

## 3D ábrázolás

## PoVRay (7. rész)

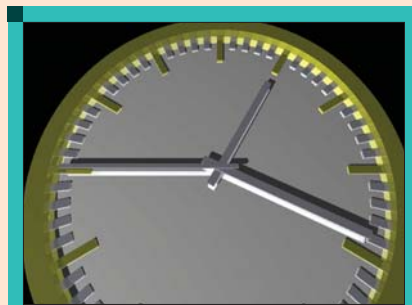


Szinte mindenki találkozott már az „animáció” szóval, amelyet általában rajzfilmekhez kötünk, holott a szó igazi jelentése a „mozgatás” vagy „haladás”, az „animátor” pedig az a személy, aki ezt a tevékenységet műveli, működteti a rá bízott „dolgot”. Nekünk a szó közismert értelmében kell animációt készítenünk, mégpedig a 3D világunkban haladni az idővel és mozgatni vagy átalakítani a tárgyakat.

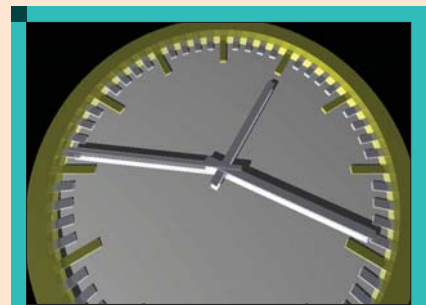
## Kiindulás

Az animációk készítésénél látszik igazán, hogy a *PoVRay* egy háttérprogram (backend), amely alapvetően nem arra készült, hogy könnyedén kezelhető legyen, hanem egy felhasználói felület mögött hatékonyan és gyorsan képes legyen előállítani a kért 3D világot. Ugyanis egyszerűen csak két változót kapunk a programtól, amelyekkel megtudhatjuk, hogy hányadik képkockát készítené el éppen; illetve egy „órajel”, amely alapesetben 0.0 és 1.0 között változik, jobban mondva az értéke egyenletesen növekszik. Ennél több kényelemre sajnos nem számíthatunk.

Az animáció elkészítéséhez szükség van egy – már kész – 3D világra, amelynek egyes elemeit meg akarjuk mozgatni vagy át akarjuk alakítani. Erre a célra tökéletes „alany” az előző részben elkészített óra, amelynek próbaképpen a másodperc mutatóját fogjuk mozgatni egy 60 kockából álló animáció keretében. Ehhez egyszerűen csak annyit kell tennünk,



■ 1. ábra Az első kocka



■ 2. ábra A második kocka

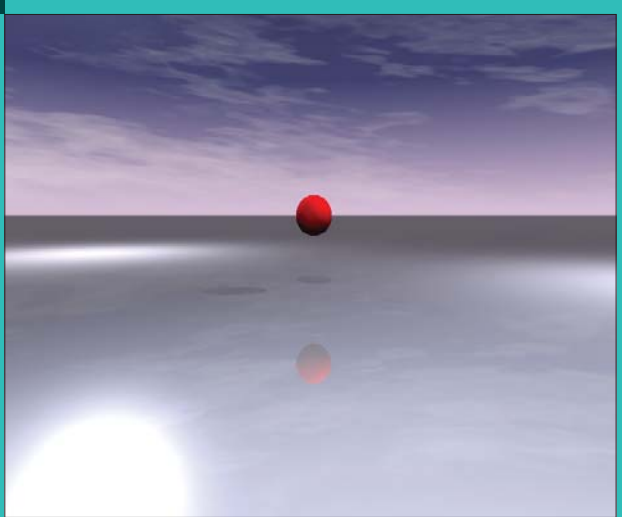
hogy az álló 3D világba bevisszük a másodpercmutató mozgatását:

```
#declare OraMasodperc=clock*60;
```

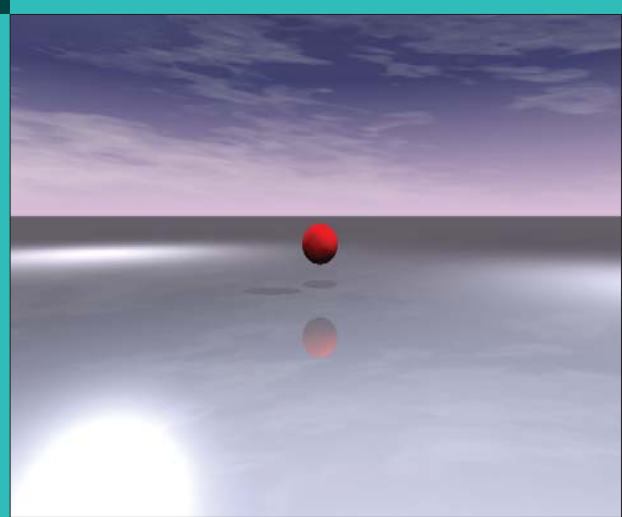
Ahhoz, hogy a *PoVRay* ne csak egy képet készítsen, meg kell adnunk a parancssorban, hogy animációt szeretnénk eredményül kapni:

```
$ povray +w800 +H600
↳ +I/usr/share/povray-3.6/
↳ include/ +KFI0 +KFF59
↳ anim00.pov
```

A *PoVRay* a parancssornak megfelelően készíteni fog 60 darab állóképet, amelyek más-más fázisát tartalmazzák a mozgóképnek. A `clock` változó, mint „órajel”, a képkockák (`frame`) számától függő kis növekményekkel tart a 0.0 értéktől az 1.0 értékig: jelen esetben 1/60-ad ez a növekmény. A *PoVRay* egyszerűen csak annyit csinál, hogy hatvan-szor meghívja a *renderelő* komponensét más-más „órajel” értékekkel. Az elkészült képeket pedig az `anim00.png` helyett az `anim0000.png` – `anim0059.png` nevű állományokba menti.



■ 3. ábra Kis piros labda lebeg a padló felett



■ 4. ábra Kis piros labda esik a padló felé (20. képkocka)

Ha megnézzük ezeket a képeket, akkor láthatjuk az egyes fázisokat, ahogy halad előre a másodpercmutató az óra számlapján. A *PoVRay* újabb hiányossága, hogy csak állóképek sorozatát képes elkészíteni, ebből összefüggő videót más programmal tudunk készíteni, például a *mencoder* tökéletes választás lehet:

```
$ mencoder "mf://anim00*.png"
↳ -o anim00.avi -ovc lavc
```

Az óra animációját az *anim00.avi* állományban találhatjuk meg, természetesen lehet akár hangot is hozzáadni, de ezt már a *mencoder* leírásában (vagy Bokor Norbert cikkében) keressük meg...

### Összetett mozgások

Sajnos csak a fenti eszközeink vannak animációk készítéséhez, amelyekkel igen szegényes módon tudunk „igazi” animációt készíteni, ahol a testek haladása nem vezethető vissza tisztán egyenes vonalú mozgássá vagy forgássá. Példaképpen egy labda pattogását nézhetjük, amely ideális esetben pont olyan magasra pattan vissza, mint ahonnan elejtettük, illetve esés közben folyamatosan gyorsul, majd felpattanva folyamatosan lassul. Egy kis fizika és matematika szükséges ahhoz, hogy a labda pattogjon a 3D világunkban, ugyanis a valóságos térben a gravitáció gyorsítja esés közben, illetve a gravitáció lassítja, miután visszapattant.

A „kísérlethez” kell egy sima padló és egy piros labda (3. ábra, *anim01.pov*):

Célunk, hogy a labda közel élethű módon zuhanjon a padló felé, majd azt elérve pattanjon vissza. Ehhez az képletre lesz szükségünk, vagyis a labda aktuális sebességét és megtett útját a Földön megszokott gravitációs gyorsulás és az eltelt idő határozza meg. Ezeket egyszerűen át kell fordítani a *PoVRay* matematikai nyelvére:

```
#if (0.5>clock)
#declare ido=clock*2;
#declare ut=3-3*ido*ido;
#else
#declare ido=(1.0-clock)*2;
#declare ut=3-3*ido*ido;
#end
```

A *PoVRay* által biztosított *clock* értéket két részre bontjuk, az idő első felében a labda esni fog, a második felében pedig felpattan. Az út kiszámolásánál ezeket mind figyelembe vesszük, majd kivonjuk abból a magasságból, ahonnan a labda indulni fog. Már csak a labda elhelyezésénél kell ezt az utat felhasználni:

```
sphere{
<0,0.8+ut,0>,0.8
texture{
pigment{
Red}}}
```

Az úthoz ne felejtjük el hozzáadni a labda sugarát, különben félig eltűnik a talajban, ami – valljuk be – nem túl realiztikus.

A valóságban a lepattanó labda egy picit benyomódik, amit nem látunk,

a 3D világunk azonban nem a valóság, a labdát a pattanás pillanatában össze tudjuk lapítani, hogy a rajzfilmekhez hasonlóan vizuálissá tegyük a pattanás tényét. Ez az összelapítás azonban nehéz ügy, hiszen akkor kell lapítanunk, amikor a labda kissé eltűnne a földben. A lapítás pedig pont annyival kell történnjen, hogy a labda mindig érintse a talajt egy pontban:

```
#if (0.5>clock)
#declare ido=clock*2;
#declare ut=3.0-3.4*ido*ido;
#else
#declare ido=(1.0-clock)*2;
#declare ut=3.0-3.4*ido*ido;
#end
```

```
#if (ut<0)
#declare labdaScale=-ut/0.8;
#else
#declare labdaScale=1.0;
#end
```

```
sphere{
<0,0,0>,0.8
scale <0,labdaScale,0>
translate <0,0.8+ut,0>
texture{
pigment{
Red}}}
```

A fenti esetben a labda útját nem nulláig, hanem -0.4-ig vezetjük, s pont annyira nyomjuk össze, mint amennyit a talajba mélyedve töltene. Nem túl szép, a rajzfilmekben kicsit kevesebb matematikával és több tapasztalattal csinálják...



5. ábra Kis piros labda esik a padló felé, s odaérve lapul (38. képkocka)



6. ábra Kis piros labda érkezik jobbról (12. képkocka)



7. ábra A kis piros labda már alig pattog (50. képkocka)

### Mozgások kombinálása

A kis piros labdánk érkezhethetne jobbról, s folyamatosan balra haladva pattanhatna néhányat. Ehhez ki kell terveznünk az órajel tartományát és leválasztani a nekünk szükséges tartományokat. Ez a leválasztás annyit jelent, hogy az órajelet 0.0 és 10.0 között változtatjuk s ennek a tört részét használjuk fel a labda pattogtatásához, a többi részét pedig a jobbról balra történő mozgathatáshoz:

```
#declare myClock=clock-floor
↳(clock);
#declare haladas=(clock-5.0)*3;

#if (0.5>myClock)
#declare ido=myClock*2;
#declare ut=3-3*ido*ido;
#else
#declare ido=(1.0-myClock)*2;
#declare ut=3-3*ido*ido;
#end

sphere{
```

```
<haladas,0.8+ut,0>,0.8
texture{
pigment{
Red}}}
```

Egy apró hiányérzet jelentkezik a videót nézve, mégpedig az, hogy a valóságban a labda egyre kisebb magasságra pattan vissza, a visszapattanás és a gurulás mozgási energiáját von el:

```
#declare myClock=clock-floor
↳(clock);
#declare haladas=(clock-5.0)*3;
#declare magassag=3/(clock+1);

#if (0.5>myClock)
#declare ido=myClock*2;
#declare ut=magassag-
↳magassag*ido*ido;
#else
#declare ido=(1.0-myClock)*2;
#declare ut=magassag-
↳magassag*ido*ido;
#end

sphere{
<haladas,0.8+ut,0>,0.8
texture{
pigment{
Red}}}
```

Az utolsó – egyben lezáró – részben csokorba szedem a *PoVRay* apró tulajdonságait, amelyekkel jelentősen javíthatjuk a 3D világ minőségét. A cikk második felében pedig bemutatok röviden és tömören pár *PoVRay* előtét programot.



**Auth Gábor**  
(auth.gabor@enaplo.hu)

Egy pécsi középiskolában informatikát és programozást oktat.

Tíz éve botlott először a UNIX rendszerekbe, 7 év Linux használat után kapta el a FreeBSD lázat, amiből máig nem tudott kigyógyulni.

### KAPCSOLÓDÓ CÍMEK

- A PovRay projekt honlapja  
↳ <http://www.povray.org>
- A cikkben említett fájlok  
↳ <http://user.enaplo.hu/~auth.gabor/pov/>
- Az első animáció  
↳ <http://user.enaplo.hu/~auth.gabor/pov/anim00.avi>
- A második animáció  
↳ <http://user.enaplo.hu/~auth.gabor/pov/anim01.avi>
- A harmadik animáció  
↳ <http://user.enaplo.hu/~auth.gabor/pov/anim02.avi>
- A negyedik animáció  
↳ <http://user.enaplo.hu/~auth.gabor/pov/anim03.avi>
- Az ötödik animáció  
↳ <http://user.enaplo.hu/~auth.gabor/pov/anim04.avi>