A Blender szabadfejlesztésű 3D modellező alkalmazása a látványtervezésben

ELEK Tamás

SzIF, Diplomamunka
2001. december 9.

1 Bevezető

„A Fény átment Isaac Newton laboratóriumán, és többé nem nézett ki ugyanúgy."

(Joel Achenbach, National Geographic)

Newton volt, aki elsőként mutatta meg egy prizma segítségével, hogy a fehér fény tartalmazza az összes színt. Ezzel nagy lépést tett az emberiség, hogy közelebb jusson a fény vizsgálatában. Elődjei, például Arisztotelész, Platón, Leonardo, mind újabb és egyre jobb megközelítést adtak, hogy megértsük, mi is a fény, de még az utódok sem tudják megmondani teljes biztonsággal, mi is az, amit mi fénynek nevezünk. A fény viselkedését kutatják csillagászok, fizikusok, festők, fényképészek, megfigyeléseik eredményét pedig képletekbe foglalva az informatika teszi elérhetővé mindenki számára. Ha ezt a tudást vegyítjük a számítógépes grafika technikai megoldásaival, akkor elérkezünk a látványtervezés tudományához.

A látványtervezés különböző eredményeivel gyakran találkozunk mindennapi elfoglaltságaink közben. Látványos mozitrükkök, figyelemfelkeltő reklámok, valósághű rajzok soha nem létezett tárgyakról úgy futnak el a szemünk előtt, hogy nincs is időnk belegondolni, mikor látjuk a valóságot, és mikor látjuk a valóságnak csak egy utánzatát. Talán azt sem látjuk, hogy napról napra jobb megoldások, hatékonyabb eszközök állnak a rajzolók rendelkezésére. Az újabb trükköknek és találmányoknak szinte csak technológiai akadályai vannak, az emberi találékonyság mindig több lépéssel előbbre jár a megvalósíthatóságot segítő számítógépek műszaki paramétereinél. Minden műszaki megoldás újabb ötleteket szül, ami még jobb műszaki megoldásokat igényel. Ennek a körfogásnak az eredménye az a műszaki fejlődés, amit - nem csak - a számítástechnika terén látunk. A fejlődésnek egyik legjobb bizonyítéka lehet az is, hogy a hetvenes évek sikerfilmjét, a Csillagok háborúját újra kiadták a közelmúltban. George Lucas ekkor már tudott olyan látványtechnikai megoldásokat alkalmazni, amelyek elképzelhetetlenek voltak az első forgatásoknál.

Diplomamunkám témája egy olyan látványtervezési rendszer bemutatása, ami csupán szabadterjesztésű programokból épül fel. Hogy ennek milyen jelentősége van,

arra akkor jövünk rá, ha utánanézünk, mennyibe kerül egy-egy látványtervezéssel foglalkozó szoftver. Ha ezek után még mindig ezzel a területtel szeretnénk foglalkozni, akkor három dolgot tehetünk: megvásároljuk a szoftvert, illegálisan használjuk a szoftvert vagy keresünk egy olcsóbbat, esetleg ingyeneset. A három lehetőség közül a harmadikat szeretném bemutatni a következő oldalakon.

A látványtervezés legolcsóbb megoldása egy Blender nevű program lehet. Ez a program egy holland székhelyű cég, a Not a Number terméke. A legtöbb ismert operációs rendszer alatt működik, így az ingyenes Linux alatt is. Diplomamunkám során további programokkal is dolgozom, melyek kiegészítik vagy segítik a modellezési munkát a Blenderrel. A teljesség kedvéért:

Mandrake Linux 8.1	- operációs rendszer
Blender 2.03	- 3D modellező
Python 2.0	- programnyelv
Gimp 1.2.2	- rasztergrafikus program
Star Office 6.0	- szövegszerkesztő

Ezek mind szabad terjesztésű vagy nyilvános kódú szoftverek, magukban hordozva a szabadterjesztésű programok minden jegyét, mint például a dinamikus fejlődést. Egy zárt kódú program, aminek az egymást követő verziói, frissítései közt hónapok esetleg évek is eltelhetnek, nem tudja követni a technikai változásokat és igényeket. Míg egy szabadon terjesztett program különböző verziói akár havonta megjelenhetnek. Természetesen fontos szempont, hogy a kompatibilitás megmaradjon a verziók között. A Blender 2.03 a 2001-es év elején jelent meg. Mikor ezeket a sorokat írom, már a 2.23-es verzió is megjelent, mégis minden, ami ebben a munkában szerepel, az a mai verziókban is ugyanúgy működik.

A fent említett programok kiemelt szabad forrású programok, olyanok, amelyeket egy kezdeményezés, úgynevezett Open Source Initiation (Szabad Forrás Kezdeményezés) ellenőriz, hogy megfelel-e az általuk meghatározott definíciónak. Csak ezek a programok viselhetik az OSI Certified (OSI ellenőrzött) jelet. A definícióban nemcsak a forrás közlésének feltételei vannak megszabva, hanem külön pontokban van

szabályozva az is, hogy semmiféle diszkriminációt nem alkalmazhat emberek vagy embercsoportok ellen. Az, hogy hogyan válik egy program a fejlődése során szabadforrásúvá vagy szabadterjesztésűvé az nem ennek a munkának a témája. A Blender kezdetben pénzbe került. Mára a szponzorok lehetővé tették, hogy mindenki szabadon hozzáférjen a programhoz, de a forráskód nem publikus, így ez "csak" egy szabad terjesztésű program.

Meg kell említeni azokat a közösségeket, amelyek körbeveszik az ilyen programokat. A Blender Közösség, a Blender honlapján tartják egymással a kapcsolatot, új információkat szereznek, segítenek egymásnak és nem utolsó sorban részesei a fejlesztésnek észrevételeikkel. Mióta figyelemmel kísérem a Blender fejlődését, a közösség regisztrált tagjainak száma megduplázódott. Számuk e dokumentum írásakor több mint 30 ezer.

Munkám három fő részre osztható. Az egyik a Blenderrel való munkát mutatja be konkrét példán keresztül, melynek végeredménye egy beltér valósághű ábrázolása. Második részben egy általam írt bővítés Blenderhez. Egy rövid parancssorozat (szaknevén: script), ami hasznos segédeszköz lehet a tárgyak modellezése során. Végül pedig egy egyszerű példán keresztül szeretném bemutatni egy végeredmény külső alkalmazásokban való felhasználását.

Célom nem az, hogy bebizonyítsam, a Blender a legjobb látványtervező program, hanem, hogy bemutassam a Blender lehetőségeit, előnyeit, hatékonyságát, és ezzel választási lehetőséget adjak azoknak, akik a látványtervezésben mélyebben el szeretnének merülni.

2 Tartalomjegyzék

1 BEVEZETŐ 2

2 TARTALOMJEGYZÉK 5

3 A BLENDER 6

3.1 A FELHASZNÁLÓI FELÜLET 7

Tájékozódás a térben 9

Az objektumok 9

3.2 MODELLEZÉSI TECHNIKÁK 13

Forgástestek és tömbműveletek 14

Render 16

Boolean-műveletek 17

Loft 19

Egyéb tömbműveletek 21

Részecskerendszerek 23

3.3 RENDERING - KÉPLEKÉPZÉS 24

Radiosity 24

Textúrák 27

Fények 31

Panorámakép 32

3.4 A BLENDER NEM CSAK 3D MODELLEZŐ 33

4 SCRIPTEK A BLENDERBEN 35

5 PYTHON 39

6 BEFEJEZÉS 41

7 FELHASZNÁLT IRODALOM 42

8 MELLÉKLET 43

8.1 A CD MELLÉKLET 43

8.2 KÉPEK 44

8.3 PREVIEW.PY – BLENDER SCRIPT 45

8.4 GALLERY.PY – PYTHON PROGRAM 47

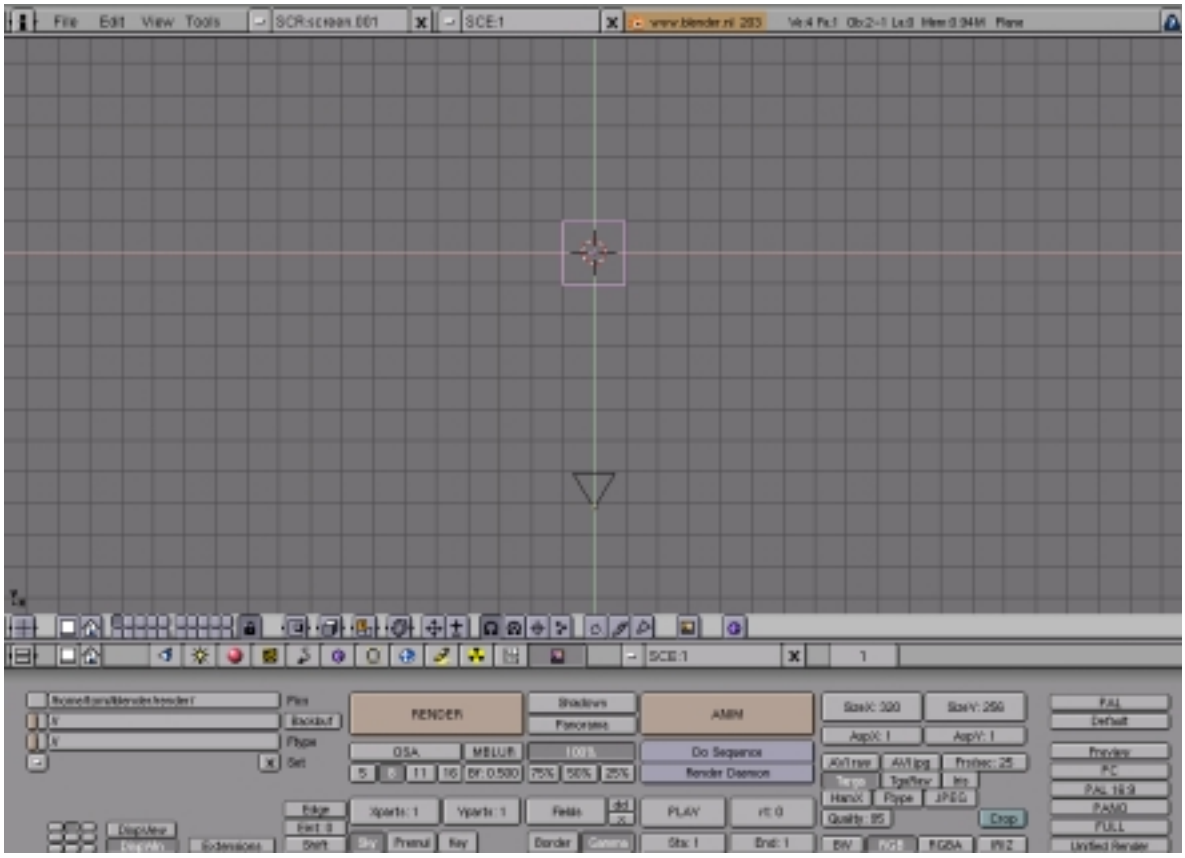
3 A Blender

Főiskolai tanulmányaim alatt a látványtervezés témakörben a 3D Studio Max programmal sikerült mélyebben megismerkednem. Ez egy kiváló program arra, hogy valaki megtanulja a látványtervezés különböző fogásait. Meggyőződésem, hogy ez egyelőre a legelterjedtebb program ebben a témakörben, ezért a Blender bemutatása közben ezzel hasonlítom össze a program tulajdonságait. Három szembetűnő különbséget már most megemlíthetünk. Az egyik az ár. A Blender ingyenes, a 3DS Max ára 3990 amerikai dollár. A másik különbség, hogy míg a 3DS Max 2 telepítő CD-ből áll, addig a Blender telepítő programja 2 Megabájt. Ebből nem sok mindent lehet következtetni, mindössze azt, hogy a telepítés és a hordozhatóság sokkal egyszerűbb a Blender esetében. A hordozhatósághoz tartozik a harmadik nagy különbség: a Blender a legtöbb operációs rendszer alatt működik.

Felhasználási területük ugyanaz: modellezés és látványtervezés, melynek végeredménye lehet egy állókép, képek sorozata (animáció) vagy akár egy megrajzolt háromdimenziós objektum, amit importálni lehet más alkalmazásokba, akár játékprogramok szereplője lehet. A bájtokban mért különbségnek valahol fel kell tűnnie, én úgy gondolom, hogy szolgáltatásokban többet nyújt a 3DS Max, amit a Blender-felhasználó az ötletességével tud ellensúlyozni.

Szinte minden programnál központi kérdés, hogy milyen fájlokkal tud dolgozni. A sajátján kívül három másik (dxf, wrml, Videoscape) fájlformátumot tud olvasni és írni. Léteznek bővítések, melyek plug-in -ok segítségével más 3 dimenziós objektumokat leíró fájlformátumokkal is dolgozhatnak (akár 3ds). Munkánkat teljes mértékben mégiscsak saját fájlformátumában (blend) tudjuk elmenteni. Érdekesség, hogy akár külső fájlokat is tudunk "belepakolni" a készülő dokumentumunkba. Ha például háttérnek egy képet használunk, akkor nem kell nyilvántartanunk, hogy fájlunk szállításakor milyen képeket kell vele mellékelni, elég lehet csak egy Blender-fájlt hordozni, ami tartalmazza a használt egyéb fájlokat. Ez persze megnöveli a fájl méretét, de utólag ezek a fájlok lekapcsolhatók róla.

Ebben a fejezetben először a Blender kezelői felületét szeretném bemutatni, majd néhány modellezési technikán keresztül szemléltetem a rajzolás technikáit, végül a megjelenítés, a textúrák tárgyakhoz rendelése, a fények, kamerák elhelyezése következik.



1. ábra A felhasználói felület

3.1 A felhasználói felület

Első ránézésre a felhasználói felület kialakítását gondolhatjuk a Blender fő hátrányának. Azonban figyelembe kell vennünk, hogy ezzel a kialakítással találkozunk, bármely platform (operációs rendszer) alatt futtatjuk a programot. A fő vezérfonal a felhasználói felületnél az volt, hogy hatékonyan és gyorsan lehessen dolgozni. A fejlesztők ajánlása alapján egyik kezünket a billentyűzeten, a másikat az egéren tartsuk. Minden parancsot el tudunk érni a billentyűzetek segítségével, és itt említhetjük meg azt a hátrányt, ami a felhasználói felületre vonatkozik: vannak műveletek, ami-

ket csak a billentyűzetről érhetünk el, nincs kihelyezett gomb minden egyes műveletre. Ez a kezdő felhasználókat rettentheti el, de haladó szinten rájövünk, hogy valóban ez a módja a leghatékonyabb munkának. A parancsok billentyűzetről való indítása megoldható az AutoCAD és a 3DS Max programokban is, amit kifejezetten ajánlanak is, ha valaki meg akarja gyorsítani és könnyíteni a munkáját. A továbbiakban, ha valamilyen műveletet írok le, akkor kapcsos zárójelben ott szerepel majd egy jel, a megfelelő billentyűkombinációra utalva. A jel a Blender dokumentációihoz igazodva így néz ki: [_KEY], ahol az alulvonás helyén a megfelelő billentyűkombináció áll. Például a dokumentum mentése az F2-es funkcióbillentyű [F2KEY], vagy a gyorsmentés [Ctrl-SKEY].

A gyakori mentésre nagy szükség van, mert a műveleteknek nincs visszavonási lehetőségük. Ajánlott menteni egy-egy olyan művelet előtt, ami lényegben befolyásolja az objektumot, és egyszerű törléssel nem állíthatjuk vissza az előző állapotot.

Az 1. ábra mutatja az első indítás képét. Középen a legnagyobb a 3D-ablak, minden rajzolást, szerkesztés itt végzünk majd. Alatta találhatjuk a Gomb-ablakot, ahol a szerkesztési funkciókat érhetünk el, valamint a jelenet beállításait módosíthatjuk. Minden ablaknak van fejléce, aminek a bal-első elemével választhatjuk ki, hogy milyen funkcióban szeretnénk használni az adott ablakot. A munkafelület tetején egy menüsort látunk. Valójában ez is egy ablak. Egy olyan ablak, aminek csak a fejlécét látjuk. Az ablakok közti határvonalat "megfoghatjuk" az egér bal gombjával és áthelyezhetjük, átméretezve ezzel az ablakot. Jobb gombbal kattintva a határvonalra ablakokat oszthatunk fel vagy vonhatunk össze. A műveletek mindig az aktuális ablakra vonatkoznak, amely felett az egér van.

A Blender a háromgombos egérre lett fejlesztve. A középső gombra néhány - környezetfüggő - műveletet definiáltak. Két gombos egér esetén a harmadik gomb helyettesíthető a bal oldali Alt billentyű és a bal egérgomb kombinációjával.

Tájékozódás a térben

Először lássuk, milyen műveletek végezhetünk az egérrel a 3D-ablakban. Bal gombbal kattintva tudjuk elhelyezni az úgynevezett 3D-kurzort. Ez a térbeli referenciapont nem található meg a 3DS Maxban. A modellépítés és szerkesztés közben a műveletek referenciapontja lehet ez a kurzor. A dokumentum további részeiben többször találkozunk majd vele.

A középső gomb nyomvatartásával és az egér mozgatásával a nézőpontot forgathatjuk el. Hasznos lehet akkor, amikor egy objektumot valamilyen nem szabványos nézetből szeretnénk látni. Könnyen visszatérhetünk elöl-, fölül- illetve oldalnézetre, ha a fejléc megfelelő ikonjára kattintunk, vagy a NumPad billentyűk segítségével ([PAD3], [PAD1], [PAD7]). A további NumPad billentyűk is a tájékozódást segítik. Ha végképp eltévednénk a térben, akkor a Home billentyű lenyomásával az összes látható objektumot az ablak látószögébe hozhatjuk. A középső gomb szolgál a mozgásra a 3D-ablakban. A Ctrl-középső gomb kombinációjával és az egér mozgatásával közelíthetünk, távolodhatunk (zoom), a Shift billentyű segítségével pedig oldalirányú mozgást végezhetünk (pan).

A jobb oldali gomb az objektumok kijelölésére szolgál. A Shift nyomvatartása közben további objektumot vehetünk fel a kijelöltek listájába, vagy szüntethetjük meg a kijelölését. Első indításkor a 3D-ablak közepén egy kijelölt négyzetet látunk, színe lila az alapértelmezett drótháló nézetben. Ha megszüntetjük a kijelölést, akkor feketére változik.

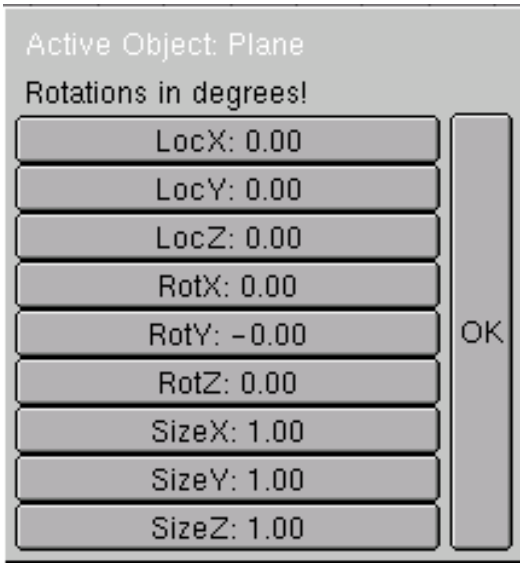
Az objektumok

Minden elemet a térben objektumnak nevezek. Könnyen tudunk objektumokat hozzáadni a jelenetünkhöz. Ha az egeret egy 3D-ablak felett tartjuk, a Space megnyomásával egy előugrómenüben választhatunk olyan menüpontokat, melyek megtalálhatók a felhasználói felület más részein is, de itt találhatunk olyan funkciókat is, melyeket csak ennek a menünek a segítségével érhetünk el. Ilyen az első menüpont is, az "ADD", mellyel primitíveket tudunk a jelenthez adni. Az objektum beillesztési

pontja a fent említett 3D-kurzorhoz igazodik. Amennyiben ez lehetséges, beillesztés után az objektum szerkesztőmódban (Edit mode) lesz. A szerkesztőmódban érhetjük el külön-külön az objektumon belüli pontokat, éleket és felületeket, majd ezeket tetszés szerint módosíthatjuk. Ebből következik, hogy például kamera vagy lámpa beillesztésénél nincs szerkesztő mód. A szerkesztő módból kiléphetünk a Tab billentyű segítségével. Ekkor nem csak a pontok (vertex), hanem az egész objektum lesz lila, jelezve, hogy ez a kijelölt elem.

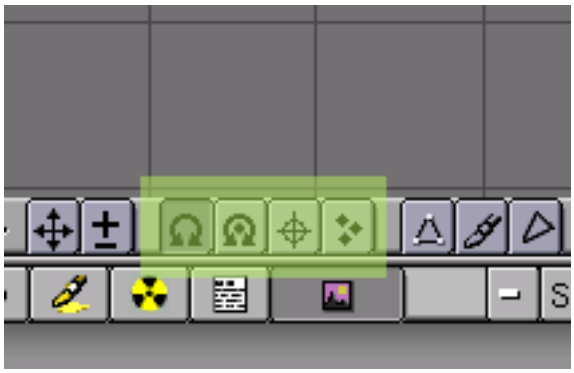
Egy valamit mindig figyelembe kell vennünk, mielőtt az objektumokon módosító műveleteket hajtanánk végre. A Blender sajátsága, hogy megkülönböztet kijelölt és aktív objektumot, megjelenítésben ezt úgy látjuk, hogy az aktív objektum színe világosabb lila, mint a többi kijelölt objektumé. A 3DS Max nem tesz ilyen különbséget. A lényeg az, hogy ha több objektum van egyszerre kijelölve és olyan műveletet akarunk végrehajtani, ami szigorúan csak egy objektumra lehet érvényes, akkor az az aktív objektumon hajtható végre. A következőkben látunk is erre példát.

Három alapvető transzformációt határozhatunk meg egy objektumra nézve: mozgatás, forgatás, átméretezés. Az ezekre vonatkozó adatokat mindig lekérhetjük [NKEY] segítségével, az adatok csak az aktív objektumra vonatkoznak. Egy ablak jelenik meg, amit a 2. ábra mutat.



2. ábra Az aktív objektum helyzetét és méretét leíró ablak

Az első három adat az objektum kiemelt pontjának térbeli helyzetét adja (LocX, Y, Z), ez általában a középpont, 3DS Maxban a pivot pont volt a kiemelt pont. A második csoport az objektum a különböző tengelyek körüli elfordulását adja (RotX, Y, Z), a harmadik hármas pedig a tengelyek szerinti átméretezést (SizeX, Y, Z). A Shift billentyűt nyomvatartva és belekattintva a különböző szövegdobozokba módosíthatjuk ezeket az adatokat mértani pontossággal. Ennél egyszerűbb, de pontatlanabb megoldás, ha az objektumon interaktívan hajtjuk végre ezeket a változtatásokat. Ha van kijelölt objektum a [GKEY] segítségével mozgathatjuk, [RKEY] -vel forgathatjuk valamint [SKEY] -vel átméretezhetjük. A forgatásnál és átméretezésnél fontos kérdés, hogy mi a viszonyítási pont, továbbá, hogy több kijelölt elem esetén egy vagy több a viszonyítási pontunk. Ezen opciók közül választhatunk a 3D-ablak fejlécén.



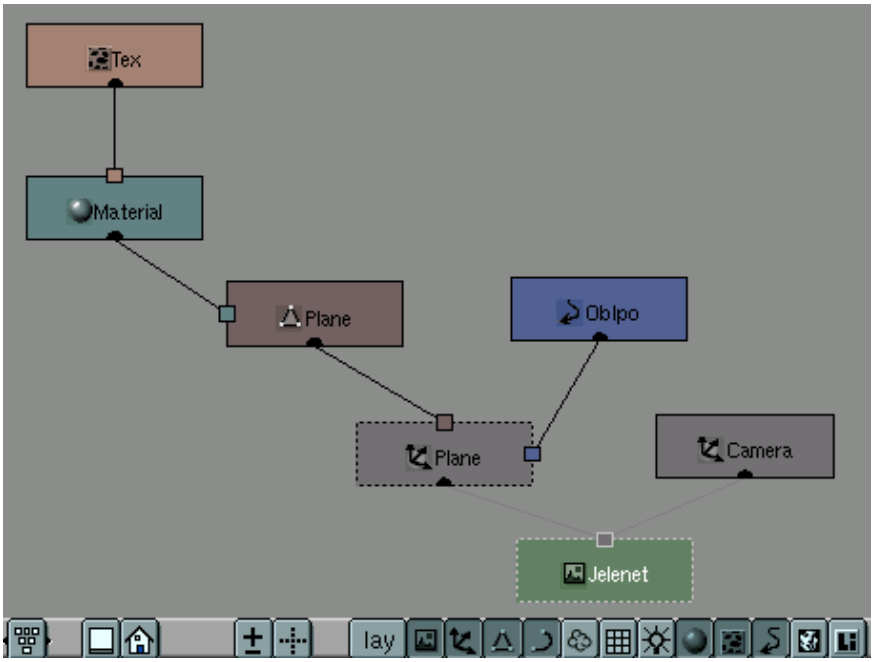
3. ábra referenciát meghatározó gombok

Az átméretező funkció kiemelt a többi közt, segítségével nemcsak a méreteit tudjuk állítani az objektumnak, illetve a méretét állítva érdekes és hasznos megoldást találhatunk. Ha 1-től nagyobb értéket állítunk be, az adott tengely szerinti mérete nagyobbodik, 0 és 1 közötti érték kicsinyít, a negatív értékek is hasonlóképpen működnek, de az adott tengely szerinti tükrözéssel. Amint később többször látjuk majd, hogy ezzel a módszerrel, -1 érték beállításával tudunk tükrözni. Ilyenkor a referenciapont általában a megfelelő helyre beállított 3D-kurzor.

Egy objektumról még többet tudhatunk meg, ha használjuk az úgynevezett Oops-ablakot (OopsWindow). Az Oops az objektum orientált programozási rendszer an-

gol megfelelőjének a rövidítése, ezzel is utalva arra, hogy a Blender objektumorientált elvre épül. Az első indításnál látható négyzet Oops-ablakbeli felépítését a 4. ábra mutatja.

A téglalapok adatblokkokat jelölnek, a köztük lévő kapcsolatokat vonalak. A hierarchia tetején a jelenet áll, melyhez csatlakozik a kamera és a négyzet objektum blokkja. Ez a blokk tartalmazza az alapvető adatokat: helyzetét, forgását, méretét. Ide csatlakozik az objektum fizikai paramétereit leíró blokk, valamint, ha van definiált mozgása a jelenetben, akkor az ezt leíró Ipo blokk. A fizikai paraméterek blokkja tartalmazza a pontokat, éleket, felületeket valamint hozzá csatlakozik a felületek megjelenítését szabályozó anyagok (materialok) blokkja. A material blokkhoz csatlakozhat egy Ipo blokk, ha a material animálva van valamint a textúrát leíró blokk. A legtöbb 3D modellező rendszerben a materialokat és a textúrákat nem választják szét, hiszen logikailag összetartoznak. A Blenderben a szétválasztás csak annyiból áll, hogy külön ablakban kezelhetjük őket, de azt párhuzamosan tesszük, a megfelelő eredmény eléréséhez.



4.ábra Az objektumok kapcsolatai az Oops-ablakban

Érdemes ezt az Oops-ablakot tanulmányozni ahhoz, hogy átlássuk, mikor melyik tulajdonság változtatásával milyen eredményt érhetünk el. Továbbá hasznos lehet az

a megoldás is, ha egy adatblokkot kereszthivatkozásokkal több más objektum is használ. Ezzel a módszerrel valósíthatjuk meg a 3DS Maxban használatos hivatkozásos klónozást (Clone - Reference).

Az objektumok alaptulajdonságait áttekintve, most nézzünk néhány modellezési technikát egy konkrét példán keresztül, mely végig kísér minket a következő fejezeteken.

3.2 Modellezési technikák

Ahhoz, hogy bemutassak a Blenderben használt modellezési technikák közül egyké egyedi megoldást, egy konkrét példát választottam témául, ami végigvezet minket ezen a fejezeten, majd a következő fejezetekben az eredménnyel dolgozunk tovább. Célkitűzésem szerint egy terem belső terét rajzolom meg. A terem kitalált elemeket tartalmaz, semmilyen építészeti stílust nem követ, mindössze didaktikai szerepe van, mely segítségével bemutathatom a módszereket, és betekintést nyújthatok a Blenderrel való munka rejtelmeibe. Többségében olyan megoldások ezek, amiknek megvan a maguk megfelelője minden más modellező programokban, de itt a munka menete vagy a módszer különbözik.

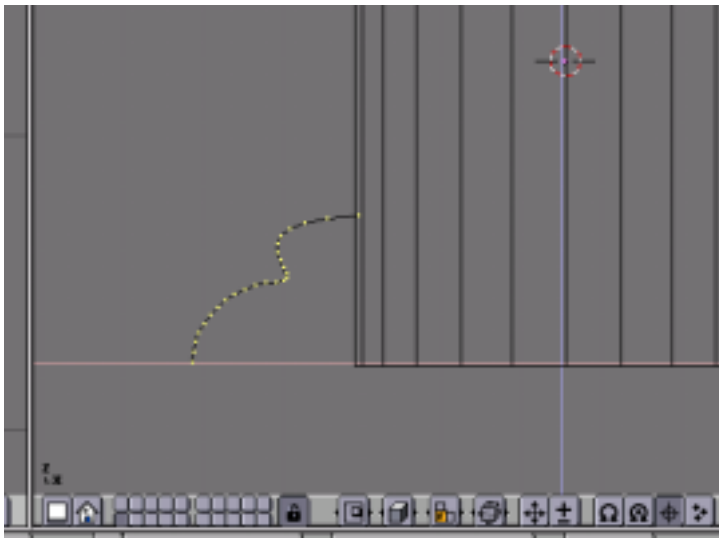
A belső tér építésének nem minden részletére térek ki, mindössze irányelveket akarok adni, hogy milyen szemléletben kell dolgozni a Blenderrel.

A következő fejezetben térek ki majd arra az eszközre, melyet bővítésként írtam. Ez a Python-script hasznos lehet a rajzolás menete alatt. Gyakran előfordul, hogy drótházós nézetben nem látjuk, valójában hogy néz ki a tárgyunk. A script segítségével állandóan ellenőrizhetjük a tárgyon végzett módosításokat. Habár ebben a fejezetben nem térek ki rá, de rajzolás közben folyamatosan használom ezt az eszközt.

Forgástestek és tömbműveletek

A Blenderben a forgástestek és a körív menti tömbműveletek szorosan összekapcsolódnak, ugyanazok a beállítások érvényesek mindkét műveletre, ezért én is egy fejezetben tárgyalom őket.

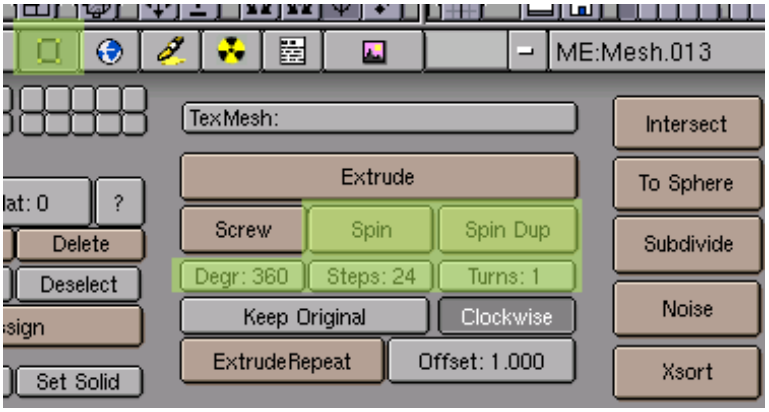
Forgástesteknek nevezzük azokat az elemeket, melyeket úgy rajzolhatunk meg, hogy egy profilt egy tengely körül elforgatjuk. Gyakori megoldás ez tárcsák vagy kelyhek rajzolásánál. Én a beltérnél használt oszlop talpának megrajzolásánál használom. Az oszlop törzse egy egyszerű henger. Miután ezt beillesztettem, az alsó részére közelítettem, hogy minél részletesebben rajzolhassam meg a talpazat profilját. Az 5. ábra mutatja az elrendezést. A profil egy bezier-görbe, melyet szintén a Space billentyű lenyomásával előugró menü segítségével illeszthetünk be.



5. ábra A talpazat profilja és a 3D-kurzor a forgatás előtt

Beillesztés után a görbe szerkesztő módban lesz, az egyes vezérlőpontokat különkülön el tudjuk érni. Miután beállítottuk a megfelelő profilt, hagyjuk el a szerkesztő módot. Forgatni csak Mesh objektumot tudunk. A kijelölt görbét [Alt-CKEY]-vel alakítsuk Mesh-sé, majd TAB billentyűvel térjünk vissza szerkesztő módba. Jelöljük ki az összes pontot. Ezt kétféle képen tehetjük: [AKEY] segítségével minden elemet kijelölhetünk, vagy mindegyiken megszüntetjük a kijelölést, vagy a [BKEY] egy keretet aktivál, melybe foglalt elemeket kijelölhetjük.

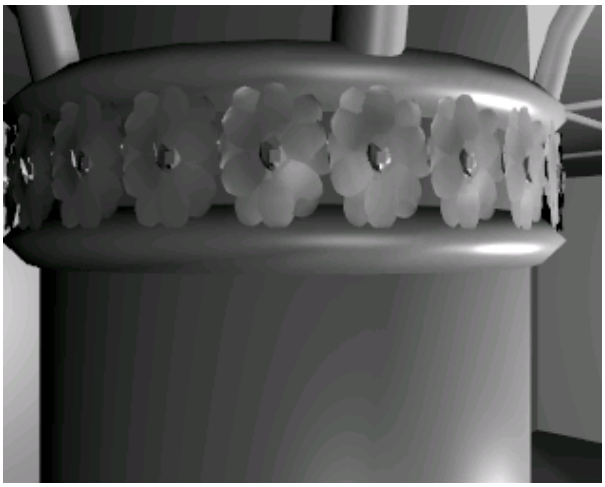
Következő lépés, hogy a Gomb-ablakban kapcsoljunk a szerkesztő gombok paneljére. A 6. ábra mutatja a használt beállításokat.



6. ábra A forgatás beállításai

A Degrees gombnál a teljes fordulás szögét adjuk meg, a lépések száma (Steps) esetemben 24. A henger szegmenseinek számát is 24-re állítottam. Mielőtt megnyomnánk a Spin (forgatás) gombot, ellenőrizzük, hogy a 3D-kurzor a henger tengelyén van. Ha a henger beillesztése óta nem mozgattuk a 3D-kurzort, akkor már csak arról kell gondoskodnunk, hogy legyen egy 3D-ablak, ami a hengert felülnéztből mutatja. Ha több 3D-ablak van, akkor a Spin gomb megnyomása után az egér egy kérdőjellé változik, kérdezve, hogy melyik ablakban forgasson a 3D-kurzor szerint. Ha csak egy 3D-ablak van, a gomb lenyomása után a forgás megvalósul. Még mindig szerkesztő módban vagyunk, amiből kiléphetünk a TAB gomb lenyomásával. Célszerű a hengert és a forgástestet egyesíteni. Jelöljük ki a két objektumot, majd [Ctrl-JKEY] egyesítsük a két Mesh objektumot eggyé. A kijelölt objektumok csatlakoznak az aktív objektumhoz, az egyesített objektum az aktív objektum tulajdonságaival (név, material, stb.) rendelkezik.

A forgástengely körüli tömbművelet hasonló a forgatáshoz. A különbség csak annyi, hogy nem szerkesztő módban forgatunk és a Spin Dup gombra kattintunk. A forgatás tárgya ez esetben egy egyszerű virág, melyből egy sormintát képzünk az oszlop felső részén.



7. ábra A virág sorminta

Render

A 7. ábrán egy úgynevezett renderelt képet látunk. Renderelésnek hívjuk azt a folyamatot, amikor a megrajzolt jelenetet egy megfigyelőpontból, a kamera szemszögéből egy raszteres képpé képezzük. A képen látható lesz minden olyan elem, aminek van térbeli kiterjedése, a kamera szemszögébe bekerül és nincs takarásban más elemek által. Nélkülözhetetlen továbbá, hogy legyen a jelenetben fényforrás. Legegyszerűbben ez egy lámpa objektum lehet, de későbbiekben találkozunk más megoldással is, amikor bármely Mesh-objektum emittálhat fényt és megvilágíthatja környezetét.

Ha a fenti nézethez hasonlót szeretnénk renderelni, helyezzük el a kamerát a megfelelő pozícióba, adjunk a jelenethez egy lámpát (Space - Add - Lamp) és nyomjuk meg az [F12KEY]-t. A kamera alapesetben része a jelenetnek, de ha nincs, akkor a lámpához hasonlóan behelyezhetünk egyet. A renderelt kép tulajdonságait a Gombablak "Display Buttons" részénél állíthatjuk be, mely ablakot aktiválhatjuk az [F10KEY]-vel. 3DS Max-ban jártas könnyen kiigazodik a beállítható lehetőségek közt, különbség mégis akad. Az "antialias" mód aktiválása az "OSA" gomb benyomásával történik, ezzel érhetjük el, hogy a ferde vonalak is folyamatosnak tűnjenek.

8. ábra Display Buttons

Az eredmény egy külön ablakban jelenik meg, melyet bezárhatunk az Esc billentyű lenyomásával, de ez nem szükséges, mert visszaválthatunk a Blender ablakára, és dolgozhatunk tovább anélkül, hogy a Render-ablakot bántanánk. A Render-ablak egy hasznos tulajdonsága, hogy válthatunk két puffer között a [JKEY]-vel. Ez akkor lehet jó eszköz, mikor két nézet különbségét szeretnénk összehasonlítani (például egy nézet az OSA bekapcsolásával és egy nélküle). A [JKEY] billentyű akkor is átváltja a puffereket, mikor nem is a Render-ablak aktív. Az [F11KEY] segítségével megnézhetjük az utolsó renderelt képet, aktívvá tehetjük a Render-ablakot újabb renderelés nélkül.

Amennyiben nemcsak egy előnézetet szeretnénk, hanem egy képformátumba ki szeretnénk menteni az eredményt, azt az Anim gombbal aktiválhatjuk. Ekkor a többi beállítástól függően kaphatunk eredményül egy képet, képek sorozatát, valamint egy animációs fájlt is. A fájl formátuma több ismert formátum közül választható. Hiányolom innen a GIF-formátumot, de egy segédprogrammal ez is megoldható.

Egy Blender-különlegesség a panoráma-kép készítésének lehetősége. Később majd ezt használjuk ki, hogy a beltérről képet kapjunk egyetlen rendereléssel.

Miután kész az oszlop, és erről ellenőrző gyorsnézetet is tudunk készíteni, folytassuk tovább a szoba építését az ablakok "kivágásával".

Boolean-műveletek

A beltér készítését egy téglatest megrajzolásával kezdtem, ami az alapja a terem falainak. Ezekben szeretnénk ablakokat vágni. 3DS Maxban láthatóan nagy hangsúly van ezen a műveleten, melyeket a tárgyakon végzett logikai utasításoknak, Boolean-

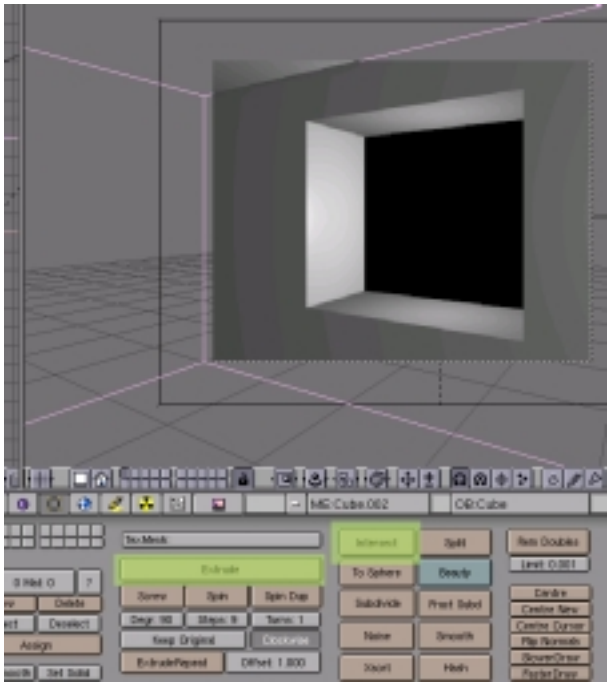
műveleteknek nevezzük. Blenderben ugyanúgy megkaphatjuk tárgyak összegét, metszetét, vagy akár egymás különbségét, csak kicsit több beavatkozással.

Első lépésként megrajzolom a kockákat, melyeket "kivonunk" a falakból. Oda helyezzük el, ahol a műveletet el szeretnénk végezni. Jelöljük ki az összes objektumot, melyek a boolean művelet résztvevői lesznek. Itt ismét figyelnünk kell arra, hogy Blenderben különbség van aktív és kijelölt objektum között. Most célszerű, hogy a kijelölt elemek közt a fal legyen világosabb lila, jelezve, hogy ez az aktív köztük. Ez a következő műveletnél fontos. Egyesítsük az elemeket egy objektummá, [Ctrl-JKEY]. Az eredmény az aktív objektum tulajdonságait örökli, tehát ha volt már a falaknak textúrájuk, akkor az változatlan marad.

Blenderben az összes boolean műveletet egy gomb aktiválja. A Gomb-ablakon váltsunk az Edit gombokra, [F9KEY]. A 3D-ablakban váltsunk át szerkesztő módba a TAB billentyűvel. Jelöljük ki az összes pontot [AKEY], majd a Gomb-ablakban nyomjuk meg az Intersect gombot. Eredményként megkapjuk az összes lehetséges boolean-művelet eredményét egyben: az összes eddig keresztező oldalak a metszéspontnál szétosztódnak. Ilyenkor már csak a fölösleges részeket kell kitörölnünk. Esetünkben az egykori kocka részeit és a falnak azt a részét, amelyet a kocka közbezár. A pontok egyenkénti kijelölése még ilyen egyszerű művelet után is hosszadalmas lenne, ezért használjuk ki itt egy, a szerkesztő módban használható kijelölési módot. Ha az egeret egy vertexpont fölé helyezzük és megnyomjuk az [LKEY]-t, akkor az összes kapcsolódó (linked) vertexpontot is kijelöljük. Így könnyedén kijelölhetjük a nem kívánt elemeket, melyeket [XKEY]-vel törölhetjük. A végeredmény a kocka alakjának megfelelő lyuk a falon.

Ha valahol tévesztenénk, olyan elemeket törölnénk esetleg, amikre szükségünk van, akkor kihasználhatjuk az egyetlen visszavonási lehetőséget, ami Blenderen belül létezik. Ha szerkesztő módban megnyomjuk az [UKEY]-t, akkor egy kérdés jelenik meg, hogy visszatöltse-e az eredeti tárgyat. Így visszatérhetünk az Intersect előtti állapotra. Ez a lehetőség elveszik, ha közben kiléptünk a szerkesztő módból (TAB),

vagy lerendereltük a jelenetet. Ezért is ajánlatos minden kritikus művelet előtt mentetni.



9. ábra Ablak a falon

A gond, amivel most szembeállunk, hogy az ablakra nézve látjuk a fal vastagságát, ami a fentiek alapján 0. Újra lépjünk szerkesztő módba, vagy ha szerkesztő módban voltunk, akkor kétszer a TAB megnyomásával biztosítsuk magunknak, hogy esetleges hibánál az Intersect utáni állapotot töltsük vissza az [UKEY]-vel.

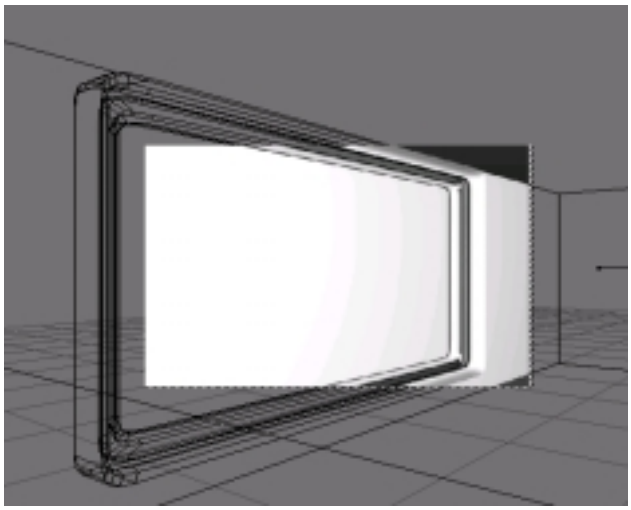
A beltér falai több helyen is módosítva voltak az Extrude (kihúzás) utasítással. Ez hasznos és gyakran használt módszer más 3D modellező programokban is. Használata egyszerű: jelöljük ki a kihúzandó pontokat, éleket vagy felületet, majd [EKEY]-t nyomjuk meg abban a 3D-ablakban, amelyikben a kihúzást le tudjuk kezelni. Így újabb felületeket adva objektumunkhoz lényegesen befolyásolhatjuk alakját.

Loft

A loft kifejezést a 3DS Max használja egy alakzat út mentén való kihúzására, miközben a keresztmetszetet hatékonyan tudjuk módosítani. Ilyen módszerrel egyszerűen lehet bonyolultabb elemeket megrajzolni. Blenderben külön loft-utasítás nincs,

csak a meglévőkből oldhatjuk meg ezt a problémát. A megoldás pedig az [FKEY] szerkesztő módban. [FKEY] segítségével több hasonló módosítást végezhetünk. Ha csak két pont van kijelölve, akkor azokat egy éllel köti össze, ha három vagy négy pont van kijelölve, akkor köztük egy felületet feszítünk ki. Esetünkben a harmadik funkciót használjuk ki, amikor felületeket kötünk össze.

A beltérben a festmény rámája készült loft-módszerrel. A rámának a profilját lerajzoltam egy felület (Surface) segítségével. A profilról másolatokat készítettem és elhelyeztem a ráma négy sarkába. Ajánlatos másolatokat készíteni, ha loft módszert akarunk használni, mert az egyik megkötés, hogy a szegmensek számának azonosnak kell lenniük az egyes keresztmetszeteknél.



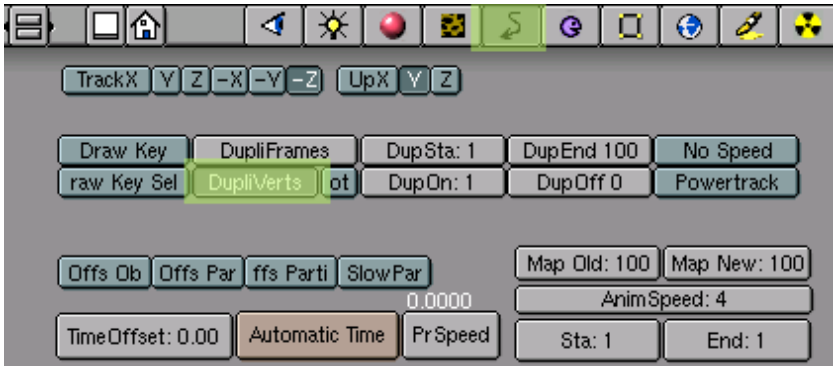
10.ábra A képkeret renderelt és drótvázas képe egyben

A keresztmetszeteket egyesítettem egy objektumba, [Ctrl-JKEY], szerkesztő módban az összes pontot kijelöltem, [AKEY], majd összekötöttem őket [FKEY]-vel. Az eredmény egy NURBS felület, mely arról ismert, hogy jól módosítható a kontrolpontok segítségével. Az alakzat viszont nem zárt. Ezt elérhetjük a [CKEY] segítségével. Az én esetemben szükség volt megduplázni a kontrolpontok számát, hogy valóban festménykeret legyen a végeredmény. Újra kijelölöm a pontokat, majd mindegyik összekötést felosztom még egy ponttal. A Subdivide aktiválható a Gombablak megfelelő gombjával, vagy 3D-ablakban a [WKEY] lenyomásával. [WKEY]-t

máskor is használhatjuk szerkesztő módban, ilyenkor ugyanis egy előugró menüben több hasznos utasítást érhetünk el.
A keletkezett kontrolpontok segítségével készült keret drótvázas és renderelt képét a 10. ábra mutatja.

Egyéb tömbműveletek

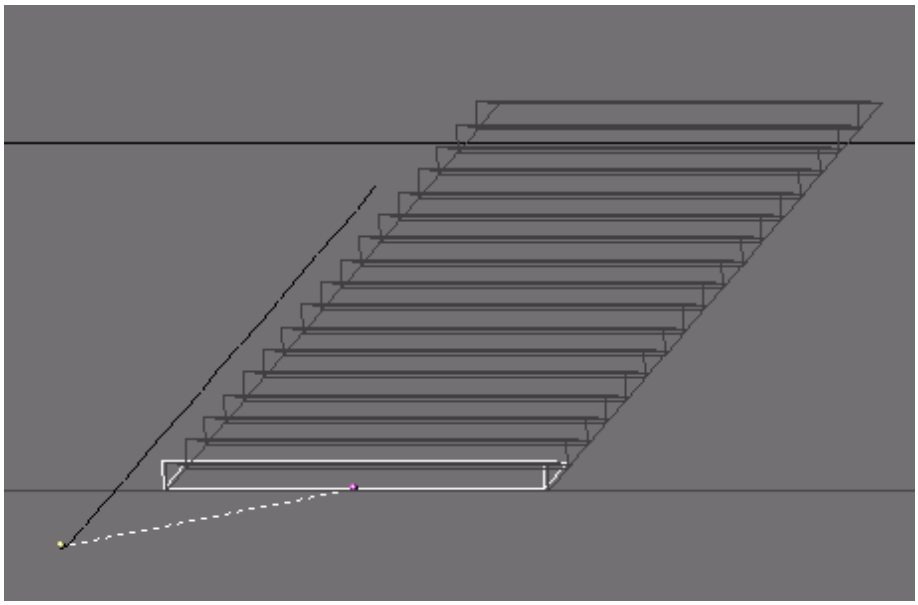
Az előző részekben már láthattuk, hogy tudunk egy tárgyat megsokszorozni egy forgáspont körül. A következő megoldás általánosabb eszközt ad a kezünkbe, ezért ezzel akár a fenti többszörözést is végrehajthatjuk.
Ezt a módszert leginkább a 3DS Maxban használatos Snapshot funkcióhoz lehet hasonlítani. Ott kellett, hogy a másolandó tárgynak legyen animált mozgása. Blenderben is a Gomb-ablak Animation Buttons részét fogjuk használni.
Miután kész a tárgyunk, amit sokszorozni akarunk, adjunk a jelenetünkhöz egy alakzatot, vagy görbét aszerint, hogy milyen elrendezésben szeretnénk a tárgyakat elhelyezni. Később látjuk majd, hogy ennek az alakzatnak a pontjain helyezkednek el majd a másolatok. Ehhez először is szülő-gyerek kapcsolatot kell létrehoznunk az alakzat és a másolandó tárgy között, amit [Ctrl-PKEY]-vel tudunk elérni. Ilyenkor a kijelölt elemek szülőeleme lesz az aktív elem, ezért ügyeljünk arra, hogy először a másolandó elemet jelöljük ki, majd az "utat" megadó alakzatot. Az aktív elem világosabb lila, mint a kijelölt elem. Miután létrehoztuk a gyerek-szülő kapcsolatot, az elemek pivotpontjait egy szaggatott vonal köti össze. Ezután ezekkel az elemekkel a 3DS Maxban megszokott módon dolgozhatunk: minden szülőn végzett módosítás hat a gyerekre, de a gyereken végzett módosítások nem érintik a szülőobjektumot.
A kapcsolat kialakítása után jelöljük ki csak a szülőt, a Gomb-ablakban pedig kapcsoljunk az Animation gombokhoz, [F7KEY]. Itt már semmi más dolgunk, minthogy bekapcsoljuk a DupliVerts gombot.



11. ábra Animation buttons

A gomb benyomása után a gyerekobjektum másolatai megjelennek a szülő minden egyes vertexpontjában. A gyerekobjektum a 3D-abalkban még mindig látszik, de a renderelt képen nem jelenik meg.

Ennek a módszernek nagy előnye, hogy utólag is módosíthatunk minden paramétert. A gyerekobjektumokon végzett műveletek láthatók a másolatokon is, továbbá, ha a szülő objektum vertexpontjait áthelyezzük, töröljük, vagy esetleg hozzáadunk, az mind hatással van a másolatok helyzetére és számára.



12.ábra A vonal és a lépcsőfok kapcsolatából kirajzolódott lépcsősor

A belső tér lépcsői készültek ezzel a módszerrel. A képen látható egy lépcsőfok, mellette az útnak használt egyenes, melynek a vonal mentén 17 vertexpontja van. Ezzel a végeredményben 17 fokú lépcsősort kapunk.

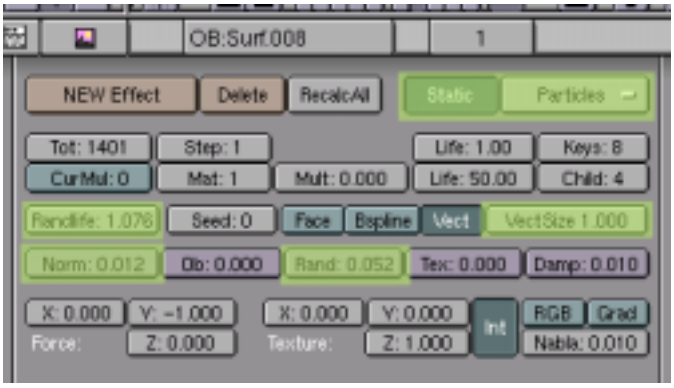
Részecskerendszerek

A részecskeobjektumok egy adott helyről kiinduló, azonos módon, bizonyos törvényszerűségek szerint viselkedő azonos alakú tárgyak. Nem kell a tárgyak mozgását külön-külön meghatározunk, elég csak a törvényszerűségeket változtatni. Ilyen részecskeobjektumok lehetnek esőcseppek, füst, de akár statikus részecskeobjektumok is lehetnek, mint a jelenetben használt rojtok a függönyön.

Akár csak 3DS Max-ban, Blenderben is egy emitter objektumra van szükség, ami a részecskerendszer alapja lesz. A felső függönyből készítettem egy másolatot, és úgy módosítottam, hogy csak az alsó kontúrvonal maradt meg, ahonnan majd a rojtok lógnak.

A részecskerendszer beállítási lehetőségeit a Gomb-ablak Animation Buttons részénél ([F7KEY]) állíthatjuk. Ezzel a gombcsoporttal már találkoztunk az előző részben, most viszont az ablak bal oldali részénél adunk a tárgyhoz új módosítót (NEW effect). A megjelenő legördülő menüből válasszuk ki a "Particles" opciót. Ekkor a sok beállítási lehetőség a részecskék élettartamára, irányára, méretére és még más viselkedési és megjelenítési tulajdonságaira vonatkoznak.

Az emitter vertexpontjaiból véletlenszerűen, a gravitációt valamennyire követve fehér csíkok jönnek ki, képviselve a rojtokat. A beállításokat az ábra mutatja.



13. ábra A jelenetben használt részecske rendszerek beállításai

Ez a részecskerendszerek nagy előnye. Olyan objektumokat rajzolhatunk, melyeket más technikával nem tudnánk, és ezt egyszerűen, néhány paraméter beállításával tehetjük.

3.3 Rendering - képleképzés

Az előző fejezetben már volt szó a gyorsnézet készítéséről. Most, hogy már kész a beltér, elhelyezhetjük a fényforrásainkat, valamint a leglátványosabb munka is hátra van: a textúrák társítása az objektumokhoz. Mielőtt ezekre kitérnénk, a Blendernek egy olyan lehetőségét ismerjük meg, ami a 3DS Max 2.5-ös verziójában nem volt meg. Ez pedig egy teljesen más, a valóságot jobban közelítő fényterjedés-számítás, a radiosity.

Radiosity

Ennek a módszernek az az alapelve, hogy a fény minden tárgyról visszaverődik valamilyen mértékben, és az megvilágítja környezetét. Egy ilyen számítási módszer hosszadalmas lehet, hisz addig tart, amíg van olyan fénynyaláb, ami kilép egy felületről.

Egy ilyen számításnál a Blender textúra beállításai közül a szín (R, G, B), az emisszió (Emit) és a diffúzió (Ref) paraméterek a meghatározóak. A számítást nem befolyásolják a lámpák. Csak olyan fényforrásaink lesznek, melyek olyan geometriák, amiknek az emissziója nagyobb, mint 0. A radiosity-számítás a jelenetben résztvevő objektumokat felosztja kis elemekre (patch), melyek a fény fogadására és továbbadására képesek. Az ezeket az elemeket leíró vertexpontok játszanak majd nagy szerepet a későbbiekben, ugyanis minden vertexpontnak beállítjuk a szín-paraméterét. Mivel a nagy felületeket ez a módszer felosztja sok kis részre, valamint minden vertexpontnak külön tároljuk a színét is, ezért a fájlméretünk nagyon megnőhet. Az én esetemben ez a növekedés négyszeres, de a végeredmény olyan valósághű fény-árnyékhatás, amit más általam ismert modellezőprogrammal nem tudnék előállítani.

Első lépésként minden ablakba helyeztem egy négyzetet, melyeknek az emisszióját nullánál nagyobb értékre állítottam. A fény a felület közepéről indul kifele, ezért az ablakba helyezett négyzetek szegmenseit megdupláztam, ezzel is segítve, hogy a valóságot jobban követve, szórt fény jöjjön a terembe.

Ha esetleg van olyan elem a jelenetben, ami nem Mesh objektum, akkor azt [Alt-CKEY]-vel Mesh-sé kell konvertálnunk. Esetemben a függönyt és a képkeretet kell NURBS-ből Mesh-sé alakítani. Jelöljük ki a jelenet összes elemét, [AKEY]. Ha esetleg nem Mesh-objektumokat is kijelölünk, azok nem vesznek majd részt a számításban, ilyenek a kamera, valamint az előnézeteknél használt fényforrások.

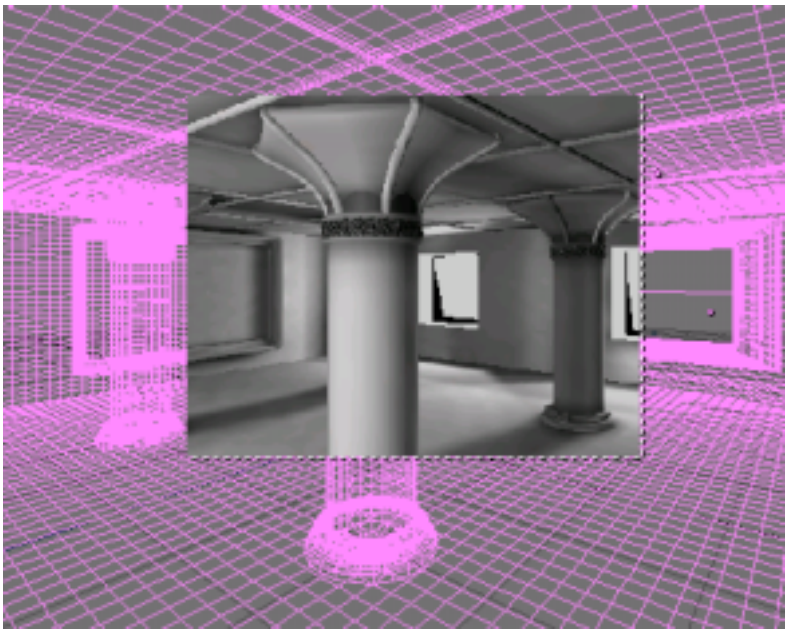
A Gomb-ablakban kapcsoljunk a Radiosity-gombcsoporthoz. Látjuk, hogy az egyetlen használható gomb a "Collect Meshes", ezzel a gombbal adhatjuk hozzá az objektumokat a radiosity-motorhoz. Megnyomva ezt a gombot, több más gomb válik aktívvá. Ezekkel a gombokkal állíthatunk a számítás pontosságán. Az ábra mutatja, a beltér számításánál milyen beállításokat használtam.

14. ábra Radiosity-beállítások

A "Go" gomb indítja a számítást, aminek a vége csak akkor lesz, ha már minden kilépő-visszaverődő fény intenzitása kisebb, mint a "Convergence" gombnál beállított érték. A fenti beállítások mellett egy Pentium 1, 240MHz, 96MB Ram konfigurációjú gép két órán át számolt, ez körülbelül 700 számítási lépést jelent. A lépések számát az egér mutatja, ami ilyenkor átváltozik egy egyszerű számlálóvá. Ha a számítás közben már elégedettek vagyunk az eredménnyel, akkor megszakíthatjuk a számítást az Esc-billentyű lenyomásával. Az addig kiszámított értékek megmaradnak.

A számítás végeztével utólag még korrigálhatunk, például a gamma korrekcióval, a monitor paraméteréhez igazítva. Amíg a radiosity-motor aktív, addig minden szerkesztő funkció tiltva van. Ha nem lennénk elégedettek az eredménnyel, akkor a "Free Radio Data"-gombbal visszatérhetünk a számítás előtti állapothoz. Ha a kiszámított eredmény tetszik, akkor a következő két gomb egyikét választjuk: "Replace Meshes" az eredeti objektumokat kicseréli az új objektummal, aminek a

vertexpontjai már tartalmazzák a színértékeket, "Add new Meshes" megtartja az eredeti objektumokat is. Miután hozzáadtuk a kiszámított objektumot a jelenetünkhöz a "Free Radio Data" gombbal kiléphetünk a radiosity-motorból.



15. ábra Radiosity-eredmény

Az eredmény egy sok vertexpontból álló objektum, ami tartalmazza a jelenet összes elemét, ami részt vett a radiosity-számításnál. Mostantól a lámpákra nincs szükségünk, [F12KEY]-vel bárhol képet kaphatunk a kamera szemszögéből. Most már egy-egy renderelés gyorsabb, mint mikor lámpák voltak a jeleneteinkben. A radiosity hosszú folyamat, de csak egyszer kell elvégezni. Amit ilyenkor látunk, az a jelenetünk elemei textúra nélkül. A következő részben adunk textúrákat a jelenethez, de előbb a nagy objektumot darabokra kell szednünk, hogy kezelhetőbb legyen a későbbiekben.

Jelöljük ki az objektumot, és lépjünk be szerkesztőmódba a TAB billentyűvel. Első ránézésre soknak tűnik a sok vertexpont, de később látjuk majd, hogy ezek könnyen kezelhetők. A fentebb említett egyesítés, [Ctrl-JKEY] ellentétje a szétválasztás lesz a kulcsmozdulat, de ennél fontosabb a kijelölés módja. A boolean-műveleteknél használt [LKEY] most még nagyobb szolgálatot tesz, hiszen egy mozdulattal sok összetartozó elemet tudunk kijelölni. Ha a kijelöléssel elégedettek vagyunk, akkor

[PKEY]-vel leválaszthatjuk a kijelölt elemet az objektumról. Ez a radiosity eredményéből kapott objektum bonyolultságától függően sokáig is eltarthat, de ilyenkor jövünk rá a Blender felhasználói felületének praktikus kialakítására. Hiszen a billentyűzeten csak két billentyűt használunk, az egéren a bal gombot a szétválasztás jóváhagyására, a középső gombot pedig a nézet forgatására. Ily módon hatékonyan és gyorsan elvégezhetjük a szétválasztásokat.

Ideje megemlíteni egy következő szembetűnő különbséget a 3DS Max-szal szemben. A Blenderben használhatunk rétegeket. A gyorsbillentyűk a rétegek aktiválására a számbillentyűk. A rétegeket az elemek logikai csoportosítására használhatjuk, így rejtve tarthatjuk azokat az elemeket, amelyekkel pillanatnyilag nem dolgozunk, és csak akkor tesszük láthatóvá, mikor a képet rendereljük. A sok vertexpontból álló objektumok nagyban lassíthatják a munkát, ezért érdemes a radiosity eredményének részeit külön rétegekre osztani. A mozgatást többféle módon is megtehetjük. A legpraktikusabb az [MKEY], amivel az összes kijelölt objektumot mozgathatjuk a megadott rétegre. A gomb lenyomása után egy előugró ábra mutatja a rétegek szimbólumát, amelyet a 3D-ablak fejlécén is láthatunk. A rétegeket kis nyomógombok szemléltetik, melyek közül választva mozgatjuk a kijelölt elemeket az adott rétegre.

Az objektumok rendezése után hozzáadhatjuk a textúrákat a jelenethez.

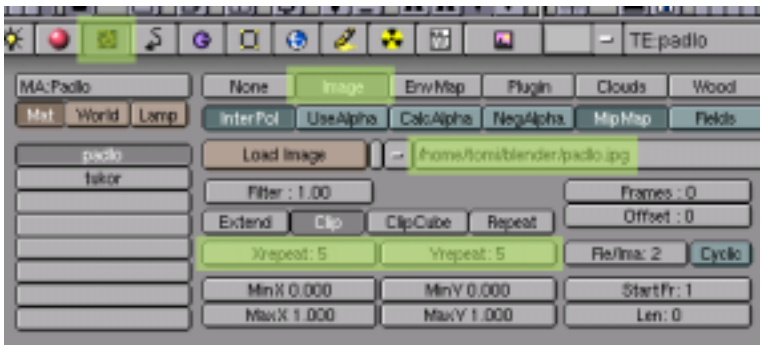
Textúrák

A jelenetünkben szereplő tárgyaknak ajánlatos valamilyen anyagot adni, hogy megjelenítésük élethűbb legyen. Ezeket az anyagokat, akárcsak 3DS Max-ban materialoknak hívjuk és logikájuk is hasonló. Vannak általános paraméterek a megjelenítésre, mint a szín, csillogás, diffúzió, emisszió, átlátszóság és még mások, melyeket mind befolyásolhatnak a materialokhoz adott textúrák. Egy tárgyhoz legfeljebb 8 texúrát adhatunk, melyeket kombinálhatunk egymással és a material alapbeállításaival. Példánkban a padló anyagának beállítását mutatom be, mely a legösszetettebb a jelenetben.

Elképzelésem egy márványkockákból összeállított padló, melynek van mintája, tükröződik, és meg kell tartania azokat az árnyakat, melyeket a radiosity számítás alatt kapott. Jelöljük ki a padlót, amit elkülönítettünk a radiosity számítás eredményéből és nyomjuk meg [F5KEY]-t, hogy a Gomb-ablakon a materialokhoz tartozó gombcsoportot lássuk. A radiosity eredményéből már valószínű, hogy egy material már van hozzáadva a tárgyhoz. Ha [F5KEY]-re egy üres ablak jelenik meg, akkor adjunk hozzá egy új materialt a fejlécen legördíthető menüből. Esetemben viszont már van materialja a padlónak, sőt a material neve kék színnel ki van emelve, ami azt jelenti, hogy a materialt több objektum is használja. A használók számát a material neve mellett álló szám mutatja. Erre a számra kattintva tehetjük egyedivé a materialt, így minden változatás már csak a padlóra fog vonatkozni.

16.ábra Egyszerre 4 használója van a materialnak

A material beállításainál a jobb oldali gombok az általános beállításokra vonatkoznak, a baloldaliak az egyes textúrák anyagra való hatását állítják be. Hogy mikor melyik textúrára vonatkoznak, azt a felső választógomb-sor adja meg. Ez a sor még üres, adjunk textúrát az anyagunkhoz. [F6KEY] segítségével átkapcsolhatunk a textúra beállításaihoz. A fejlécen a legördülő menüből adjunk új textúrát az anyaghoz. Érdemes már most nevet adni a textúráknak, hogy később jobban kiigazodhassunk rajtuk.

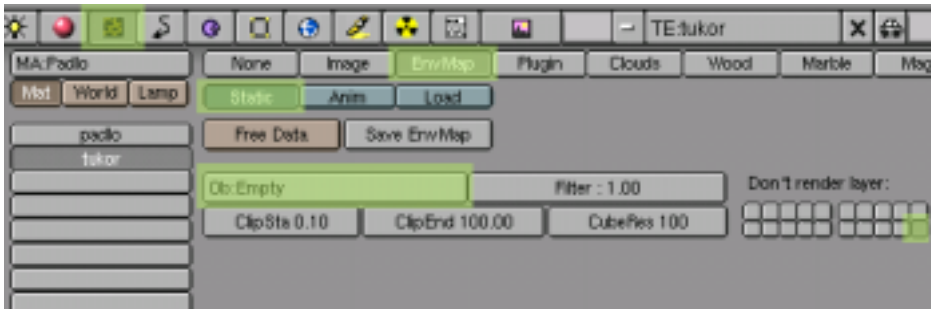


17.ábra Textúra ablak, a képet X és Y irányban is ismételjük 5-ször

A materialhoz kapcsolódó textúrák közti prioritást a material-ablakban látható textúrák listája dönti el, ezért először a mintát kell létrehoznunk, ami egy raszteres kép lesz. A márványminta Gimp-ben készült. A kép hozzáadásakor állíthatunk több paramétert is, melyek hasonlóak a 3DS Max beállításihoz is. Mi azt állítjuk be, hogy hányszor ismételjen X és Y irányban, ezzel kockázzuk fel a padlót.

Váltsunk vissza a material-ablakba, [F5KEY]. Az előnézet-ablakban már látjuk a textúra hatását. Ebben az ablakban beállíthatjuk a tárgyra feszítés koordinátáit, amit ellenőrizhetünk egy gyors renderrel, [F12KEY]. Ahhoz, hogy a radiosity-számításából származó árnyalások ne vesszenek el, a "VColLight" gombot benyomva kell tartanunk.

Ezek a lépések megegyeznek a többi tárgynál is. Most pedig a tükröződést állítjuk be a padlónak egy újabb textúra hozzáadásával, ez pedig egy "Enviroment" típusú textúra lesz. A tükröződő felületek képzése nagyban különbözik a 3DS Max-hoz képest. A Blenderben egy objektumot kell elhelyezni abba a pontba, ami a kamera helyzetének a tükrözése a padló síkja szerint. Ilyen esetekben használjuk a Blenderben található üres objektumot, az Empty-t, melynek nincs kiterjedése és anyaga, csak három tengelye. A kamerából a padlóra tekintve olyan mintája lesz a padlónak, mintha az Empty-ből néznénk fel, ezért fontos, hogy az Empty helyzete a padló szerint tükrözve legyen a kamerához képest. Az Enviroment beállításai a következő ábrán láthatók.



18.ábra A tükröződés beállításai a textúra-ablakban

Szükséges, hogy a padló egy különálló rétegen legyen, és ezt a réteget kell beállítani az Enviroment beállításoknál, hogy ezt ne renderelje, ezzel engedve, hogy az Empty átlásson a padlón. A material ablakban két beállítást kell még elvégeznünk. Az egyik a feszítés számításánál a "Ref" opciót, a megjelenítésnél pedig "CMir"-t kell választani. Ügyelnünk kell a material beállításainál, hogy az Enviroment-textúra felülírja a radiosity számításait, ezért a textúra hozzáadásának módját kell befolyásolnunk, hogy az előző textúráknak is legyen hatásuk.

19.ábra A tükrözés beállításai a material-ablakban

Ha van tükröződő tárgy a jelenetünkben, akkor renderelésnél először a tükröződő felületek textúráját számolja ki a Blender 6 meghatározó irányban. Ez a renderelési időt lassítja, de ha csak állóképet renderelünk, akkor a textúra beállításainál a "Static" opciót választva megjegyzi az előző renderelésből származó eredményt. Az "Anim" opció akkor szükséges, mikor a jelenetben végzett változtatásokat mindig látni szeretnénk a tükörképben is.

Láthatjuk, hogy két textúra kombinációjával is összetett megjelenítést érhetünk el, ezért nem tűnik kevésnek, hogy legfeljebb 8 textúrát használhatunk egy-egy materialnál.

Fények

Eddig a jelenetünkben nincsenek lámpaobjektumok, melyek megvilágítanák a tárgyainkat. A renderelt képeken az árnyékokat és a fényeket a radiosity-számításból adódó színértékek adták, amik az egyes vertexpontokhoz vannak rendelve. Ezek valósághűbb ábrázolást adnak, mint a sugárkövetéses módszerek, mikor egy lámpából jövő fénysugár útját követjük a megfigyelő szemszögéből. Jelenetünkbe mégis elhelyezünk lámpákat, hogy az ablakon beszűrődő fényt szemléltessük, mintha a közvetlen fény megvilágítaná a levegőben levő porszemcséket.

A 3DS Max-ban volume light-nak hívott jelenséget a környezeti paramétereknél állítottuk be, az egyes fényforrásokhoz rendelve. Blenderben ezt közvetlenül a fények beállítására szolgáló gombcsoportnál, a Gomb-ablakban állíthatjuk be, [F4KEY]. Az ablakban akkor jelennek meg gombok, ha ki van jelölve egy lámpa.

Blenderben 4 féle fényforrást különböztetünk meg. Ezek közül csak a pontvilágító (Spot light) tud árnyékot vetni. Ezt használjuk majd a volumetrikus fénynél is.

A jelenetben használt két lámpa beállítási paramétereit a 20. ábra mutatja.

20. ábra „Halo” féle lámpák a jelenetben

Amire feltétlenül figyelnünk kell a volumetrikus fény készítésénél, az a lámpa típusa (Spot), paramétere közt a Halo gomb aktiválása, az intenzitás (HaloInt) és az árnyékszámítás (Halo step) értéke nagyobb legyen nullánál, valamint a jelenetünk megvilágított része a fényforrás kezdeti és végpontjai közt legyenek.

Panorámakép

Ebben a fejezetben láthattuk, mennyi mindent kell beállítani ahhoz, hogy valamennyire valósághű végeredményt kapjunk. A két legfontosabb dolgon (textúrák és fények) kívül, több más lehetőség is van, amikre ezen diplomamunka nem tér ki. Ilyen például a környező világ beállításának lehetősége, ahol csillagokat, felhőket esetleg ködöt rajzolhatunk jelenetünkbe.

A beltér kész a végeredmény leképzésére. Renderelhetünk állóképet, vagy a kamera mozgásával az építészetben népszerű látványtechnikai megoldást, bejárást szimulálva animációt is készíthetünk. Egy köztes megoldásra is ad lehetőséget a Blender, a panorámaképre. Ennek végeredménye valójában csak egy állókép, de egy nézőpontból nézve 360 fokos szögben láthatjuk a jelenetet. Egy ilyen hosszúkás kép kiterítve nem mutat jól, viszont diplomamunkám része egy egyszerű, Pythonban írt programkód, ami kezeli ezt a képet. Ezt a programkódot a harmadik fejezetben mutatom be. Először lássuk milyen beállítások szükségesek egy panorámaképhez.

21. ábra A panoráma képhez használt értékek

A Gomb-ablak megjelenítésért felelős gombcsoportját az [F10KEY]-el tudjuk aktiválni. A jobb oldali választógombok közül választva előre definiált beállításokat kaphatunk, melyek a renderelés paramétereit állítják be. Többek közt itt is megtalálhatjuk a panorámakép beállítási lehetőségét. Ha benyomjuk ezt a gombot, akkor csak egy 180 fokos látószöget kapunk, viszont láthatjuk, mennyi paraméter változik meg a Gomb-ablakban. Például a renderelt kép mérete és a vízszintes irányú ismétlés. Ezeket az értékeket állítva kaphatunk egy 360 fokos képet is. Az én beállításaimat a következő ábra mutatja. Az "ANIM" gombbal tudunk egy képet renderelni. A

renderelés szokatlan módon először kis szeletekben látjuk, majd ezek összegét a panoráma képet.

3.4 A Blender nem csak 3D modellező

Eddig a Blenderre úgy tekintettünk, mint egy modellező programra, de a Blender több ennél. Már a 2.03 –as verzióban is található olyan lehetőség, ami túlnyúl a 3D modellező programok kötelességein, ma pedig, amikor már a 2.23 –as verzió is letölthető, a szolgáltatások egész sora vár a felhasználókra. Ezek közül már nem minden ingyenes, de később látjuk majd ennek okát.

Ami már több mint egy éve gyarapítja a Blender felhasználók körét, az az úgynevezett Játék Motor (Game Engine). Ez a Blender egy beépített része. Segítségével a megrajzolt tárgyakkal készíthetünk saját játékot, definiálva a vezérlő műveleteket és a tárgyak viselkedését. A Blender honlapján látható, hogy ez egy népszerű része a programnak, több fejlesztés és javítás is vonatkozik erre részre. Természetes volt tehát az igény, hogy olyan játékprogramokat is lehessen írni, amelyek futtatásához nincs szükség Blenderre a felhasználói oldalon. Ez viszont már egy olyan eszköz lenne a rajzolók kezében, amivel már kereskedelmi forgalomban is terjeszthető programokat írhatnak. A Blendert fejlesztő cég ezért három különböző terméket kezdet el forgalmazni a 2001 őszétől. Blender Creator lett a neve a rajzoló programnak, amiről e diplomamunka szól. Blender Publisher az a rész, ami segítségével a játékprogramok önállóan futtatható állományokká fordíthatók. Ez a rész pénzbe kerül. A honlapon találhatók bemutató programok, melyek ilyen módon lettek fordítva. Eddig csak egy Windows alá írt programot próbáltam ki, de sajnos túlságosan erőforrás igényes, és nem találtam gépet, mellyel élvezhető sebességen lehetett volna játszani. A nem rég megjelent újabb verzió teszi lehetővé, hogy Linux operációs rendszer alá is lehessen fordítani játékokat, de ilyen bemutató programot még nem állt módomban letölteni.

A harmadik rész, ami a fejlesztők bevallása szerint is gyerekcipőben jár, a Blender Plug-In. Ez a web-böngészők kiegészítője, melynek segítségével háromdimenziós

honlapokat fejleszthetünk és tekinthetünk meg. Az ilyen honlapok fejlesztésénél ajánlatos az előző fejezetekben tárgyalt radiosity eljárással dolgozni, mely gyors és látványos megoldást ad például egy beltér interaktív bejárásához. A Blender Plug-In egyedi újítás a 3D modellezés világában, és a Blendert ismerők csak remélni tudják, hogy egyszer a Plug-In a böngészők része lesz, és a honlapok látogatói nem lesznek kötelezve kényelmetlen letöltésre.

Mindezek mellett a Blender legnépszerűbb része a 3D modellező, melyet az utóbbi verzióktól Blender Creatornak hívnak.

4 Scriptek a Blenderben

A modellező programoknál megszokhattuk, hogy van egy script-nyelvük, mellyel bizonyos műveletsorokat automatizálhatunk. Ez a megoldás hasznos eszközt ad a kezünkbe AoutCAD-nél és 3DS Max-nál is. A Blendernél is létezik ez a lehetőség azzal a különbséggel, hogy egy létező programnyelv, a Python a script-nyelve. Ezeket a programkódokat a fájl tárolja magában, nincs szükség arra, hogy minden használat előtt újra betöltsük. A Blender része egy egyszerű szövegszerkesztő, melyet bármelyik ablakban aktiválhatunk a [Shift-F11KEY] kombinációval. A legördülő menü segítségével választhatunk a betöltött scriptek közül, vagy újat tölthetünk be, vagy egy üres lapon kezdhetünk egy újat.

Két fajta scriptet különböztessünk meg az elején. Az egyik, amit hozzárendelhetünk objektumokhoz, és meghatározott eseményeknél befolyásolja az objektumot. Ilyenre például akkor lehet szükségünk, ha az objektumot korlátozni akarjuk valamelyik szabadsági fokában, hogy csak egy síkban tudjuk mozgatni. A másik fajta scriptet először futtatni kell, hogy kifejthesse hatását. Az ilyen scriptekkel automatizálhatunk feladatsorokat, de más hasznos megoldásokat is elérhetünk.

Diplomamunkám része egy olyan script, ami a modellezést segíti abban, hogy egy külön ablakban mutatja az aktív objektumot, ahol elforgathatjuk, hogy jobban láthassuk a módosításainkat. A script [Alt-PKEY]-vel futtatható, miközben az egér a szövegszerkesztő ablak felett van. Ekkor az ablakban megjelenik az aktív objektum, ha van éle és oldala, vagyis, ha Mesh-objektum. A vezérlőfunkciókat a script elején levő megjegyzés aprólékosan leírja. Tudjuk a tárgyat forgatni, közelíthetünk-távolodhatunk, választhatunk drótvázas megjelenítést is.

A mellékletben megtekinthető a teljes programkód, ebben a fejezetben néhány jellemző részt ragadok csak ki, melyeket részletesebben kommentálok.

A program négy fő részre osztható: egy osztály, két függvény és a főprogramot képviselő függvényhívás. Ez a függvényhívás a Blender sajátsága, nem része a Python utasításkészletének.

Register(draw, event)

Ez az utasítás hivatott kezelni az eseményeket az ablakban. Az első paraméter egy függvényhívás, amit akkor hív meg a program, mikor az ablakot kell frissíteni, újrarajzolni. A második paraméterben szerepel annak a függvénynek a neve, amelyek lekezelik a felhasználó beavatkozásait. Ez a függvény csak feltételek sorozata, mely vizsgálja, melyik billentyű lett lenyomva. A billentyűkódokat képviselő konstansok szintaktikáját követtük a diplomamunka eddigi részeiben.

```
def event(evt, val):
      if (evt==QKEY): Exit()
      if (evt==ESCKEY): Exit()
      if (evt==LEFTARROWKEY): valami.oldal = valami.oldal - 1
      if (evt==RIGHTARROWKEY): valami.oldal = valami.oldal + 1
      if (evt==UPARROWKEY): valami.fel = valami.fel - 1
      if (evt==DOWNARROWKEY): valami.fel = valami.fel + 1
      if (evt==WKEY):
            valami.mode4 = GL_LINE_LOOP
            valami.mode3 = GL_LINE_LOOP
            valami.color = 0
            valami.volt  = 0
      if (evt==SKEY):
            valami.mode4 = GL_QUADS
            valami.mode3 = GL_TRIANGLES
            valami.color = 1
            valami.volt  = 0
      if (evt==RKEY): valami.volt = 0
      if (evt==XKEY): valami.zoom = valami.zoom + 0.02
      if (evt==CKEY): valami.zoom = valami.zoom - 0.02
```

Az objektum megjelenítésére az OpenGL utasításokat használtam, melyeket akkor használhatunk, ha importáljuk a Blender OpenGL utasításkészletét.

Import BGL

A programkódban túlnyomó többségben OpenGL utasításokat használtam, melyekre mind szükség van az objektum megjelenítésénél, ilyenek a környezet és a nézőpont beállításai. Ezek a frissítésért felelős függvényben találhatók meg.

```
      glEnable(GL_DEPTH_TEST)
      glMatrixMode(GL_PROJECTION)
      glLoadIdentity()
      glOrtho(-5.0*zoom, 5.0*zoom, -5.0*zoom, 5.0*zoom, 0.1, 10.0)
      glMatrixMode(GL_MODELVIEW)
      glLoadIdentity()
      glTranslatef(0.0, 0.0, -5.0)
      glClearColor(0.3, 0.3, 0.3, 0.0)
      glClear(GL_COLOR_BUFFER_BIT)
      glClear(GL_DEPTH_BUFFER_BIT)
```

A script csak olyan elemeket tud megjeleníteni, melyeknek van felületük. Ez igaz a jelenetben használt objektumok többségére. Nem tudja megrajzolni az olyan objektumokat, melyeknek nincs kiterjedésük (kamerák, lámpák), de a problémát a NURBS objektumok jelentik. A jelenetben szereplő függönyök kiinduló eleme NURBS volt, ha ezt szerettük volna megnézni a programmal, akkor a Blender figyelmeztetés nélkül kilép. Ezért van szükség egy ellenőrzésre, ha nincs az objektumnak felülete (face), akkor a futás kilép a függvényből. Ez a megoldás biztosítja, hogy scriptünk nem okoz végzetes hibát.

```
try:
      me.faces
except:
      return
```

A scriptben szereplő osztály „rajzol” metódusa felelős az objektum rajzolásáért. Szintén OpenGL utasításokkal találkozunk itt, ahol külön kell kezelnünk azokat a felületeket, melyeket 3 illetve 4 pont határoz meg. Itt határozzuk meg azt is, hogy a megjelenítés drótvázas vagy tömör legyen. Tömör megjelenítés esetén a rajzolás színe felváltva piros és fehér, hogy a szomszédos felületek megkülönböztethetőek legyenek.

```
if harom:
      glBegin(self.mode4)
      for v in face.v:
            glVertex3f(v.co[0], v.co[1], v.co[2])
      glEnd()
else:
      glBegin(self.mode3)
      for v in face.v:
            glVertex3f(v.co[0], v.co[1], v.co[2])
      glEnd()
```

Korábban láthattuk, hogy az objektum megjelenítéséért felelős nemcsak az egyes pontok helyzete, melyek a felületet határozzák meg, de a különböző tengelyek szerinti átméretezés is. Ha [NKEY] segítségével átméretezünk egy tárgyat az nincs hatással az objektumban tárolt pontok helyzetére. Ezt a méretváltozást az objektum más paramétereinél kell kiolvasnunk, és ez szerint módosítani a rajzolás módját.

```
glScalef(parent.SizeX,parent.SizeY,parent.SizeZ)
```

Ez a script hasznos eszköznek bizonyult a rajzolások során, hisz voltak olyan helyzetek és objektumok, amikor két nézőpont sem volt elég, hogy pontosan lássuk a

módosításainkat. Ilyenkor egy külön ablakban láthattam az aktív elemet, tetszőlegesen forgatva pedig nyomon követhettem a változtatásokat.
A script, habár a mellékelt CD-én önállóan is megtalálható preview.py néven, mégis csak Blenderen belül futtatható. A következő fejezetben tárgyalunk egy Python kódot, mely függetlenül futtatható.
Meg kell még említeni, hogy a Blender script-nyleve követi a Python verzió változásait is, így vigyáznunk kell arra, ha scriptet írunk, az más verziókban is működjön. A Python legújabb változata, ami nagyobb változásokat hozott, idén jelent meg. A fenti scriptet habár a Blender 2.03 programhoz írtam, a mai verziókban is működik

5 Python

Utolsó fejezetben szeretném röviden bemutatni azt a programnyelvet, amely egyben a Blender script-nyelve is. Egy aránylag fiatal programnyelv a Python. Fejlesztését 12 évvel ezelőtt Guido van Rossum kezdte Hollandiában, ma már egyre nagyobb teret hódít magának a programnyelv, és egyes vélemények szerint a következő évek Pascal-jának nőheti ki magát. Ugyanis felépítése áttekinthető, érthető és jól tanítható. Ezeket a tulajdonságokat mind tanúsíthatom, mindezek mellett teljes mértékben támogatja a jelen és a jövő technológiáját, az objektum-orientált szemléletet.

A programkódok interpreter módban futnak. Több publikáció csak script-nyelvnek titulálja (de azok közt is az egyik legjobbnak), míg a fejlesztők állítják, hogy semmiben sem különbözik ebből a szempontból a Java nyelvtől. Már abban is hasonlóak, hogy mindkét nyelv platformfüggetlen. Integritása más nyelvekkel megoldott, például Java-val és C-vel is.

A következőkben ismertetett programot azért írtam, hogy a Blender által készíthető panoráma képeket kezelhetővé tegyem, vagy egy felhasználási módszert mutassak. A program felépítése egyszerű, hisz csak az a dolgunk, hogy egy képet betöltsünk, majd hogy figyeljük a felhasználói műveleteket, amelyek szerint mozgatjuk a képet az alkalmazás ablakában. Három fajta felhasználói beavatkozást figyel a program, Esc billentyűre kilép, az oldalirányú kurzorbillentyűkkel forogni tudunk a beltérben, a fel- és lemutató nyilakkal pedig következő képeket tölthetünk be, ha van ilyen.

```
ablak.bind("<Escape>",      kilep)
ablak.bind("<Left>",  fordulas)
ablak.bind("<Right>", fordulas)
ablak.bind("<Up>",    lep)
ablak.bind("<Down>",  lep)
```

A program jelenlegi verziója kevesebb, mint 10 képre készült, hogy az érthetőség érdekében a kód a lehető legrövidebb legyen. Egy továbbfejlesztett verziója a programnak, amikor lényegesen több képpel dolgozunk és azt a látszatot keltjük, hogy lépkedünk a teremben. Ennek a verziónak a fejlesztését felfüggesztettem, mert Win-

dows operációs rendszer alatt az egyes képváltások ideje 1-2 másodpercet is meghaladta, ami a komfortérzetet csökkentette.

A használt képek az aktuális könyvtár /gif alkönyvtárában vannak. A képek betöltésnél ellenőrizni kell, hogy létezik-e az a kép.

```
def betolt(self, fn):
  try:
   self.kep = PhotoImage(file='gifs/'+valt.extend+'%s.gif' % fn)
  except:
   valt.filenumber = fn = 1
   valt.extend = '000'
   self.kep = PhotoImage(file='gifs/'+valt.extend+'%s.gif' % fn)
```

A forgási műveleteknél használok egy „irany” nevű változót, amit a képek váltása mellett is megjegyez. Ez a továbbfejlesztett verzióból maradt, melynek segítségével mindig a nézőpont megmaradhat a „lépkedés” mellett is.

Ahhoz, hogy ezt a programot futtathassuk, szükséges, hogy legyen a gépre telepítve Python. Linux operációs rendszereknél ez nem jelenthet problémát, hisz a telepítő programok része a Python csomag is. Windows rendszerekhez meg kell először szerezni a Python telepítő programját. A CD melléklet viszont tartalmaz egy futtatható állományt, melyet egy py2exe nevű program segítségével fordítottam le. Ezt a programot Thomas Heller írta, és szabadon letölthető a Python hivatalos honlapjáról (www.python.org).

6 Befejezés

Több mint egy évvel ez előtt azt hittem véletlenül találtam rá a Blender modellező programra az Interneten, de ma már tudom, hogy ha bárki a háromdimenziós modellezés témakörben keres valamit a Világhálón, akkor kétségkívül szembetalálja magát ezzel, a maga nemében egyedi programmal. Két tulajdonsága kiemelkedik a számítástechnika világában: ingyenes és platformfüggetlen. Ez a két tulajdonság segített abban, hogy megvalósítsam célkitűzésemet, és olyan, látványtervezésre alkalmas rendszert alakítsak ki, ami csak ingyenes programokból épül fel.

Miközben írtam a dokumentációt, addig rajzoltam a belteret, így folyamatosan követhettem, milyen problémákkal találhatja magát szemben az, aki e dokumentum alapján szeretne betekintést nyerni a Blender rejtelmeibe. Aki már dolgozott más 3D modellező programokkal, láthatta, hogy sok az egyedi megoldás. Én több esetben a 3D Studio Max programhoz hasonlítottam a Blendert, és a végére megállapíthatjuk, hogy többnyire felfogásbeli különbség van a két program között, de felhasználási területük azonos. A különbségek közt említhetjük a rétegeket, melyek megkönnyítik a munkát, a 3D-kurzort, ami nélkülözhetetlen több Blender műveletnél, de talán az egyik legnagyobb különbség egy renderelő modell használatának lehetősége, amit a szakirodalom radiositynek nevez. Ez az eljárás valósághűbb képek renderelését teszi lehetővé a Blenderben.

Az utóbbi években több modellező programmal találkoztam. Köztük több ingyenes programmal is, de a Blender az egyetlen, ami kis mérete, alacsony ára, kis erőforrásigénye, gyors és hatékony megoldásai miatt komoly versenytársa lehet a kereskedelemben kapható modellező programoknak. Egy olyan programmal sikerült megismerkednem, amelyik teljesíti minden követelésemet egy 3D modellezővel szemben.

7 Felhasznált irodalom

Ton Roosendaal, Carsten Wartmann: Tho official Blender 2.0 guide, [Not a Number, 2000]

Aurum – Boca: 3D Studio Max, [Aurum DTP Stúdió, 1997]

Aurum: 3D Studio Max 2, [Aurum DTP Stúdió, 1998]

Mark Lutz: Programming Python, [O'Reilly & Associates, Inc. ,2001]

Dr. Szirmay-Kalos László: Számítógépes grafika, [ComputerBooks Kiadó Kft, 1997]

Elek Tamás, Mihalovics Sándor: Ember-gép kapcsolatok speciális kérdései, [OTDK munka, 2001]

Mark Segal,Kurt Akeley: The OpenGL Graphics System:A Specification (Version 1.2), pdf dokumentum

Fredrik Lundh: An introduction to Tkinter, 1999, pdf dokumnetum

Martin Strubel: No fear of snakes (Python scripting in Blender), 2001, pdf dokumentum

Blender honlap: www.blender.nl

Python honlap: www.python.org

Open Source Initiation honlap: www.opensource.org

8 Melléklet

8.1 A CD melléklet

A diplomamunka része egy CD melléklet. Ezen található a Blender program telepítője, valamint a CD-ről futtatható verziója is. A programok különböző platformokra megírt változatai külön könyvtárban vannak.

A könyvtárakban megtalálható az a Blender dokumentum, mely a belteret tartalmazza, továbbá két Python kód, melyek nyomtatott változata ebben a fejezetben is megtalálható.

A jobb tájékozódást a könyvtárak közt egy html-keretprogram segíti.

8.2 Képek

8.3 Preview.py – Blender script

```
############################################################
# A script forgathato nezetet ad az aktiv objektumrol     #
#                                                          #
# Inditas: Alt-P (mikozben az eger e ablak felett van)    #
#                                                          #
# Futas kozbeni interakciok:                               #
#                                                          #
#   forgatas  - kurzorbillentyuk                           #
#   drotvazas - W                                          #
#   tomor     - S                                          #
#   frissites - R                                          #
#   zoom      - X, C                                       #
#                                                          #
#   kilepes   - Q, Esc                                     #
#                                                          #
############################################################
## Elek Tamas (Szif, Gyor) diplomamunkajanak resze       ##
## 2001. December                                         ##
##                                                        ##
############################################################

import Blender
from Blender.BGL import *
from Blender.Draw import *

class preview:
      def __init__(self):
            self.x     = 0
            self.y     = 0
            self.volt  = 0
            self.oldal = 0
            self.fel   = 0
            self.color = 1
            self.mode4 = GL_QUADS
            self.mode3 = GL_TRIANGLES
            self.zoom  = 1.0

      def rajzol(self, parent):
            if (parent==None):
                  self.volt = 0
                  return

            glScalef(parent.SizeX,parent.SizeY,parent.SizeZ)
            parent = parent.data

            if self.volt==parent.name:
                  glCallList(1)
                  return

            glNewList(1, GL_COMPILE_AND_EXECUTE)
            i=0
```

```
            for face in parent.faces:
                  if self.color: i = i+1
                  glColor3f(1.0, 1.0, 1.0)
                  if i==2:
                        glColor3f(1.0, 0.0, 0.0)
                        i=0
                  try:
                        harom=face.v[3]
                  except:
                        harom=0

                  if harom:
                        glBegin(self.mode4)
                        for v in face.v:
                              glVertex3f(v.co[0], v.co[1], v.co[2])
                        glEnd()
                  else:
                        glBegin(self.mode3)
                        for v in face.v:
                              glVertex3f(v.co[0], v.co[1], v.co[2])
                        glEnd()

            glEndList()
            self.volt =  parent.name


      def forgat(self):
            glRotatef(self.x, 0.0, 1.0, 0.0)
            self.x= self.x+self.oldal
            if (self.x>360): self.x= 0

            glRotatef(self.y, 1.0, 0.0, 0.0)
            self.y= self.y+self.fel
            if (self.y>360): self.y= 0


valami = preview()

def draw():

      Redraw(1)
      zoom = valami.zoom
      glEnable(GL_DEPTH_TEST)
      glMatrixMode(GL_PROJECTION)
      glLoadIdentity()
      glOrtho(-5.0*zoom, 5.0*zoom, -5.0*zoom, 5.0*zoom, 0.1, 10.0)
      glMatrixMode(GL_MODELVIEW)
      glLoadIdentity()
      glTranslatef(0.0, 0.0, -5.0)
      glClearColor(0.3, 0.3, 0.3, 0.0)
      glClear(GL_COLOR_BUFFER_BIT)
      glClear(GL_DEPTH_BUFFER_BIT)

      ob= Blender.Object.GetSelected()
      if(not ob): return
      else: ob= ob[0]
```

```
      me= ob.data
      try:
            me.faces
      except:
            return

      valami.forgat()
      valami.rajzol(ob)

def event(evt, val):
      if (evt==QKEY): Exit()
      if (evt==ESCKEY): Exit()
      if (evt==LEFTARROWKEY): valami.oldal = valami.oldal - 1
      if (evt==RIGHTARROWKEY): valami.oldal = valami.oldal + 1
      if (evt==UPARROWKEY): valami.fel = valami.fel - 1
      if (evt==DOWNARROWKEY): valami.fel = valami.fel + 1
      if (evt==WKEY):
            valami.mode4 = GL_LINE_LOOP
            valami.mode3 = GL_LINE_LOOP
            valami.color = 0
            valami.volt  = 0
      if (evt==SKEY):
            valami.mode4 = GL_QUADS
            valami.mode3 = GL_TRIANGLES
            valami.color = 1
            valami.volt  = 0
      if (evt==RKEY): valami.volt = 0
      if (evt==XKEY): valami.zoom = valami.zoom + 0.02
      if (evt==CKEY): valami.zoom = valami.zoom - 0.02

Register(draw, event)
```

8.4 Gallery.py – Python program

```
from Tkinter import *

class Valtozok:
  filemax = 9
  szog = 5
  pos = -425
  filenumber = 1
  extend='000'

class Gallery(Canvas):
  def __init__(self, parent=None):
    Canvas.__init__(self, parent)
    self.pack()
    self.betolt(valt.filenumber)

  def betolt(self, fn):
    try:
     self.kep = PhotoImage(file='gifs/'+valt.extend+'%s.gif' % fn)
    except:
     valt.filenumber = fn = 1
```

```
      valt.extend = '000'
      self.kep = PhotoImage(file='gifs/'+valt.extend+'%s.gif' % fn)
     self.config(width=1.25*self.kep.height(), height=self.kep.height())
     self.kep1=self.create_image(valt.pos-self.kep.width(), 2, ima-
ge=self.kep)
     self.kep2=self.create_image(valt.pos, 2, image=self.kep, anchor=NW)

  def forog(self, irany):
    if valt.pos < (-0.65 * self.kep.width()) :
     valt.pos = round(0.35 * self.kep.width())
    if valt.pos > (0.85 * self.kep.width()) :
     valt.pos = round(-0.15 * self.kep.width())
    valt.pos =  valt.pos+irany
    self.coords(self.kep1,valt.pos-self.kep.width(),2)
    self.coords(self.kep2,valt.pos,2)

def kilep(event):
  ablak.destroy()

def fordulas(event):
  if event.keysym=="Right":
    irany=-1*valt.szog
  else:
    irany=valt.szog
  terem.forog(irany)

def lep(event):
  if event.keysym=="Up":
    valt.filenumber = valt.filenumber +1
  else:
    valt.filenumber = valt.filenumber -1
    if not valt.filenumber: valt.filenumber = valt.filemax
  terem.betolt(valt.filenumber)

if __name__ == '__main__':
  ablak=Tk()
  ablak.title('Gallery')
  valt = Valtozok()

  ablak.bind("<Escape>",      kilep)

  terem = Gallery()

  ablak.bind("<Left>",  fordulas)
  ablak.bind("<Right>", fordulas)
  ablak.bind("<Up>",    lep)
  ablak.bind("<Down>",  lep)

  ablak.mainloop()
```