3D-programozás Pythonban

Jason néhány Python-modul segítségével PyOpenGL programozási módszereket mutat be nekünk, épp csak érintve az OpenGL felszínét.

A grafikaprogramozás olykor fárasztó, igen fárasztó tevékenység. Ha programjainkat behemót 3D-függvénykönyvtárakhoz csatoljuk, a fordítási idő megnövekszik. Mivel gyakran megesik, hogy rengeteg finomhangolás szükséges ahhoz, hogy minden tökéletesen nézzen ki, általánosnak mondható, hogy az apróbb változtatások kipróbálását és a hosszú fordítgatást igénylő programokat összecsapják. Ezek a hosszadalmas hibakeresési ciklusok elég okot szolgáltatnak arra, hogy 3D-alkalmazásaink prototípusát olyan magas szintű nyelv segítségével készítsük el, mint amilyen a Python is.

Rengeteg 3D grafikai API-hoz a Pythonban kiegészítés is található. IRIX-rendszereken a Pythont olyan modullal csomagolják, amely hozzáférést nyújt az SGI IRIX GL könyvtárához. Egy Python-programból a JPython segítségével akár a Java3D API-t is használhatjuk, mely egy olyan Python-változat, ami Java virtuális gépből fut. Ez a cikk elsősorban az OpenGL-re összpontosít elterjedtsége és a hozzá biztosított, mind a Pythonhoz, mind a Linux alá megvalósított nagyszerű támogatás miatt.

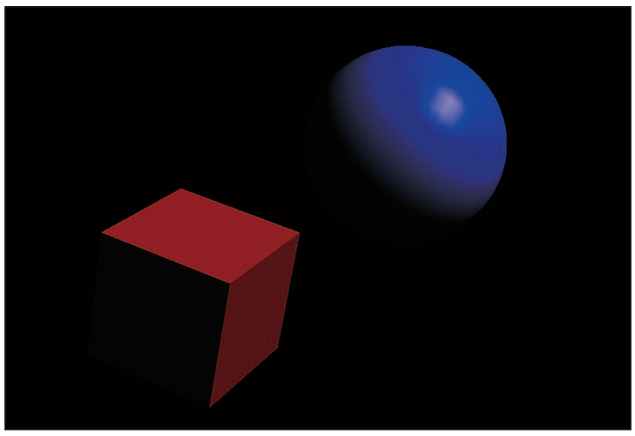
A PyOpenGL letöltése és telepítése

A PyOpenGL Python-modulokból álló készlet, mely hozzáférést ad az OpenGL-hez, csakúgy, akár egy sor más segédeszközhöz és OpenGL-kiegészítéshez, mellyel az OpenGL alacsony szintű felületét egészíthetjük ki. Az OpenGL-t eredetileg ***James Hugunin***, ***Thomas Schwaller*** és ***David Ascher*** fejlesztette ki. A program fejlesztését mostanában ***Tarn Burton*** vette át, a csomagot pedig ***Rene Liebscher*** és ***Michael Flatcher*** tartja karban.

Mivel az PyOpenGL szolgáltatásait nagyrészt OpenGL-hívások alkotják, ahhoz, hogy programozni tudj vele, az OpenGL alapszintű ismeretére lesz szükséged. Az OpenGL elsajátításához rengeteg ismertető és referencia létezik, ajánlásokat a cikk végén találsz hozzájuk.

A legelső követelmény a PyOpenGL-hez maga az OpenGL. Ha a gépeden még nincs működőképes OpenGL-változat, nézz utána, hogy GNU/Linux-terjesztésedben létezik-e hozzá csomag, vagy töltsd le a Mesa 3D grafikus könyvtárat a ➲ http://www.mesa.org címről. Hogy a PyOpenGL teljes gőzzel működhessen, szükség van még egy „Numerical Python" nevű könyvtárra is. A Numeric és a PyOpenGL forráskódja a ➲ http://numpy.sourceforge.net címről tölthető le. Telepítésük ***Greg Ward*** `distutils`-moduljának köszönhetően viszonylag egyszerű, ezt az 1.6-os változattól kezdődően megtaláljuk a Python-csomagban. Ha a kicsomagolt forráskód saját könyvtárából lefuttatjuk a `python setup.py install` parancsot, a parancs a modulokat összeállítja és telepíti. Mielőtt elindulnál beszerezni a csomagokat, győződj meg, hátha az általad használt GNU/Linux terjesztés is tartalmazza ezeket a programokat. Debianban például mindkét csomag megtalálható. Figyelem, a cikk megírásakor a PyOpenGL 1.5.7-es változatát használtam, de azóta már a 2.0-s változat is megjelent.

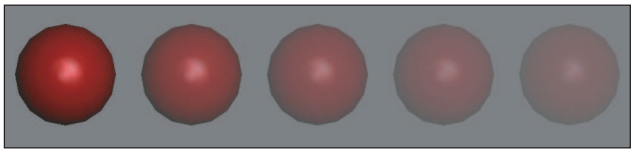
1. kép A 3. lista eredménye



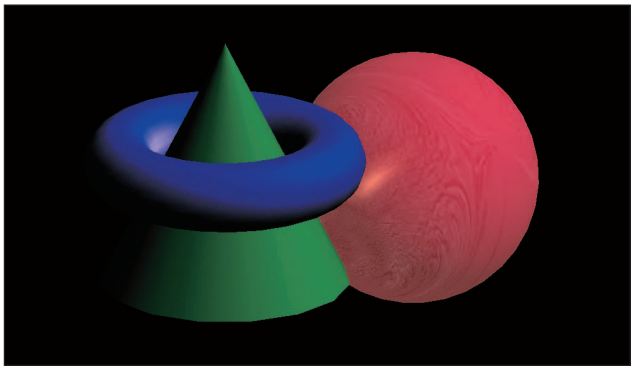
2. kép A 4. lista eredménye

Egy egyszerű Python OpenGL-alkalmazás

Az OpenGL megvalósítása nem határozza meg, hogy az OpenGL hogyan működjön együtt az ablakkezelő rendszerekkel. Ennek következtében az OpenGL-t használó programokban valamilyen külső GUI-eszköztárra is szükség van. Az első listában található program a GLUT-ot használja fel, mely az OpenGL-hez több felületen is elérhető. Hacsak nem valamilyen kereskedelmi OpenGL-változatot használsz, a GLUT minden bizonnyal rendelkezésre áll a rendszereden. Ez a kód egy ablakot nyit meg, beállítja a világítást és megjelenít egy teáskannát (1. lista ➲ http://www.linuxvilag.hu/Python). Eltekintve a Python írásmódjának egyszerűségétől, ez a program szinte ugyanúgy néz ki, mint C-beli megfelelője. Apró különbség a „display" szolgáltatás visszahívó függvényének beállítása. Ez C-ből vagy C++-ból a `glutDisplayFunc(display)` függvényhívással oldható meg. A visszahívó függvény beállítása az PyOpenGL-ből két lépésben történik: először meghívjuk a `glutSetDisplayFunc()` függvényt, ezt követően a `glutDisplayFunc()`-ot. Ez a sajátosság egyéb visszahívó függvényekre is jellemző, például a `glutMouseFunc()`-ra, vagy a `glutReshapeFunc()`-ra.



3. kép A fog.py a PyOpenGL része

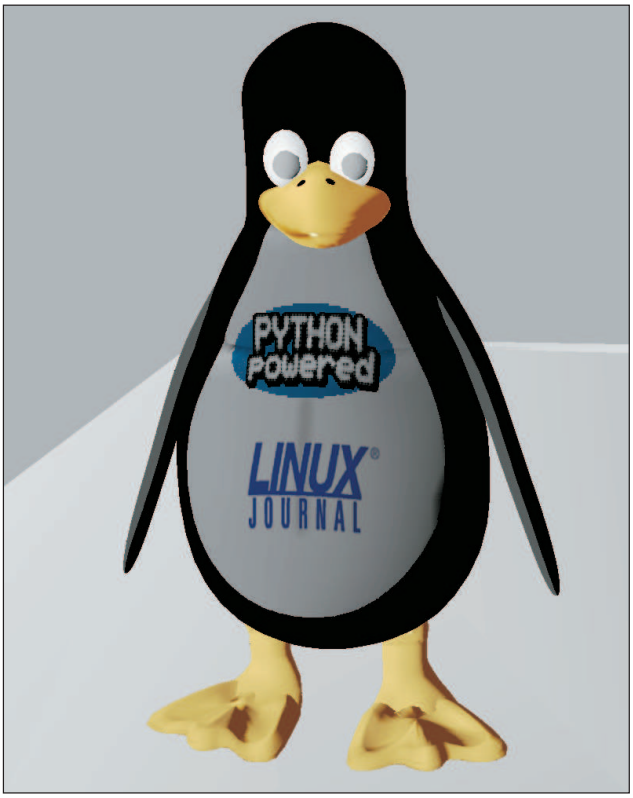


4. kép Egyszerű jelenet fénnyel és textúrával

A GLUT kisebb alkalmazásoknál nagyszerűen használható, ellenben meglehetősen sok munka szükséges ahhoz, hogy programjainkhoz alapvető és elvárt szolgáltatásokat adjunk hozzá, mint például egérrel vezérelt ráközelítések, kamerakövetés vagy forgatás. A Togl egy olyan Tkinter-összetevő, amely ezeket biztosítja, és emellett alapvető megvilágítást is beállít. A 2. lista (➲ http://www.linuxvilag.hu/Python) az előző programnak Togl felhasználásával készült változata.
Ez a program lényegesen rövidebb kódból áll, szolgáltatásaiban ugyanakkor felülmúlja az előzőt, de ennek ára a rugalmasság. Ha a Togl által biztosított fények és a felhasználófelület nem felel meg az igényeidnek, a Togl segítségével újratervezheted őket. A Togl prototípuskészítésre is tökéletes eszköz, mivel a melegítőlap megvilágítását és a navigálás megvalósítását is elvégzi helyettünk.
A PyOpenGL más 3D-s GUI-eszközkészletekkel is jól összevonható. Létezik kapcsolat wxWindowshoz, FLTK-hoz, FOX-hoz és GTK-hoz is.

Textúraleképezés használata

Textúraleképezésnek hívjuk azt a módszert, amellyel képadatot, mondjuk egy fényképet ragasztunk valamilyen sokszögre. Ha textúrákat szeretnénk használni, a képet először valamilyen OpenGL által támogatott alacsonyszintű formátumra át kell alakítani. Bár a Pythonban írt kód lényegében megegyezik az OpenGL alkalmazása során használt normál kóddal, a Python mégis nagymértékben leegyszerűsíti a textúrának használt képek betöltési és alkalmazási folyamatát.
A Python alapértelmezett könyvtára tartalmaz egy `rbgimg` nevű modult, mellyel az SGI ***imglib*** (***.rgb***) állományait olvashatjuk be. Bár elég homályos formátumnak tűnik, egyszerűségének köszönhetően könnyedén az alapkönyvtár része lehet, mivel nem sok helyet foglal. A Gimp vagy az ImageMagick segítségével bármilyen png, jpeg vagy tiff formátumú képet átalakíthatsz az SGI ***imglib*** formátumába. Az `rgblib.longimagedata(fÆjlnØv)` paranccsal egy teljes SGI imglib formátumú fájlt olvashatsz be, míg a függvény egy négybájtos RGBA képpontokat tartalmazó binárist karaktersorozat formátumba alakít át. Ez az adattípus már a `GL_RGBA` formátumkapcsoló és a `GL_UNSIGNED_BYTE` adattípus felhasználásával átadható az OpenGL függvényeinek



5. kép 3D Studio modell PyOpenGL-lel

(mint amilyen a `glTextImage2D` is).
Egy másik nagyon hasznos modul a Python alapkönyvtárából az ***imageop***. Ez a modul a képek vágásához, átméretezéséhez és szürkeárnyalatúra alakításához tartalmaz függvényeket. Az `imageop` függvények ugyanazon az `GL_RGBA/GL_UNSIGNED_BYTE` adattípuson dolgoznak, amellyel az `rgblib.longimagedata()` függvénye tér vissza.

A PyOpenGL kisegítő függvényei

Amellett, hogy a PyOpenGL tartalmazza az OpenGL függvényeit, még egy sor más, magasabb szintű függvényt is biztosít. Volt már szükséged arra, hogy egy OpenGL-jelenetből egyetlen állóképet merevíts ki magadnak? Természetesen bármikor lehetőséged van rá, hogy teljes képernyőképet (screenshot) készíts a képernyőről, mondjuk az xv-vel vagy a Gimppel, de mi történik olyankor, ha az OpenGL-jelenet egy olyan képkockájára van szükséged, amelyik csak bizonyos idő eltelte után jelenik meg, és csak egy pillanatra? A képernyőkép készítése akkor sem megoldás, ha olyan programról kell képet készítened, mely felhasználói beavatkozás nélkül fut, mint mondjuk egy CGI-parancsfájl. Az sem igazán kivitelezhető, hogy egy videó elkészítéséhez minden másodpercben egy sor képet készíts. Emiatt a PyOpenGL a képek kimerevítéséhez beépített függvényeket tartalmaz, amelyek azután többféle formátumban felhasználhatók. A *3. listában* található program annyival módosult az előző teáskannás programunkhoz képest, hogy a kép megjelenése után egy PostScript-pillanatképet készít róla, majd a lemezre mentve azonnal kilép a programból. Az *1. képen* magad is láthatod, programunk milyen képet készített.
A jelenetet a következő paranccsal mentettük:

```
openglutil.glSaveEPS( ex3.eps , 240, 240)
```

Az első érték annak a fájlnak a nevét adja meg, ahová menteni szeretnénk. A második és harmadik pedig rendre a szélesség és magasság értéket tartalmazza. A `glSavePPM()` függvényt ugyanezekkel az értékekkel meghívva ugyanúgy egy képet készítene, csak EPS helyett úgynevezett „portable pixmap" fájlformátumban. Ennek megfelelően ha a PyOpenGL tartalmaz tiff-támogatást, a képet a `glSaveTIFF` paranccsal tiff-be menthetnénk.
A PyOpenGL az OpenGL objektumkijelelő módszeréhez is kényelmes kezelőfelületet biztosít. Ezzel a kijelölő móddal egyértelműen megállapítható, hogy a kép egy adott régiójában mely objektumok találhatók. Ez annyit jelent, hogy a képen bármely pillanatban megjelölhető egy pont, ahol mondjuk az egérrel kattintottunk, és az OpenGL megmondja, hogy abban a pontban éppen melyik objektum található.
A PyOpenGL kijelölő művelete a következő módon állítható be:

1. Készíts függvényt vagy eljárást, amellyel a jelenet objektumait fogod kirajzolni, mindezt az OpenGL `glPushName()` és `glPopName()` függvényei között elhelyezve.
   A `glPushName()`-nek minden objektum kirajzolása előtt egy integer típusú számot adj meg, amellyel az OpenGL később hivatkozni tud az objektumaidra.
2. Állítsd be az egérkattintás eseménykezelőjét. Ennek beállítása attól függ, hogy milyen grafikus eszközkészletet használsz. Tkinter esetén például a `bind()` eljárásra van szükséged, ennek kell átadnod annak a függvénynek vagy eljárásnak a referenciáját, amelyet szeretnél, ha kattintáskor meghívódna.
3. Amint az egérrel kattintasz, az x és y koordinátákat az 1. pontban készített referenciával együtt add át az OpenGL `GL.glSelectWithCallback()` függvényének. Ez a függvény egy „tuple" típusú (közelség, távolság, nevek) elemeket tartalmazó listával tér vissza, melyből az első két érték az objektum mélységét jelöli, a harmadik pedig azon értékek a listája, melyet a `glPushName()` függvénnyel az első pontban megadtunk. Amennyiben a kérdéses helyen semmilyen objektum nincs, a függvény egy üres „tuple"-lal tér vissza.

A *4. listában* erre a kijelölő módszerre található példa. Próbáld ki, milyen hatást érsz el, ha a CONTROL gomb nyomva tartása mellett a bal egérgombot különböző helyeken nyomod le!
Ennek a feladatnak az elvégzésére a kijelölőmechanizmus a legegyszerűbb mód, de természetesen nem az egyetlen. Egy másik technika, ha az objektumokat egy háttér átmeneti tárban állítjuk össze, melyet a képernyőn nem jelenítünk meg. Ennek elvégzéséhez minden objektumot más színnel mentsünk, és a jelenetet kétdimenziós háttér átmeneti tárban állítsuk össze. A szín lekérdezésével megállapíthatjuk, milyen objektum található a képernyő adott részén. Ezt a háttér átmeneti tárat az OpenGL adagoló átmeneti tárának hívjuk.
A harmadik lehetőség az objektumok kijelölésére, ha a metszéspontok megadásával kézzel számítjuk ki, hogy egy pontban milyen objektum lehet. Ez annyit tesz, hogy el kell indítanod egy sugarat a képernyő valamely pontjáról, és meg kell határoznod, hogy a sugár mely objektumokat metszi. Amint megkaptad a metszett objektumok listáját, a mélységi tulajdonságokat figyelembe véve megállapíthatod, hogy melyik objektum található a legközelebb a képernyőhöz. Bár ez a megoldás jelenti a folyamat feletti legnagyobb irányítást, ennek a megoldásnak a kivitelezése egyúttal a legbonyolultabb is, és Pythont használva nem is a leghatékonyabb.

Teljesítménynövelés

Most, hogy már tudod, hogyan írj egyszerű OpenGL-programokat, bizonyára érdekel, hogy a méretezhetőség szempontjából a Python mennyire felel meg a 3D-s grafikus objektumok támasztotta igényeknek. Alapesetben a Python grafikus objektumainak sebessége természetesen csak kullog a C és a C++ nyomában, de megfelelő módszereket alkalmazva egészen szép eredményeket érhetünk el.
A legfőbb stratégia a sebesség növelésére, hogy minél jobban csökkentsük azt az időt, amelyet a kód a Python értelmezőjében tölt el, mindezt úgy, hogy az időigényes műveleteket natív kódba helyezzük át. Ennek elvégzéséhez minden lomha műveletet Python helyett C-ben vagy C++-ban írunk meg, így a lefordított kódnak nem kell a Python értelmezőjével vesződnie, viszont az eredmény a Pythonból is elérhető és felhasználható. Bár ezzel a megoldással jelentős sebességnövekedést érhetünk el, elveszítjük a Python használatából adódó egyszerűséget. Továbbá az is szükséges, hogy tudjuk a módját, hogyan írhatunk egy másik nyelven Pythonhoz kapcsolódó modulokat. Ugyanakkor ha az egészet C-ben írnád, nem kellene Python-bővítmények használatával vesződnöd.
Az OpenGL megjelenítési listái lehetővé teszik, hogy a sebességigényes dolgokat natív kódban írd meg, mindezt az előző megoldással járó számos gond elkerülésével. A megjelenítési listákkal egész OpenGL-műveletek helyezhetők az OpenGL gyorstárába, ahonnan az adatok egyenesen az OpenGL objektum-összeállító csővezetékébe haladnak tovább. Bizonyos körülmények között az OpenGL arra is képes, hogy megjelenítési listáit magán a grafikus kártyán tárolja, messze a Python gépezetétől.
A `glGenLists()` függvénnyel üres megjelenítési listákból álló tömböt hozhatunk létre. A függvény egyetlen értéket vár, annak számát, hogy hány megjelenítési listát hozzon létre. Visszatérési értéke a sikeresen létrehozott listák száma.
Az OpenGL `glNewList()` és `glEndList()` függvényeivel határolhatjuk be az elkészítendő listákat. Miután ezzel megvagyunk, az egyes listákat a `glCallList()` függvénnyel hívhatjuk újra elő. A PyOpenGL megjelenítési listák kezelésére szolgáló API-ja mondhatni teljesen megegyezik a hasonló feladatot ellátó C nyelvű kiterjesztéssel.

További lépések

Az eddigiekkel még csak épp hogy megérintettük a lehetőségek felszínét, melyet a Python és OpenGL eszköztára nyújt. További adatokért a PyOpenGL leírásához fordulj, melyet
➲ http://pyopengl.sourceforge.net/documentation/index.html
címen érhetsz el.

Jason Petrone
(jpetrone@acm.org) a CNRI (Nemzeti Kutatási Kezdeményezés) technikai munkatársa.
A Python és OpenGL erejét még előző munkahelyén ismerte fel, miközben egy CAVE virtuális valóság projekten dolgozott az NCSA-nál.

Könyvek

A Pythonban való OpenGL-programozásról egyelőre még nem jelent meg könyv, de az OpenGL általános programozásához3 az Addison-Wesley már megjelentetett egy írást, címe „The OpenGL Programming Guide". Mindemellett az OpenGL kézikönyv szintén nagyon jó könyv, melyet nem árt, ha magadnál tartasz.