Java 3D avagy 3D grafika bárhol

Java. Ez a szó mindenkinek ugyanazt kell az eszébe juttassa: objektumorientáltság, platformfüggetlenség, könnyen elsajátítható szintaxis. 3D. Ezt olvasván pedig mindenki pörgő cselekményre gondol, háromdimenziós világba helyezett agyonoptimizált kódok, fölösleges utasítások nélkül. Most pedig joggal jöhet a kérdés: Miként lehetséges ezt a két fogalmat kibékíteni egymással. Márpedig lehetséges. Hogy miként? Hát erről szól a cikk.

Mi is ez a Java3D API?

A Java3D egy Java standard kiterjesztés, mely segítségével 3D grafikát megjelenítő, illetve ezzel a grafikával interakciót végző Java programokat készíthetünk. Az API a rendelkezésünkre bocsát egy osztálygyűjteményt, mely minden 3D grafikához kapcsolódó művelet elvégzését biztosítja. Ezen osztályok ú.n. magas-szintű hozzáférést biztosítanak a grafikai elemekhez, ez alatt azt kell érteni, hogy nekünk nem kell olyasmivel bajlódni, mint koordináta transzformációk, meg „polygon fillező” algoritmusok gyártása.

A Java3D ideális a kezdő 3D programozók számára, mert könnyen elsajátítható szintaxisa révén betekintést nyújt a 3D világ felépítésébe, anélkül, hogy a programozónak bajlódnia kellene mélyremenő szintekig a koordinátákkal, meg hasonló „kis” részletkérdésekkel. Ideális eszköz szemléltető alkalmazások, 3D bemutatók készítésére, hisz aránylag kis erőfeszítéssel nagyon látványos eredményre juthatunk.

Platform, függetlenség és másegyebek

A Java3D csomag mostmár szinte minden operációs rendszerre elkészült, bővebb információ a **www.j3d.org** címen található. Szintén ezen a címen (angolul nem tudók részére: sajnos angolul) található rengeteg tutorial, kód, meg link, ahol sok Java3D hez kapcsolódó hasznos információ található.

Manapság mindent le lehet tölteni az internetről, így a Java3D t is, a következő címről: **http://java.sun.com/products/java-media/3D/**. Miután kiválasztottuk a nekünk megfelelő változatot, és le is töltöttük, egyszerűen végrehajtva a letöltött programot már fel is installálódott a rendszer. Vigyázzunk azonban egy dologra: a Java3D nek minimum egy Direct3D (Windows) kompatibilis videokártya az alapigénye, jobb esetben pedig megpróbálkozhatunk az OpenGL változattal is.

Sebességgondok a java programok esetében bizony előfordulnak, viszont ez a Java3D esetében a minimálisra lett csökkentve. A Java3D egyenes kapcsolatban van az alatta elterülő réteggel, ami nem más mint a Direct3D, vagy az OpenGL, és aránylag jó eredményeket ér el. Sajnos nincs mód ezt a két réteget „direkt” programozni Java ból, csak közvetett módon, az API segítségével érhetjük el a grafikát.

Nézzünk bele a csomagba

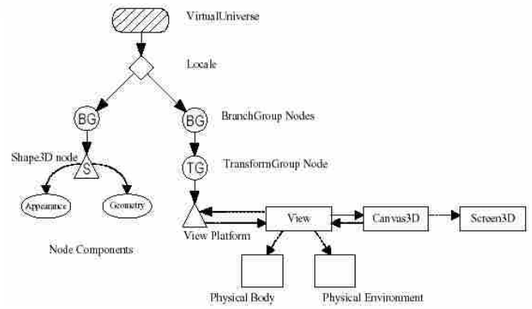
Minden Java3D program Java3D objektumokból van összerakva, szigorúan követve az objektumorientált programozás szabályait. Egy Java3D program mindig egy virtuális világot tartalmaz (VirtualUniverse). Ez a virtuális világ tartalmazza a maga során a 3D objektumokat, és az ezeken az objektumokon végrehajtott transzformációkat (forgatás, eltolás... igen, ezek is objektumok).

Egy Java3D virtuális univerzum egy gráf alapján van felépítve, mely sokatmondóan a „Scene graph” nevet viseli. Ez a Scene graph tartalmazza az összes Java3D objektumot, amelyek a világ geometriáját definiálják. Ez a gráf csomópontokat tartalmaz, amelyek nem másak, mint a világban előforduló objektumok, forgatások, fények, satöbbi. Ezek közt a csomópontok közt a legtöbb esetben apa fiu kapcsolat van (pld, egy transzformációs csoport tartalmaz egy kockát), viszont pár esetben referencia típusú a kapcsolat (egy 3D objektum tartalmaz egy referenciát a pontokra, amelyek őt alkotják). A csomópontok olyan tulajdonságokkal bírnak, mint például futás közbeni módosítás (a csomópont tulajdonságainak írása, olvasása, például a koordináták változása, amit igen jól lehet használni a forgatáskor, meg morpholáskor, stb...) és másegyebek, melyekre nem térek most ki, hisz megtalálhatóak a Java3D dokumentációban, melyet a fent említett címről lehet letölteni.

A Scene graphnak két fő ága van: az egyik, mely tartalmazza az objektumokat, ez a „content branch”, vagyis a tartalom ág nevet viseli. A másik ág a világban előforduló transzformációkat tartalmazza. Ennek a neve: „view branch”.

Egy másik fontos művelet a Scene graph lefordítása, vagyis „kompilálása”. Ezzel operációs rendszer baráttá tesszük az osztályunkat, és rengeteg számítást elvégezünk, úgymond előregenerálunk egy pár változót. A következő kép a Java3D világ Scene graphjának bemutatását tűzte ki céljául.

Forrás: Sun Microsystems



A világ receptje

Hogy egy Java3D programot logikailag helyesen fel tudjunk építeni, a következő lépéseket kell végrehajtani:

1. Hozzunk létre egy Canvas3D objektumot. Ez az objektum lesz felelős a felhasználóval való kommunikációért, illetve ez a Java objektum, amely „rajzolni tud” a képernyőre.

2. Hozzunk létre egy VirtualUniverse objektumot. Ez lesz a világunk alapja.

3. Hozzunk létre egy Locale objektumot. Ez fogja tartalmazni a virtuális világ összes 3D objektumát.

4. Hozzuk létre a view branchot:
   a. Hozzunk létre egy View objektumot
   b. Hozzunk létre egy ViewPlatform objektumot
   c. Hozzunk létre egy PhysicalBody objektumot
   d. Hozzunk létre egy PhysicalEnvironment objektumot.
   e. Kapcsoljuk össze a ViewPlatform, PhyisicalBody, PhyisicalEnvironment, és a Canvas3D objektumokat a View objektummal.

5. Hozzuk létre a ContentBranch-ot (azaz a 3D világ objektumait, mint pl. kocka, satöbbi)

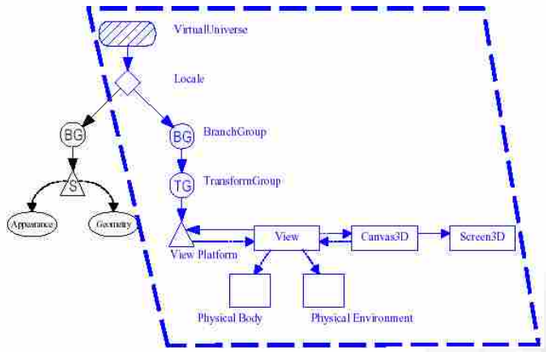
6. Fordítsuk le a gráfot

7. Szúrjuk be a gráfot a Locale objektumba

Egy egyszerű világ

Azoknak, akiket megijesztett az előbbi lista, megnyugtatásként közlöm: van egyszerűbb módja is, hogy Java3D programot írjunk. Mégpedig használván egy SimpleUniverse objektumot, amelyet természetesen a SimpleUniverse osztály alapján hozunk létre. Ez a SimpleUniverse osztály teljes egészében kiveszi kezünkből a 4. pontot, és létrehozza az összes szükséges objektumot, mint ahogy a következő kép mutatja

Forrás: Sun Microsystems



A kékkel bekeretezett rész kezelését egyszerűen el lehet felejteni, nekünk nem marad más, mint az objektumok, és transzformációk létrehozása, illetve beillesztése a virtuális világba.

És végül egy program...

...mely a szokásos alapműveletet végzi el: forgat egy kockát. Figyeljük meg, milyen elegáns módon van „elintézve” a kocka forgatása: Egyszerűen hozzáadtunk egy Rotation Interpolator objektumot a gráfhoz. És ennyi. Apropó, ahhoz, hogy leforduljon, szükség van a java3d jar file-ok berakására a CLASSPATH ba, mint pl: j3dcore.jar, satöbbi...

```
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
import java.awt.GraphicsConfiguration;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.ColorCube;
import com.sun.j3d.utils.universe.*;
import javax.media.j3d.*;
import javax.vecmath.*;

public class HelloUniverse extends Applet {
        /**
         * Ez a fuggveny letrehozza a scene graph objektumot.
         **/
        public BranchGroup createSceneGraph() {

                // Ez lesz a graf "gyokere"
                BranchGroup objRoot = new BranchGroup();

                // Hozzuk letre a transzformacios csoportot,
                // melynek beallitjuk azt a tulajdonsagat, hogy
                // futas kozben modisithato legyen.
                TransformGroup objTrans = new
                TransformGroup();
                        objTrans.setCapability
                (TransformGroup.ALLOW_TRANSFORM_WRITE);

                //Adjuk hozza ezt az obejktumot a grafhoz.
                objRoot.addChild(objTrans);

                // Hozzunk letre egy egyszeru objektumot, pld egy
szines kockat
                objTrans.addChild(new ColorCube(0.4));

                // Hozzunk letre egy viselkedes objektumot, amely
szerep
                // az lesz, hogy elvegzi a szukseges
transzformaciot a
                // megfelelo transfrom groupon.
                Transform3D yAxis = new Transform3D();
                Alpha rotationAlpha = new Alpha(-1, 4000);

                RotationInterpolator rotator =
                        new RotationInterpolator(rotationAlpha,
objTrans, yAxis, 0.0f,
                                        (float) Math.PI * 2.0f);
                BoundingSphere bounds =
                        new BoundingSphere(new Point3d(0.0, 0.0,
0.0), 100.0);
                rotator.setSchedulingBounds(bounds);
                objRoot.addChild(rotator);

                // Forditsuk le a grafot
                objRoot.compile();

                return objRoot;
        }
        /**
         * Ez egy uj obketkumot hoz letre.
         **/
        public HelloUniverse() {
                setLayout(new BorderLayout());
                // hasznaljuk a SimpleUniverse osztalyt, hogy
megkimeljuk
                // magunkat sok munkatol
                GraphicsConfiguration config =

SimpleUniverse.getPreferredConfiguration();

                Canvas3D c = new Canvas3D(config);
                add("Center", c);

                // Hozzuk letre a scene graphot.
                BranchGroup scene = createSceneGraph();
                SimpleUniverse u = new SimpleUniverse(c);
                // Kis modositas, hogy a "legjobb" ralatast
kapjuk a vilagra.
u.getViewingPlatform().setNominalViewingTransform();

                u.addBranchGraph(scene);
        }
        /**
         * Ezzel futtatjuk a programot
         **/
        public static void main(String[] args) {
                new MainFrame(new HelloUniverse(), 256, 256);
        }
}
```

Akiknek a fantáziáját megragadtam ezzel a kis bemutatóval, annak javallom, hogy mélyebben tanulmányozza ezt a java kiterjesztést, hisz nagyon „handy tool”-t kap a kezébe 3D világok tervezéséhez.

Deák Ferenc f.deak@freemail.hu