OpenGL.HU XI. Rész

glInformation() Bevezető

A cikksorozatnak immáron a tizenegyedik részét tarthatják a kezükben az olvasók, akiktől egyre több levelet kapok, bennük kérdéseket, ötleteket, felvetéseket és nem utolsósorban témákat, amelyek a cikksorozat lényegét adják. A továbbiakban is buzdítok mindenkit, hogy jelezze, ha egy adott dologra kíváncsi. Ettől a számtól kezdve az OpenGL.Hu szerkezete egy kicsit át fog alakulni úgy, hogy öt részre bontom: lesz egy bevezető, aktuális információkkal, majd egy olyan rész, ahol leírom az aktuálisan használt szoftvereszközök listáját, megelőzve ezzel néhány kérdést. Ezek után következik egy, az olvasók által kért téma, s rögtön utána az, amit én választottam erre az alkalomra. Végül a zárszó és előretekintés fejezi be a cikket. A részek címei a már itt látható glInformation(), glToolBox, glRequest(), glArticle() és glEnd() lettek, nem kis szakmai ártalommal átitatott névválasztási rituálé után

glToolBox() Ami kell...

Fordító:

- BloodShed DevC++ v4.9.8.0

Library:

- GLAux,
- GLUT v3.7.6,
- ExtGLGen

glRequest() Mindent az olvasóért!

Erre a hónapra a kért témák közül olyat választottam, ami a legutóbbi szám „ködös” részére épült. Meglepően sok levelet kaptam, melyben azt kérdezték, hogy ezen az egyszerű köd-technikán kívül van-e más megoldás is, hiszen vannak esetek, amikor nem biztos, hogy a ködnek minden irányban ugyanúgy kell látszódnia. Nos, igen, van ilyen technika, s nem is egy bonyolult dologról van szó: ez a volumetrikus köd, eredeti, hangzatos nevén: volumetric fog. A dolog lényege, meglepő módon pont az, ami nekünk itt kell: egy olyan köd, amelyet sokkal jobban lehet kontrollálni, mint a korábban látott distance fog-ot. Egészen pontosan, a volumetric fog vertexekhez köthető, ugyanúgy, ahogyan egy textúra. Tehát egy vertexek által határolt teret tudunk vele meglepően egyszerűen kitölteni, mint például egy sötét szakadék vagy liftakna, aminek nem látszik az alja. A szemfüleseknek egyből beugorhat, milyen jól megoldható ez egy fekete ködde, amelyik csak azon a részen létezik, s a szakadék alján áthatolhatatlanul sűrű, feljebb haladva viszont ritkul.

Tehát, akkor most erre fogunk egy példát adni! Hogy mire is lesz szükségünk? A szokásos függvényeken és egyebeken kívül használni fogunk egy EXT_fog_coord nevű estension-t, amit remélhetőleg minden gond nélkül be fogunk tudni tölteni, ugyanis nem egy vadonatúj darab. Korábban már volt szó az extension-ök elegáns betöltéséről a OglExt library segítségével, így most is ezt kellene segítségül hívnunk. Igen, kellene. De sajnos ez esetben úgy tűnt, ez a lib csődöt mondott, így a továbbiakban mellőzni is fogjuk a használatát. A helyére természetesen kellett valami, hiszen valljuk be, senkinek sincs kedve pointereket gyártogatni az egyes extension-ökhöz. Az új „játékszer” glExtGen névre hallgat. Egy kis programról van szó, amely az SGI oldaláról letölthető glext.h fileból a videokártyánk tulajdonságait figyelembe véve egy headert és egy cpp filet generál, amelyet a programunkba illesztve gond nélkül érjük el az extension-öket.

Na, akkor lássuk most a lényeget: a kódot. Egyetlen vertex definiálását emelném csak ki, hiszen a dolog egyszerű elven alapul: a volumetrikus ködöt vertexekhez köti, pontosan, mint a textúrákat. Itt azonban koordináta helyett azt kell megadnunk, hogy az adott pontban milyen sűrű a köd. Természetesen a nem megadott pontokban a sűrűséget az OpenGL interpolálja, minél simább átmenetet képezve az egyes pontok között. Egyetlen pontra hasonló sorokat kell látnunk annak megadásakor:

```
glFogCoordf( 0.0f);
glTexCoord2f(0.0f, 0.0f);
glVertex3f(-.5f, -.5f,  -1.0f);
```

A glFogCoordf(float) függvény adja meg a sűrűséget az adott pontban, 0.0f és 1.0f közötti értékkel. Természetesen lehet más (pl. Normal) koordinátákat beállító függvényeket is használni, a FogCoord nem interferál velük. Sajnos, mint később látni fogjuk, vannak helyzetek, amikor nem tudjuk használni, de ettől függetlenül igen hasznos és kellemes látványt nyujtó kis technikáról van szó. Ha a koordinátákat beállítottuk, akkor azt kell megadni, hogy milyen messzire látszik el a köd és hol kezdődik, hasonlóan, mint a 'rendes' ködnél. Nos, itt a dolog annyival egyszerűbb, hogy a ködöt vertexek fogják körbe, így ezzel a távolsággal nem sok tennivalónk van, a legtöbb esetben a következő beállítás bőven megteszi:

```
glFogf(GL_FOG_START, 1.0f);
glFogf(GL_FOG_END, 0.0f);
```

Mióta használtam a volumetric fog-ot, ez a beállítás még mindig megtette, de persze nyugodtan lehet kísérletezni. A legrosszabb esetben nem jelenik meg köd...

Még egyetlen lépés van hátra, hogy teljesen működőképes volumetric fog boldog gazdái legyünk: meg kell mondani az OpenGL-nek, hogy a megadott koordináták alapján számolja a ködöt, effektíve hozzákötni ezzel azt az objektumhoz. Erre a célra az alábbi utasítást használhatjuk:

```
GlFogi(GL_FOG_COORDINATE_SOURCE_EXT,GL_FOG_COORDINATE_EXT);
```

Maga a függvény már talán nem ismeretlen: integer értéket állítunk be vele egy OpenGL rendszerparaméternek. Mint a példakódból is látszik, természetesen az összes eddigi ködbeállítást (típus, stb.) most is lehet, sőt kell alkalmazni.

glArticle() És lőn világosság!

Ez a téma egy kicsit több szempontból is érdekes lesz. Először is azért, mert igen fontos része az OpenGL-nek, másrészről azért, mert amennyire csak lehet, kerülik a fejlesztők a használatát, mert igen erőforrásigényes és mindemellett igen behatárolt feltételek között működő dologról van szó. Ez a dolog nem más, mint a dinamikus világítás (dynamic lighting, vagy esetünkben konkrétabban, a per-face dynamic lighting). A dinamikus világítás a már korábban tárgyalt shadow-mapping (vagy light light-mapping, ahogy tetszik) technikával ellentétben nem stacionárius fényeket és árnyékokat hoz léte, hanem nagyon is élőket. Itt olyasmire gondolok, mint a kiszáradt kútba dobott égő fáklya vibráló, ugráló fénye és az általa vetett árnyékok. Ez az ideális, mármár közhelyszámba menő példa egyenlőre még igen messze van tőlünk, de elég lesz egyelőre megismerni a dinamikus világítás alapjait. Az „alapok” szó apropóján megjegyezném, hogy a megvilágítás matematikáját és kevésbé a programozáshoz tartozó szabályait most nem szándékozom tárgyalni, mert ebben a témában bőven áll rendelkezésre egyetemi jegyzet és könyv, vagy cikk a világhálón és a könyvtárakban, boltokban.

A lehető legjobban a lényegre térve elmondhatjuk, hogy a legegyszerűbb világítási módszer, a per-face dynamic lighting gyakorlatilag az általános iskolai fizikaóra tananyagában a témáról fellelhető információk egy-az-egyben törénő számítógépes adaptációja. A lényege, hogy egy fényforrást hozunk létre valahol, amely megvilágítja az n db face-ből álló objektumunkat. Ha egy face egy kicsit meg van világítva, az jelen esetben azt jelenti, hogy az egész face meg van világítva: vagy világos a face vagy nem. Ennyire egyszerű. Nos, a fény definiálásához meg kell adnunk pár paramétert, mint például a fény színe:

```
GLfloat LightAmbient[]= {1.0f, 1.0f, 1.0f, 1.0f };
GLfloat LightDiffuse[]= {0.0f, 1.0f, 0.0f, 1.0f };
```

Már látom, hogy sokan vakargatják a fejüket, mi ez a két tömb. Nos, mindkettő standard RGBA formátumú adat, a fény színét adják meg. Hogy miért is itt van két darab szín definiálva? A válasz egyszerű: minden fény világítja a környezetét és a kiszemelt tárgyat is. Az első szín azt mutatja, milyen színnel lesz megvilágítva a környezet (úgy kell elképzelni, mint egy nagy fénycső „mindenhová eljutó”, egyenletes fényét!). Ez a szórt fény. A második az a direkt módon a tárgyakra vetülő fény, igazából ez látszik rajtuk meg jobban. Ezzel kapcsolatban megint csak kísérletezésre tudok majd bátorítani mindenkit, illetve arra, hogy írjon e-mailt, vagy kérdezzen bátran.

Ha a fény színével megvolnánk, jöjjön a megvilágítást elősegítő fény-objektum pozíciója, mint szükséges beállítás:

```
GLfloat LightPosition[]= {0.0f, 0.0f, -1.0f, 1.0f };
```

A pozíció itt az ún., homogén-koordinátás megadásban szerepel, aminek az első három tagja a szokásos X,Y,Z érték, a negyediket pedig hagyjuk most 1.0f-re állítva, hacsak nem akarunk egy hosszas „teológiai” elmélkedésbe belekezdeni a homogén-koordináták mibenlétével kapcsolatban.

Ha már ezeket az értékeket ilyen lelkesen definiáltuk, a következő kis blokk megmutatja, hogyan hozzuk őket az OpenGL tudomására.

```
GlLightfv(GL_LIGHT1, GL_AMBIENT,LightAmbient);
glLightfv(GL_LIGHT1, GL_DIFFUSE,LightDiffuse);
glLightfv(GL_LIGHT1,GL_POSITION,LightPosition);
```

Amint láthatjuk, az első paraméter egy definiált konstans, amely megjelöli, melyik fény számára lépnek életbe a beállítások. (A fények számára még visszatérünk!) Az OpenGL alapból 8 fényt támogat, de a legtöbb esetben ez még soknak is bizonyult, tekintve, hogy a dinamikus fény mindig hatásosan emészti fel a futtató rendszer erőforrásait. A második paraméter azt jelenti, hogy az adott számú fény melyik paraméterét szeretnénk beállítani, a harmadikról pedig már beszéltünk korábban is.

Ez a kódrészlet a legtöbb leírásban valamilyen egyszer lefutó függvényben van elhelyezve, de ezzel egy jó néhány dinamikus megoldástól fosszuk meg magunkat: pl falon vibráló fáklya fény. Ha megbékéltünk a fenti pár utasítással és ezeknek a felhasználásával már elkezdtünk egy kis programot csinálni, csak egy dolog van hátra: engedélyezni kell az OpenGL-nek a fény használatát.

```
glEnable(GL_LIGHT1);
glEnable(GL_LIGHTING);
```

Tehát, összegezzük: kiválasztottuk, beállítottuk, aktiváltuk a fényt és..... semmi?! Na, igen, egy dolog még kimaradt. Nos, a fizikában amikor azt számoljuk, hogy egy felület hol van megvilágítva, használni szoktunk egy ún. beesési merőlegest, amelyhez a fénysugarak szögét viszonyítjuk. Ugyanaz a viselkedés, ugyanazok a szabályok megvannak az OpenGL-ben is, de itt a beesési merőlegest ún. felületnormálnak hívjuk. Legegyszerűbb úgy elképzelni, mint a felületre merőleges 1.0f egységnyi hosszú szakaszt vagy félegyenest, amelyiknek azt a végpontját adjuk meg, amelyik nem érintkezik a felülettel. (Tehát egy pontot, amely pont egy egységnyire van a felülettől, s a kettőt össze tudjuk kötni úgy, hogy a képzeletbeli összekötő vonal merőleges legyen a felületre.) Ennek a pontnak a megadása a szokásos XYZ koordinátákkal történik az alábbi módon:

```
glNormal3f( 0.0f, 0.0f, 1.0f);
```

Ezt minden felületre csupán egyetlen egyszer kell megadni, s a többi számítást majd az OpenGL végzi már el. Ha vertexenként adjuk meg a testen az összes face-t, akkor elég kiábrándító lehet ezeknek a pontoknak a kiszámítása, de szerencsére könnyen lehet olyan függvényt írni, amely a koordinátákból ezt kiszámolja nekünk. Ha minden igaz, most már látnunk kell a megvilágítás hatását feltéve, hogy nem számoltunk el valamit.

`glEnd()` Zárszó, előretekintés

Remélem mindenkinek elnyeri tetszését ez az új cikkszerkezet! Ezek után főleg várok, mindenféle kommentárt és jobbító tanácsot, valamint glRequest() témát. Szóval, kérdezzenek bátran és én megpróbálok válaszolni. A következő számig kellemes kódolást és hasznos időtöltést kívánok mindenkinek.

Merczel „MaNiAc” László laszlo.merczel@mailbox.hu