PoV-Ray ismeretek (2. rész)

Ebben a részben az egyszerű testek meghatározását, a testek közötti halmazműveleteket és a különféle transzformációkat tekintjük át.

Bemutatjuk az egyszerű testek (sík, gömb, téglatest, kúp, csonka kúp, henger és tórusz) meghatározását, a testek közötti halmazműveleteket (CSG – Constructive Solid Geometry), valamint a különféle átalakításokat (tranformációkat). A jelenet megadásához a PoV-Ray néhány alapvető testet bocsát a rendelkezésünkre, amelyekből összetett testek képzésére nyílik lehetőségünk – így például forgástesteket, Bezier-felületeket is létrehozhatunk.

Amint sorozatunk első részében már láthattuk, egy *gömb* meghatározása egy koordinátával és a gömb sugarának megadásával történhet, a következő formában:

```
sphere {
        VEKTOR
        SUGAR
}
```

A megadott `VEKTOR` a gömb középpontját jelöli ki, a valós `SUGAR` pedig a gömb sugarát határozza meg.

Az előző részben ugyancsak használtunk *sík* felületet. Ezt a következő módon hozhatjuk létre:

```
plane {
        VEKTOR, VALÓS
}
```

Itt az első érték egy vektor, ami merőleges a síkra (a sík normálvektora), az ezt követő valós szám pedig azt határozza meg, hogy a síkot hány egységgel kell eltolni a normálvektor által kijelölt tengelyen.

Egyszerű téglatestet a `'box'` kulcsszóval hozhatunk létre, mely a következő értékekkel bírhat:

```
box {
        VEKTOR1
        VEKTOR2
}
```

Az első vektor itt a nullponthoz közelebb eső bal alsó csúcspont helyvektorát adja meg, míg a második vektor az előbbi csúcsból induló testátló másik végén lévő csúcspont helyvektora.

Egy testátló két végének megadásával a téglatestet egyértelműen meghatároztuk.

Kúp és *csonkakúp* megalkotása a következő sorokkal lehetséges:

```
cone {
        VEKTOR1, SUGAR1
        VEKTOR2, SUGAR2
}
```

A két vektorral meghatározzuk a kúp tengelyének végeit, míg a vektorok után megadott két valós szám a csonka kúp végeinek sugarát szabja meg. Szabályos kúpot úgy hozhatunk létre, hogy az egyik sugarat 0-ként adjuk meg.

```
cone {
        <0,0,0>, 2
        <0,4,0>, 0
}
```

Ha azt szeretnénk elérni, hogy a csonka vagy a szabályos kúp mindkét vége nyitott legyen, akkor a fenti két sor után az `'open'` kulcsszót is használnunk kell. *Hengert* a következő módon adhatunk meg.

```
cylinder {
        VEKTOR1
        VEKTOR2
        SUGAR
}
```

A két vektor szerepe megegyezik a fentebb tárgyalt kúpnál szereplő vektorokéval, vagyis ezekkel adjuk meg a henger két végét. Tekintettel arra, hogy a henger minden keresztmetszete azonos sugarú, itt csak egyetlen sugármegadás szükséges, mely a henger tengelye mentén állandó lesz. Itt is használható az `'open'` kulcsszó, hatására a henger mindkét vége nyitott marad.

A *tórusz* talán nem ismeretlen fogalom azok körében, akik már használtak valamilyen 3D-modellező programot. A többiek pedig könnyedén képet alkothatnak maguknak erről a testről, ha egy lyukas közepű fánkra vagy hullahopp-karikára gondolnak.

A PoV-Rayben tóruszt a következő utasításokkal hozhatunk létre:

```
torus {
        NAGYOBB_SUGAR, KISEBB_SUGAR
}
```

A fenti listában a NAGYOBB_SUGAR valós érték határozza meg a tárgy tényleges sugarát, míg a KISEBB_SUGAR a cső sugarát adja meg.

Ezzel végére is értünk az egyszerű testek sorának.

Következzenek a különféle *átalakítások* (transzformációk) formai követelményei! Három alapvető átalakításról beszélhetünk, ezek minden 3D-modellezéssel kapcsolatos programban megtalálhatók. Az első ilyen az eltolás, amely nem csupán tárgyakra értelmezhető, hanem természetesen kamerára és fényforrásokra is. Formája a következő:

```
translate {
        VEKTOR
}
```

A VEKTOR az eltolás vektora.

Szintén egy-egy vektorral adható meg a következő két átalakítás: a méretezés (scale) és a forgatás is.

```
scale {
        VEKTOR
}

rotate {
        VEKTOR
}
```

A fentiekre álljon itt példaképpen ez a rövid lista, amiben mindhárom átalakítás fellelhető.

```
box {
< 0, 0, 0 >, < 1, 1, 1 >
translate { < 2, 2, 2 > }
rotate { 20 * x }
scale { < 1, 1.2, 2 > }
}
```

Ha nincs szükségünk az átalakítási vektorok minden elemére, használhatjuk a PoV-Raybe beépített egységvektorokat is, amint azt a forgatásnál láthattuk.

Az alábbiakban erre a három műveletre alapozva létrehozunk egy *kereket*. A kerék abroncsát egy tórusz fogja képezni, a küllőket pedig

hengerek alkotják. A kerékagyat és a tengely csatlakozási helyét két torzított gömbből alakítottam ki. Az elkészült képet a CD-mellékleten kerek.tga néven találhatjuk meg.

```
// Makróként megadjuk a küllőt, hogy a
// későbbiekben csak a forgatással és eltolással
// kelljen foglalkozni.
#declare kullo=cylinder {
        <0, 0, 0>
        <0, 0.8, 0>
        0.1
        pigment { color rgb <0.2, 0.2, 0.5> }
}

// Szórt fény
global_settings {  ambient_light 0.9 }

// Kamera
camera
{
  angle 40
  location  <-2.0 , 2.0 ,-2.0>
  look_at   <0.0 , 0.0 , 0.0>
}

// Direkt fényforrás
light_source
{
  0*x
  color rgb <1,1,1>
  spotlight
  translate <20, 40, -20>
  point_at <0, 0, 0>
  radius 10
  falloff 20
}

// Alaplap
plane {
       y, 0
       pigment { color rgb <0.6, 0.6, 0.6> }
}

// Kerékabroncs
torus
{
  0.8,
  0.15
  pigment { color rgb <0.2, 0.5, 0> }
  translate 0.25*y  // <dX dY dZ>
}

// 6 küllő a kerékhez
object { kullo
       rotate <90, 0, 0>
       translate y*0.25
}
object { kullo
       rotate <90, 60, 0>
       translate y*0.25
}
object { kullo
       rotate <90, 120, 0>
       translate y*0.25
}
object { kullo
       rotate <90, 180, 0>
       translate y*0.25
}
object { kullo
       rotate <90, 240, 0>
       translate y*0.25
}
object { kullo
       rotate <90, 300, 0>
       translate y*0.25
}

// Kerékagy
sphere {
       <0, 0, 0>
       0.4
       pigment { color rgb <0.8, 0.3, 0.2> }
       scale y*0.3
       translate y*0.25
}
sphere {
       <0, 0, 0>
       0.2
       pigment { color rgb <0.4, 0.15, 0.1> }
       scale y*1.2
       translate y*0.25
}
```

Az alapvető átalakításokat áttekintve, ebben a hónapban nem maradt más hátra, mint a *CSG-műveletek* megértése és használatuk elsajátítása. A CSG – azaz a már említett Constructive Solid Geometry – a legkönnyebben úgy érthető meg, hogyha az egyes testeket térbeli pontok halmazának tekintjük. Ebben az esetben a műveletek egyszerű halmazműveletek, nevük is ennek megfelelő: unió (union), különbség (difference) és metszet (intersection). A PoV-Rayben létezik egy negyedik művelet is, ezt összekapcsolásnak (merge) nevezhetjük, és feladatát tekintve nagyon hasonlít az unió-művelethez.
A CSG segítségével rendkívül összetett alakzatokat is létrehozhatunk, de talán felmerül a kérdés, hogy miért nem elégséges egyszerűen csak egymás mellé helyezni azokat a tárgyakat, amelyekből az összetett objektum áll? Ha az objektumokat csupán egymás mellé helyezzük, akkor annak minden egyes apró elmozdulásakor minden tárgyat pontosan ugyanannyival kell arrébb tennünk, amennyivel maga is elmozdult – és ugyanez lesz igaz a méretezésre és a forgatásra is! Ezenkívül ha tárgyunk minden része azonos anyagból készült, felesleges munka lenne ezt az anyagot minden rész számára megadni, egyszerűbb a részeket egymáshoz kapcsolva kezelni. A CSG-vel létrehozott tárgyak szerkezetét faszerkezetként ábrázolhatjuk, ahol a levelek az egyes részobjektumok, míg a fa belső csomópontjai az egyes műveletek. Elsőként az egyes műveletek formai követelményeit vizsgáljuk meg, majd létrehozunk egy összetett objektumot, amelynek szerkezete a következő ábrán látható. Objektumunkat nevezzük – a nagyfokú hasonlóság miatt – mondjuk *hamutartónak*.
Az első művelet legyen az unió. Ennek formai követelménye az alábbi:

```
union {
       OBJEKTUM_1
       OBJEKTUM_2
       ...
       ...
       OBJEKTUM_N
}
```

2. lista

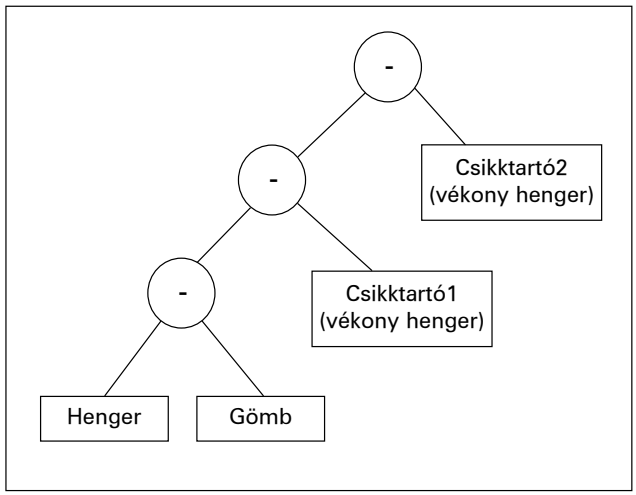
```
**
// A hengerből kivonjuk a gömböt
#declare Hamu_alja=difference {
                    cylinder {
                            <0,0,0>
                            <0,0.5,0>
                            0.9
                    }
                    sphere {
                            <0,0.82,0>
                            0.8
                    }
            }

// Az előbbi művelet eredményéből kivonjuk
// az egyik kis fekvő hengert
#declare Alja_egylyukkal=
        difference {
                object { Hamu_alja }
                cylinder {
                        <0,0.3,0>
                        <0,2.8,0>
                        0.1
                        rotate <90, 90, 0>
                        translate y*0.5
                }
        }

// Egylyukú hamutartó - másik kis fekvő henger =
// = készen van a hamutartó.
difference {
        // Hamutartó alja - egyik csikktartó
        object { Alja_egylyukkal }
        cylinder {
                <0,0.3,0>
                <0,2.8,0>
                0.1
                rotate <90, 270, 0>
                translate y*0.5
        }
        // Adjunk neki kék színezetet
        pigment { color rgb <0.3, 0.3, 1> }
}
*
```

A fentiekből kitűnik, hogy ez a művelet nemcsak két objektumon értelmezhető, hanem egyidejűleg több objektumot is eggyé olvaszthatunk. Az egyes műveletek értékei összetett objektumok is lehetnek, ahogyan azt a későbbi forráslistákon láthatjuk. Tehát a különbségképzés és a metszet a következőképpen néz ki:

```
difference {
        OBJEKTUM_1
        OBJEKTUM_2
        ...
        ...
        OBJEKTUM_N
}
intersection {
        OBJEKTUM_1
        OBJEKTUM_2
        ...
        ...
        OBJEKTUM_N
}
```



Hamutartó szerkezete CSG-fa ábrázolásban

Mivel a különbségképzés eredménye számos objektum esetében nehezen képzelhető el előre, ezt a műveletet célszerű egyszerre csak két tárgyon elvégezni és a kapott eredményt használni a továbbiak során. Utolsó műveletként az összekapcsolást (merge) tárgyaljuk, amely az uniótól annyiban tér el, hogy a tárgyak belső felületei eltűnnek. Ez főként átlátszó tárgyak létrehozásakor lehet fontos, hiszen például egy domború lencsében nem szép, ha úgy tűnik, mintha a végső forma két félből lenne összeragasztva. Formája az előbbiek alapján könnyen érthető:

```
merge {
        OBJEKTUM_1
        OBJEKTUM_2
        ...
        ...
        OBJEKTUM_N
}
```

A következő listákban kétféle módon határozzuk meg a hamutartó formáját: elsőként egyszerre végezzük el az összes műveletet (1. lista, mely a 15. CD Magazin/PoV-Ray könyvtárában található), majd a második megoldás során az egyes részeredményeket különkülön objektumként kezeljük (*2. lista*). Így végezetül ugyanahhoz az eredményhez jutunk.

Ezzel lezárjuk ezt a részt, a következőkben pedig olvasóinkat az objektumok létrehozásának bonyolultabb módszereivel ismertetjük meg, ezt követően térünk rá a különböző anyagokkal és mintázatokkal való ismerkedésre.

Addig is remélem, hogy mindenkinek kellemes perceket szerez a PoV-Ray és az emberi alkotóerő találkozása.

Fábián Zoltán (dzooli@freemail.hu, dzooli@yahoo.com) 23 éves, jelenleg programozóként dolgozik. Szabadidejében szívesen kirándul, túrázik. Emellett szeret rajzolni, érdekli a 3D grafika és a Linuxszal kapcsolatban minden olyan program és programnyelv, amit még nem ismer vagy nem próbált ki.