3D ábrázolás – PoVRay (6. rész)

Egy saját 3D világ felépítése és működtetése sok vesződséggel jár. Ez alatt tulajdonképpen azt kell érteni, hogy a környezetünk tele van olyan tárgyakkal és textúrákkal, amelyek rendkívül változatosak vagy összetett struktúrát alkotnak. Leginkább a növények és a távoli táj tartozik ebbe a kategóriába, mivel változatosak, kaotikusak és a legnagyobb igyekezettel sem tudjuk szabályos testekből elkészíteni ezeket. Ezen túl könnyű észrevenni, hogy a – főleg az emberek által épített – tárgyak kisebb részek ismétlődéseiből állnak.

Csalás és ámítás

A *3D* programok egyik nagy területe a filmekből is ismert digitális trükkök kivitelezése. Tulajdonképpen arról van szó, hogy a valódi világból származó képre ráteszünk egy *3D* programmal készített másik képet. Ha azonos nézőpontot, látószöget és mélységélességet állítunk be a *3D* test elkészítésénél, akkor nagyon jó minőségű végeredményt kaphatunk, amely az egyszerű szemlélőt könnyedén megtéveszti. Egy lehetetlen helyre tett tárgy még nem okoz túl nagy problémát, a tűéles kontúrok sem nagyon feltűnőek, ha nem avatott szemmel nézünk egy ilyen képet. Ellenben a nézőpont és a perspektíva (amelyet a látószög határoz meg) durva eltérése esetén azonnal felkiált minden ember, hogy valami nem stimmel ezzel a képpel.

Nézzünk példának egy tíz-tizenöt éve még javában dúló repülő csészealj őrületet, ugyanis az akkori filmtrükköket jelenleg már képesek vagyunk egy egyszerű számítógéppel elkészíteni. Ehhez keressünk egy néptelen és jellegtelen tájat, példaképpen az *1. ábrán* látható képből fogunk kiindulni, amely ideális helyszínnek ígérkezik. Ez a kép 35 milliméternek megfelelő objektívvel készült, s éppen Pécs egyik külszíni fejtése látszik a rajta a háttérben, az előtérben pedig egy lejtősebb tisztás.

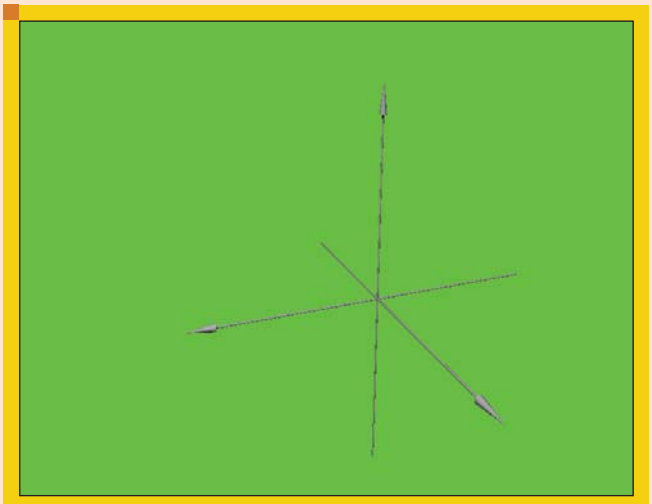
A végleges képnek illene átmennie egy egyszerűbb hihetőségi próbán, így

1. ábra Tisztás az erdőben

az csészealjunkat olyan méretűre kell készíteni, amely még nem feltűnő egy egész város vagy megye számára, viszont eléggé nagy ahhoz, hogy a „funkciójának” eleget tegyen. A legjobb pozíció ezért a *2. ábrán* látható képnek a vörösen bekarikázott része, ahova egy stilizált *UFO* is került (jelen esetben a *GIMP* jóvoltából).

A megfelelő csalás elkészítéséhez pontosan meg kell tervezni a méreteket és a távolságokat, vagyis fel kell mérnünk a fotózás pontjától az elhelyezendő tárgy irányát és távolságát.

2. ábra A repülő csészealj helye



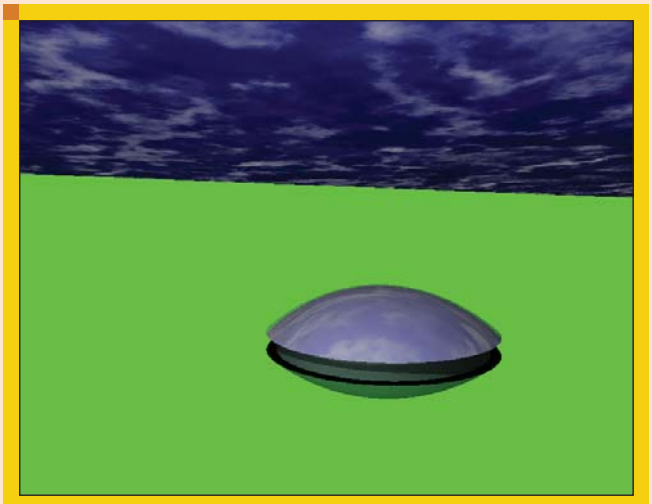
3. ábra Zöld háttéren koordináta rendszer

4. ábra A repülő csészealj helye a mezőn

Ezek után jön a munka nehezebb része, ugyanis el kell készítenünk egy repülő csészealjat a ***PoVRay*** segítségével.
Első lépésként a filmesek által használt ***bluebox*** technológiához kell folyamodnunk, ugyanis a ***PoVRay*** nem képes olyan képet készíteni, amelynél a végtelent átlátszó háttérrel helyettesítené. Sajnos olyan képet sem tud készíteni, amelynek a megadott egyéni kép van a hátterében.
A ***bluebox*** lényege, hogy az elkészített tárgyat egy matt fényű, ugyanakkor egyszínű háttérre tesszük. Ez a szín régebben a kék volt, azonban mostanában már a zöld színt szokták használni a trükkmesterek.
A mi esetünkben ez nagyon egyszerűen kivitelezhető, hiszen a háttér színét nagyon könnyen befolyásolni tudjuk, s ezen a háttérszínen semmi nem fog se megcsillanni, se tükröződni, se árnyékot vetni:

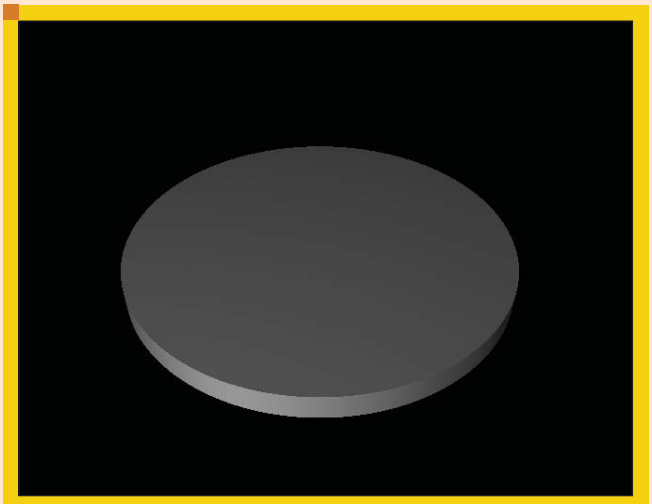
```
background{
 color Green}
```

Kezdésképpen érdemes egy szimpla koordináta rendszert készíteni, amelyet majd ráillesztünk a háttérképre (*3. ábra, pov98.pov*).
Ezen a ponton a ***GIMP*** (vagy egyéb képszerkesztő program) segítségét kell igénybe vennünk, hogy a két képből egyet tudjunk csinálni. A *3. ábrán* látható képet nyissuk meg, majd a Szerkesztés menüben válasszuk a másolást, ugyanis ennek a képnek a háttere valószínűleg az előtér fehér színe lesz, amely nekünk nem megfelelő. Ha a kép kikerült a vágólapra, akkor a ***Fájl*** menüben az ***Új*** menüpontot választva az új kép mérete pont azonos lesz a kimásolt kép méretével. Ezen dialógus ablakon belül válasszuk ki az Átlátszó kitöltést, így kapunk egy üres képet, amelynek a háttere átlátszó. Ebbe bele tudjuk illeszteni a kimásolt képünket, amelyről el kell távolítanunk a matt zöld hátteret. Ezt a ***GIMP*** szín szerinti kijelölésével tudjuk megtenni, amellyel a háttérre kattintva az összes ilyen színű képpont kijelölt lesz: ezeket egy egyszerű kivágással el tudjuk távolítani. Arra kell figyelnünk, hogy a kivágás éles átmeneteket fog eredményezni, amely leginkább a ferde vagy görbe felületek lépcsőzetességében nyilvánul meg. Ha a ***PoVRay*** oldalon próbáljuk megoldani a problémát egy kis élsimítással, akkor a ***GIMP*** nem tudja szépen kivágni az adott színt, és az átlátszó háttér nem a tárgyunk kontúrjával fog találkozni, hanem egy zöld-kontúr átmenettel. Ebből adódik, hogy a ***GIMP*** egyik elmosó

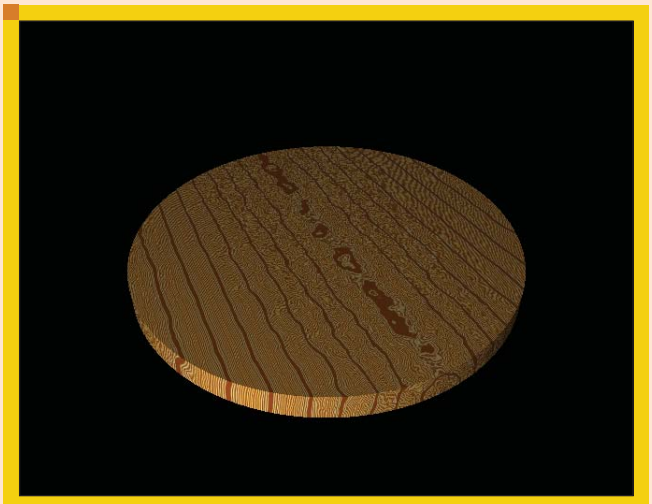


5. ábra A repülő csészealj bluebox

6. ábra A repülő csészealj a mezőn



7. ábra Az óra kezdeményei



8. ábra Az óra számlapja textúrával

eszközét kell igénybe vennünk a feladat elvégzéséhez, s megkapjuk a *4. ábrán* látható eredményt.
Ezek után már csak a csészealjat kell elkészítenünk, amelyet egyszerűen a koordináta rendszer közepébe kell tennünk. Még a mérete sem annyira fontos, mivel a ***GIMP*** képes átméretezni akkorára, amely nekünk megfelel. Az én csészealjam két gömbből indult, amelyeknek csak egy részét hagytam meg, majd ezeket ráillesztettem egy hengerre. A felső gömbcikk fényes, fémszerű textúrát kapott, az alsó pedig zöldes színt, mintha a fű tükröződne vissza rajta. A magasba pedig tettem egy felhős textúrával borított téglalapot, amelyet majd ki kell vágni a beillesztéskor, de szépen tükröződik az ***UFO*** felső részén. A kész kép az *5. ábrán* (*pov100.pov*) látható, ahol a zöld háttér felett ott egy felhős égboltrészlet is.
Néhány egyszerűbb ***GIMP*** művelet elvégzése után már csak ki kell nyomtatni a kész képet, amelyen egy kicsit többet dolgozva egyre inkább megtévesztő lehet a művünk. Bár az én kis ***UFO*** ábrám eléggé kezdetleges, ezért a 10 percnyi munkáért 15 éve egy vagyont fizettek volna a „szaklapok” (*6. ábra*).
A fenti lépéseket szinte bármilyen fényképpel megtehetjük, így csak a fantáziánk és a tudásunk szab korlátokat, ha egy nem létező dolgot a létező világban szeretnénk viszontlátni.

A virtuális világ programja

A ***PoVRay*** leíró nyelve sokban hasonlít a ***C*** vagy ***C++*** nyelvhez, s ezt már említettem a sorozat első részében. Ezen hasonlóság még arra is kiterjed, hogy a ***C*** vezérlési szerkezeteit teljesen azonos módon tudjuk használni a ***PoVRay*** állományaiban. Képesek vagyunk elágazást és ciklust készíteni, amelyek jól jönnek bizonyos testek elkészítésekor.
A leíró nyelv megismeréséhez induljunk ki egy egyszerű tárgyból: egy szimpla faliórát fogunk elkészíteni, amelyet majd bárhol fel tudunk használni, mint egy speciális objektumot. A ***PoVRay*** használatához – ha nem is feltétlen szükséges, de – jól jön egy kis programozói véna, így az óra elkészítését teljesen más alapokon kell véghezvinnünk, mint azt a grafikusok szokták művelni: programot kell írnunk.
Az egyik ***C***-ből átvett szerkezetet már ismerjük, ez az `#include` névre hallgat és arra szolgál, hogy külön állományokban tároljunk különböző dolgokat, így a fő ***PoVRay*** állományunk sokkal egyszerűbb és áttekinthetőbb lesz. Erről a direktíváról nem is lehet többet elmondani, egyszerűen csak használnunk kell, így hozzunk létre egy ***ora.inc*** nevű állományt, amely az óra alkatrészeit fogja tartalmazni:

```
#declare Ora=cylinder{
 <0.0,1.0,0.0>,<0.0,0.0,0.0>,
 ➥10.0
 pigment{
  color White}}
```

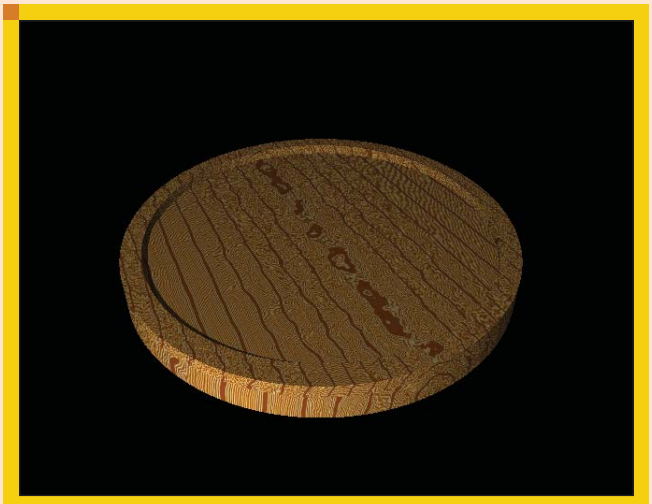
A „főprogramunk” hivatkozni fog a fenti fájlra:

```
#include "colors.inc"
#include "ora.inc"

object{Ora}
```

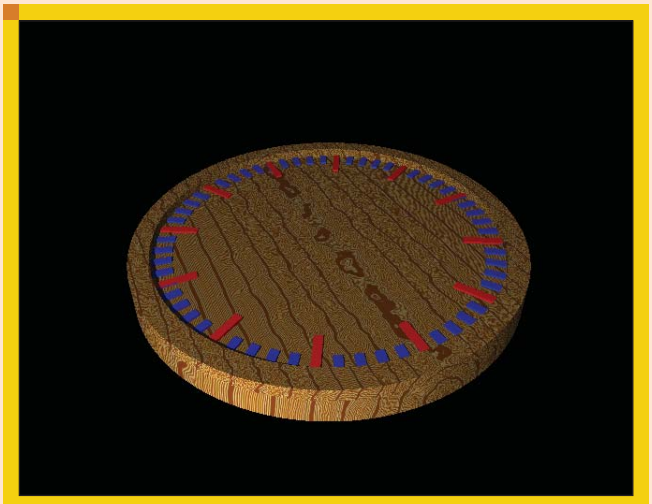
Ennek megfelelően kapunk egy szürkés korongot (*7. ábra*), mint az óra számlapja.
A ***PoVRay*** korai verziói nem tudtak változókat kezelni, bár ezt a hiányosságot nagyon hamar pótolták. A változók nagyon hasznosak tudnak lenni, ha egy-egy összetett tárgy egyes paramétereit szeretnénk módosítani, ezért célszerű az óra számlapjának textúráját egy ilyen változóba tenni, amelynek kezdőértéket is adunk:

```
#include "woods.inc"

#declare Szamlap=
 texture{
  T_Wood31
  scale 10.0}

#declare Ora=
 cylinder{
  <0,1,0>,<0,0,0>,10
  texture{
   Szamlap}}
```

Az óra számlapja egy fa erezetéhez lesz hasonló (*8. ábra*), kivéve, ha a `Szamlap` változónak a főprogramban más értéket adunk és `#declare` helyett `#macro` direktívát használunk, de erről egy kicsit később még lesz szó. Vegyünk egy karimát az órához, amelyre majd a fedő üveget fogjuk tenni, s nevezzük ezt `Karima`-nak. Fontos, hogy a változókat úgy nevezzük el, hogy ezzel se önel-



9. ábra Az óra számlapja és karimája



10. ábra Az számlap jelölésekkel

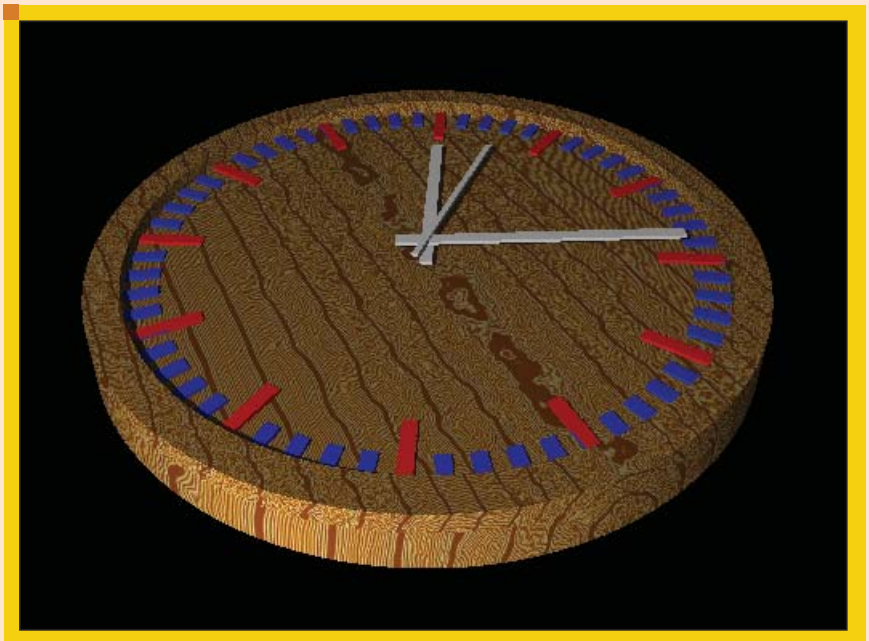
lentmondásba, se más állomány változóival ne keveredjünk. S ha a számlapon kívül egy kis karimát is szeretnénk az óránknak, akkor már lesz egy kis problémánk az elnevezésekkel, így jobb ennek elébe menni:

```
#declare OraSzamlapTexture=
 texture{
  T_Wood31
  scale 10.0}

#declare OraKarimaTexture=
 texture{
  T_Wood31
  scale 5.0}

#macro OraSzamlap()
 cylinder{
  <0,1,0>,<0,0,0>,10
  texture{
   OraSzamlapTexture}}
#end

#macro OraKarima()
 difference{
  cylinder{
  <0,1.5,0>,<0,1,0>,10
  texture{
   OraKarimaTexture}}
  cylinder{
  <0,1.6,0>,<0,0.9,0>,9
  texture{
   OraKarimaTexture}}}
#end

#macro Ora()=
 union{
  OraSzamlap()
  OraKarima()}
#end
```

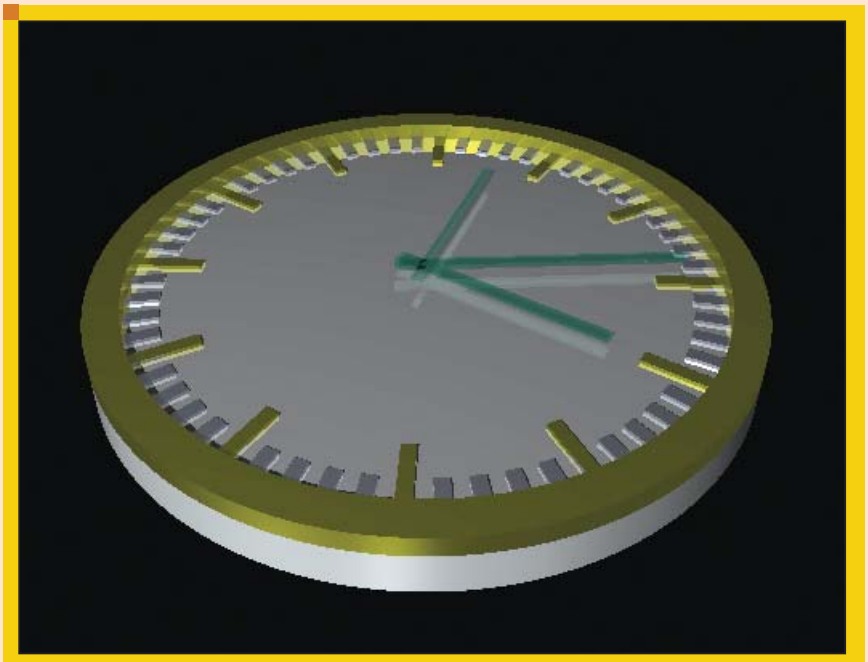
A fájl elérési útját mindenféleképpen vegyük bele az elnevezésekben (jelen esetben ez az `Ora`), amelyhez fűzzük hozzá az alkatrész nevét (például `Szamlap`), majd a *PoVRay* szerkezet nevét (például `Texture`). Ha ehhez tartjuk magunkat, akkor nehezen kerülünk névütközésbe, hiszen az `OraSzamlapTexture` egyértelműen meghatároz egy elemet, amelyből csak egy van. Egy új elem is megjelent, mégpedig a `#macro` direktíva, amely egy függvényhíváshoz hasonlít a leginkább. Egyelőre nincs paramétere, de majd azzal is találkozunk kicsit később.

A `#declare` és a `#macro` között abban van különbség, hogy az előbbi statikus viszonyt jelent, a PoVRay egyszerűen csak beilleszti az elkészített tárgyat a megadott paraméterekkel, míg a `#macro` esetén újra feldolgozza annak törzsét. Ennek akkor van jelentősége, ha meg szeretnénk változtatni a tárgy textúráját: `#declare` esetén nem dolgozza fel a PoVRay újra a megadott tárgy „programját”.

Az óra számlapjára rá kell tennünk pontosan 12 jelet, amely az egész óráknak felel meg. Ezt megcsinálhatjuk nyers erőből is, egyenként létrehozva őket, de akár okosan is: ciklust használva. A ciklus arra lesz jó, hogy az órát jelző testet lemásoljuk 12 példányban és 30 fokonként elforgassuk:



11. ábra Az elkészült óra, alapértelmezett textúrákkal



12. ábra Az elkészült óra, fémes textúrával

```
#macro OraOraJel(ora)
 box{
  <9,1,-0.2>,<7,1.1,0.2>
  texture{
   OraOraJelTexture}
  rotate <0,30*ora,0>}
#end

#macro Ora()
 union{
  OraSzamlap()
   OraKarima()
  #local szamol=0;
  #while (szamol<12)
   OraOraJel(szamol)
   #local szamol=szamol+1;
  #end}
```

A `#declare` és a `#local` között annyi a különbség, hogy a `#local` hatóköre a *PoVRay* blokkok ({ és }) között marad, míg a `#declare` globális hatókörrel rendelkezik. A ciklusváltozót – amellyel elszámolunk tizenkettőig – célszerű helyi változóként kezelni, így nem kerülünk ütközésbe egy másik ciklussal, ha például több órát szeretnénk kitenni a falra.
Az `OraOraJel` makrónak már van egy paramétere, mégpedig egy `ora` nevű változó, amelyet megszorozva harmiccal elforgathatjuk 30-30 fokkal az egész órákat jelző téglalapot. A sikeren felbuzdulva tegyük ki a perceket jelölő kis téglalapokat is (*10. ábra*).

Ezeken túl már csak az óra mutatói vannak vissza, ezeket paraméterekkel meg kell tudnunk adni, s figyelnünk kell arra, hogy hibás értéket ne lehessen megadni az óránknak. Erre szolgál az feltételtől függő elágazás, amellyel korrigálni tudjuk a hibás értékeket. Először is tegyük ki az óramutatót és vizsgáljuk le, hogy 1 és 12 között van-e a megadott érték:

```
  #if (OraOra<1)
   #declare OraOra=1;
  #end
  #if (OraOra>12)
   #declare OraOra=12;
  #end
  OraOraMutato(ora)
```

A perc- és a másodpercmutató azonos módszerrel kitehető, így tulajdonképpen kész is van a falióránk (*11. ábra*), esetleg egy védőüveget tehetünk rá, de ez a virtuális világban nem fontos.
Csak annyi van vissza, hogy az órát saját textúrával lássuk el, ehhez a főprogramban át kell írnunk a megfelelő változók értékeit:

```
#declare OraSzamlapTexture=
 texture{
  pigment{
   color White}
  finish{
   reflection 0.9
   roughness 0.01
   specular 0.4}}

#declare OraKarimaTexture=
 texture{
  Gold_Texture}

#declare OraOraJelTexture=
 texture{
  Gold_Texture}

#declare OraPercJelTexture=
 texture{
  Silver_Texture}

#declare OraOraMutatoTexture=
 texture{
  Green_Glass}

#declare OraPercMutatoTexture=
 texture{
  Green_Glass}

#declare
OraMasodpercMutatoTexture=
 texture{
  Green_Glass}

#declare OraOra=8;

Ora()
```

A következő részben egy virtuális világban a mozgás bírjuk a tárgyakat és a kamerát, illetve MPEG videót készítünk a *PoVRay* kimenetéből.

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/