyeneket csinálni és az ***OpenGL*** képességeit próbálgatni azok kedvéért álljon itt egy üres sablon amelybe csak be kell helyettesíteniük a saját kódjukat (2. *kód*).
Nézzétek el nekem, hogy ebben a cikkben szinte semmi gyakorlat nincs. A következő részben már sokkal gyakorlatiasabb dolgokról esik majd szó: textúrázás és megvilágítás, effektek (köd, környezet-tükröző textúrák, átlátszóság/áttetszőség), a ***GLUT*** rejtelmei, animációk, az ***OpenGL*** kiterjesztései, játékfejlesztés ***Linuxra***.
Akinek különleges kérése, vagy kérdése van, írjon levelet! Megpróbálok válaszolni ahogy az időm engedi.

Tóth Péter
(thotacc@drotnet.hu)
A BMF hallgatója vagyok, mellette kis- és közép vállalatok rendszereinek Linuxos átállításával és szoftverfejlesztéssel foglalkozó cég informatikai vezetőjeként tevékenykedem. Kevés szabadidőmet barátnőmmel és barátaimmal töltöm egy-két sör társaságában.