Diplomamunka

László Gábor

Debrecen
2010

Debreceni Egyetem

Informatikai Kar

Szabadon választott 3D CAD szoftver alkalmazásai

(AutoCAD és 3ds Max)

Témavezető:

Tomán Henrietta

Egyetemi adjunktus

Készítette:

László Gábor

Programtervező matematikus

Debrecen
2010

Tartalomjegyzék

1. Bevezető

Még emlékszem azokra a napokra, amikor megkértem édesapámat, hogy segítsen betölteni egy kis kazettáról a faltörő játékot a Commodore 64 –es számítógépünkbe, hogy játszani tudjak vele. Már akkor magával ragadott a számítógépes tervezés világa és elhatároztam, hogy én is megírom a saját játékomat. Kitartásom addig vitt el, hogy a Basic nyelv segítségével megrajzoltam egy kezdetleges kétdimenziós autót, amit a képernyő egyik végéből a másikba mozgattam. Az egyetem évei alatt kezdtem el komolyabban foglalkozni a 2D és 3D tervezéssel, amikor önerőből kezdtem megismerkedni a 3dsMax világával és édesapám hatására az AutoCAD iránt sem mutattam közömbösséget.

A diplomamunkámban a 3dsMax és az AutoCAD világát próbálom bemutatni, elemezni, egy képet alkotni róluk.

Ahhoz, hogy jobban megértsük a számítógépes tervező programokat tudnunk kell, mi is az a CADD (Computer Aided Design and Drafting), mire képes, mit várhatunk tőle és mire jó ez nekünk? Diplomamunkám első részében ezekre a kérdésekre próbálok érthető formában választ adni, bemutatva a CADD képességeit.

Ezt követően először az AutoCAD elemzésével foglalkozom. Bemutatom röviden a történetét és leírom, hogy minek is köszönheti dominanciáját a számítógépes tervező szoftverek körében. Ezek után a közelmúlt verzióinak fejlődését veszem szemügyre napjainkig, kiemelve az általam fontosabbnak vélt funkciókat, több szempont szerint csoportosítva őket és kiemelem az egyes szoftverek hatását a fejlődési szakaszokban. Bemutatom az AutoLISP nyelvet és végül egy példa megoldási folyamatát követem végig.

A diplomamunkám másik részében a 3dsMax modellező és animációs programot mutatom be, rövid történetét, sajátosságait és kitérek a parametrikus modellezésre is. Ezután elemzem a 3dsMax főbb sajátosságait, majd egy általam készített 3dsMax modellt mutatok be. Összehasonlítom a 3dsMax és 3dsMax Design programokat.

Diplomamunkám utolsó részében az AutoCAD és 3dsMax kapcsolatát mutatom be, a kapcsolatteremtés lehetőségeit és megválaszolom azt a kérdést, hogy miért is jó, hogy ezek a programok kapcsolatban vannak egymással.

2. Köszönetnyilvánítás

Szeretnék köszönetet mondani tanáraimnak, akik mindent megtettek elméleti tudásom elsajátítása érdekében, témavezetőmnek, Tomán Henrietta egyetemi adjunktusnak, hasznos tanácsaiért, aki mindig készséggel segített bármivel is fordultam hozzá. Végül, de nem utolsó sorban szeretnék köszönetet mondani szüleimnek, hogy tanulmányaim során mindenben támogattak.

3. CADD

3.1. Mi a CAD vagy CADD?

Amikor a CADD-re gondolunk (Computer Aided Design and Drafting, magyarul számítógéppel segített tervezés és vázlatkészítés) olyan kérdések is felmerülnek, amikre nem is gondolunk, amikor a rajztáblán dolgozunk [9]. Nem használunk elengedhetetlen rajzeszközöket: papír, ceruza, vonalzó, radír stb, mégis meg kell terveznünk vagy el kell készítenünk egy rajzot. Tudjuk, mennyire bosszantó lehet, ha akár egy is ezen eszközök közül hiányzik, viszont a CADD-el nincs szükségünk egyikre sem.

A CADD egy elektronikus eszköz, ami lehetővé teszi gyors és pontos rajzok készítését a számítógép használatával. Ellentétben a hagyományos rajzolási módszerekkel, a CADD segítségével hátra dőlve egy kényelmes székben gyönyürű rajzokat készíthetünk úgy, hogy csak a billentyűzet vagy egér gombjait nyomogatjuk. Emellett a CADD-el készített rajzok előnyösebbek a rajztáblán készült rajzokkal szemben. A CADD rajzok arányosak, tiszták és teljes mértékben prezentálhatóak. Az elektronikus rajzok könnyen módosíthatóak és különböző formátumokban bemutathatóak.

Két évtizeddel ezelőtt a CADD-et csak bizonyos mérnöki alkalmazásoknál használták, amelyek nagy pontosságot igényeltek. Mivel a CADD drága volt, csak pár szakember engedhette meg magának, hogy ezzel dolgozzon. Az évek során jelentősen csökkentek a számítógépek és alkalmazások ára, így egyre több szakember tudta kihasználni a CADD előnyeit.

Manapság több száz CADD program létezik. Egyesek csak általános rajzkészítésre használatosak, mások specifikus műszaki alkalmazásokra fókuszálnak. Vannak programok, amelyek lehetővé teszik 2D vagy 3D rajzok készítését, renderelést, árnyékolást, műszaki számításokat, épülettervezést, csővezeték-hálózat tervezést, projektmenedzsmentet, stb. Tulajdonképpen minden műszaki tudományághoz, amire csak gondolni lehet, létezik egy CAD program.

A CADD elsődlegesen single-line (egyvonalas) tervezést valósít meg, nagyon korlátozott képességei vannak, hogy művészi hatást érjünk el vele, mindazonáltal a CADD 3D és render képességei lenyűgözőek. Egy objektumról 3 dimenziós modellt készíthetünk és bármelyik

szögből nézhetjük, megfelelő árnyékolással és rendereléssel tökéletesnek tűnő képet készíthetünk.

3.2. A CADD-del szemben támasztott elvárások

Elképesztő dolgokra képes a CADD, amikről toll vagy ceruzával készült rajzok esetén nem is gondolnánk, hogy lehetséges. A következő képességek csak néhány azok közül, amik a CAD-et hatásos, erős eszközzé teszik:

- Prezentációk
- Rugalmasság a módosításban
- Egység és pontossági szintek
- A rajzok tárolása és hozzáférhetősége
- CADD rajzok megosztása
- Projektjelentés
- Műszaki analízis
- Computer Aided Manufacturing (CAM) – számítógép segítette gyártás
- Design
- Add-on programok

3.2.1. Prezentációk

Nagyszerű rajzokat készíthetünk több száz szín, vonaltípus, sablon, prezentációs szimbólum és szövegstílus segítségével. Miután befejeztük a prezentációt és nem tetszik valami, gyorsan tudunk változtatni rajta. Csak pár egyszerű lépésbe kerül, hogy megváltoztassuk a szövegstílust, színt vagy vonaltípust és egy új másolatot készíthetünk a rajzról.

Számos előre megkonstruált prezentációs szimbólum és sablon található a CADD-ben, ami feljavíthatja a rajzunk kinézetét. Például, egy tereptervező faszimbólumokat, cserjéket, járdákat, emberi alakzatokat és egyéb környezeti elemeket szúrhat be, hogy egy terepet, közterületet megtervezzen. Hasonlóan, egy építész többek között előre elkészített ajtó-, ablak-, bútorelemeket, szimbólumokat használhat egy prezentáció elkészítésénél.

Ezeken kívül hatásos prezentációk papíron való előkészítéséhez, használhatjuk a CADD-ot vetítéses prezentációk készítéséhez. A számítógépünkhöz csatlakoztathatunk egy projektort és

kivetítve is bemutathatjuk az ötleteinket. Számos CADD program lehetővé teszi animált képek készítését. Illusztrálhatjuk, hogyan is nézne ki egy épület, miközben végigsétálunk rajta vagy az egyes alkatrészek hogyan működnek, ha egy gép üzemel.

3.2.2. Rugalmasság a módosításban

A CADD biztosítja azt a rugalmasságot, hogy gyors változtatásokat végezhessünk a rajzokon. A rajz bármely részét hajszálpontossággal törölni tudjuk. Pár másodpercig tart egy olyan munka elvégzése, ami a rajzlapon órákat venne igénybe. Legtöbb esetben nem is kell törölnünk, hogy a módosítást végre tudjuk hajtani, ugyanis át tudjuk rendezni a rajz meglévő komponenseit, hogy megkapjuk az új alakzatot. Ez lehetővé teszi, hogy minimális erőfeszítéssel elemezni tudjuk a tervezési opciókat.

Néhány módosítási opció, képesség, amit végre tudunk hajtani a CADD-del:

- Rajzelemek módosítása vagy törlése
- A rajz egyes részeinek nagyítása vagy kicsinyítése
- Egyik rajz hozzáadása a másikhoz
- A rajz kinyújtása, hogy megfeleljen az új méreteknek
- A rajzelemek többszörösítése
- A szöveg méretének és stílusának megváltoztatása
- A mérési egységek, pontosság, dimenziók stílusának megváltoztatása

3.2.3. Egység és pontossági szintek

A CADD lehetővé teszi, hogy nagy pontossággal dolgozzunk. Ha nagy pontosságú geometriai alakzatokat akarunk készíteni, a CADD a megoldás. Segíthet elkerülni időigényes matematikai számításokat. Különböző mérési egységekkel dolgozhatunk, úgy, mint építészeti, földmérő, műszaki vagy tudományos mérési egységek. Ezek a mérési egységek, amit a szakemberek gyakran használnak, standard formátumban vannak feltüntetve.

Példa: Amikor műszaki egységekkel dolgozunk, megadhatjuk, hogy minden méret hüvelykben, centiméterben vagy méterben legyen megadva. Hasonlóan szögek mérőegységeit is használhatjuk: fok, perc, másodperc vagy radián.

A mérőegységeket nagyon nagy pontosságra beaállíthatjuk, 1/1000 hüvelyk pontossággal dolgozhatunk, viszont ilyen pontosság ritkán szükséges. Gyakran kisebb pontosságra kell állítanunk, hogy elkerüljük a vonalak nem kívánatos töredezettségét.

3.2.4. A rajzok tárolása és hozzáférhetősége

Gyors és kényelmes a CADD rajzok számítógépen való szervezése, tárolása. Több ezer rajzot tárolhatunk a merevlemezen és bármelyiket másodpercek alatt meg tudjuk nyitni.
A számítógépes fájlrendszer előnyei a hagyományos, papír alapúval szemben:

- A számítógépen sokkal nagyobb szervezettséget lehet elérni, könnyen átlátható
- Sokkal kisebb helyen lehet tárolni, mivel merevlemezen van az információ és nem papír formában
- Egy elektronikus rajz sosem halványul el, nem lesz régi és akármikor egy új példányt tudunk kinyomtatni belőle.

3.2.5. CADD rajzok megosztása

Az elektronikus rajzokat meg lehet osztani bizonyos számú felhasználóval, ezzel lehetővé téve, hogy összehangolják a munkájukat és csapatként dolgozzanak. Ehhez csupán a számítógépeket egy hálózatba kell kapcsolni.

Példa: Egy építészeti projektben különböző szakemberek, mint építészek, mérnökök és építészeti menedzserek ugyanazokat a rajzokat használhatják, hogy összehangolják a munkájukat. Ha valamilyen módosítás történik a rajzokon, ez az információ automatikusan elérhető lesz minden munkatárs számára.

A modemek és Internet használatával lényegesen egyszerűbb lett az információ megosztása. A legtöbb építész és mérnök napjainkban elektronikus úton osztják meg rajzaikat.

Ugyanakkor az Internet lehetővé tette, hogy egy weblap használatával publikálhassuk rajzainkat és CADD projektekben együtt működjünk. A legtöbb CADD program olyan speciális funkciót is tartalmaz, amellyel a rajzokat olyan formátumba exportálhatjuk, amelyeket meg lehet tekinteni az Interneten.

3.2.6. Projektjelentés

A számítógépet használhatjuk projektjelentések elkészítésére, mint pl. helyzetjelentés vagy mennyiségi és költségbecslésekre. A CADD adatbázisának a képességeit felhasználva, a nem-grafikus információkat (szöveg, érték) összekapcsolhatjuk a rajz grafikus elemeivel. A nem-grafikus információkat az adatbázisban tároljuk, ezeket felhasználhatjuk jelentések készítésére.

Például: Egy építész egy rajzban ajtókhoz és ablakokhoz rendelt szöveges jellemzőihez teremti meg a kapcsolatot (link). Ezen jellemzők leírhatják az ajtó méretét, anyagát, költségét stb. Ezen jellemzőkkel teremtett kapcsolatok segítségével a számítógép automatikusan előállíthat egy jegyzéket az ajtókhoz, amely kilistázza az ajtókat és ablakokat a rajzban.

A nem-grafikus információkat közvetlenül hozzákapcsolja a rajzobjektumokhoz. Amikor a rajzon egy változtatás történik, a jelentésben az értékek automatikusan frissülnek. Ez egy hatásos eszköz nagy projektek kezelésekor.

Létezik egy speciális szoftverkategória, Computer Aided Facility Management (CAFM), amelyet menedzsment előállítására terveztek. Ezen programok lehetővé teszik, hogy CADD rajzokat importáljunk és tulajdonságokat kapcsoljunk hozzá, hogy egy adatbázist hozzunk létre, amely segítségével nyilvántarthatjuk a tereket, költségeket, embereket, felszereléseket, bútorokat, épület karbantartási jegyzéket, stb.

3.2.7. Műszaki analízis

Létezik programok egy külön kategóriája, Computer Aided Engineering (CAE), amely műszaki elemzéseket tud végezni CADD rajzokon. A CAE programoknak egy sor alkalmazása van szerkezeti tervezetekhez, építőmérnöki munkákhoz, gépészethez és villamosmérnöki munkákhoz.

Példa: Egy szerkezeti mérnök arra használhatja a CAE programot, hogy tesztelje egy épületben a szerkezeti elemek tervét, vázlatát. A mérnök azonnal analizálni tudja a szerkezeti elemekre gyakorolt behatást, amikor a szerkezetre egy eltérő nyomást gyakorolunk, vagy ha az elemek közötti távolságot megváltoztatjuk. Hasonlóan, léteznek olyan programok gépészeknek, amelyek segítségével tesztelhetik a gépek összeszerelését, szerkezetét. A gépész készíthet egy elektronikus modell prototípust és tesztelheti anélkül, hogy elkészítené a valós, fizikai modellt.

Fejlettebb műszaki programok azt is lehetővé teszik, hogy összekössék a számításokat a CADD rajzokkal. Ez a képesség parametric design (paraméter alapú tervezés) néven ismert, amely lehetővé teszi, hogy a számítógép automatikusan frissítse a rajzokat, ha a hozzárendelt számítások megváltoznak vagy fordítva.

3.2.8. Computer Aided Manufacturing (CAM) – számítógép segítette gyártás

A CADD jelen van egy másik műszaki ágban is, Computer Aided Manufacturing (CAM). A CAM egy megszokott gyártási módszer, amelyet nagy vállalatok alkalmaznak. A CADD-et és a gyártási terveket gyakran egy rendszerbe integrálják, amit CAD-CAM –nak neveznek. Ezek a rendszerek a CADD rajzokat CAM programokba importálják, mellyel automatizálják a gyártási folyamatot.

Példa: Egy mérnök megrajzolhatja egy gép valamely részét a CADD segítségével. Ezt a rajzot betöltik egy Computer Aided Engineering (CAE) programba, műszaki analízisek céljából. Amikor a tervezés befejeződik, a rajzot betöltik egy CAD-CAM rendszerbe, ami a CADD rajzból vett numerikus adatokat használja az aktuális gyártáshoz.

3.2.9. Design

A CADD kényelmes eszközöket biztosít ahhoz, hogy terveket készíthessünk bármilyen műszaki tudományág területén. Használhatjuk építészeti tervezéshez, terep és park illetve belsőtér tervezéséhez, gépészetben, villamosmérnöki munkálatokban, ipari tervezésben, csővezeték illetve elektromos vezetékek tervezéséhez, ruhatervezéshez, terméktervezéshez stb.

A CADD vázlatkészítési képességei bőséges eszközöket és erőforrásokat biztosítanak, hogy egy tervet elkészítsünk. Nagy pontossággal tudunk terveket készíteni és könnyedén módosíthatjuk őket, ez lehetővé teszi, hogy gyorsan terv alternatívákat készítsünk.

A CADD rajzszerkesztési képességein felül, léteznek olyan programok, amelyek terveket ellenőrizhetnek vagy akár újakat is készíthetnek. Ezen programok mesterséges intelligenciát használnak, hogy “gondolkodjanak” és saját döntéseket hozzanak a tervezés során. Csak pár ilyen CADD program létezik. Az ilyen tervező programok általában nem generikusak, gyakran külön-megírt programok egy bizonyos feladat végrehajtására.

3.2.10. Add-on programok

Léteznek különböző különálló programok, amelyek növelik a CADD "erejét". Az add-on programok úgy működnek, mint a CADD kibővítései, amelyek jellegzetes feladatokat teljesítenek, hajtanak végre. Napjainkban a népszerű CADD programokhoz több száz add-on program létezik. Egy építészeti add-on program lehetővé teheti, hogy azonnal, egy lépésben például ajtókat, ablakokat, konyhákat, fürdőszobákat rajzolhassunk. Egy árnyékoló és renderelő program a 3D-s képek megjelenését fokozni tudja. Egy vízvezeték-rendszer tervező program speciális funkciókat tartalmaz, amivel csöveket, csatornákat rajzolhatunk.

A legtöbb CADD programot fejlesztő és tervező cég a CADD programokat különálló részekben értékesítik. Egy bizonyos összegért értékesítik a program egy alapverzióját, amely aztán olyan modulokkal kibővíthető, amelyek speciális funkciókra alkalmasak.

4. AutoCAD

Az AutoCAD első verziójának, a Release 1-nek 1982 decemberében történt megjelenése óta több mint két évtized telt el [5]. Az AutoCAD-et fejlesztő Autodesk Inc. azóta a világ negyedik legnagyobb szoftverfejlesztő vállalkozásává fejlődött. Napjainkban az Autodesk szoftvertermékek a CAD (Computer Aided Design/Drafting) piac majdnem 80%-át lefedik, a maradék 20% megoszlik közel 400 féle más szoftver között. Hogy minek köszönheti az Autodesk a sikert? Elsősorban annak, hogy az AutoCAD egy nyitott architektúrájú, grafikus alaprendszer, amely rugalmasságánál fogva lehetővé teszi a rendkívül szerteágazó mérnöki területek kiszolgálását. A kezdetben csak síkbeli szerkesztési eszközökkel rendelkező AutoCAD mára egy mérnöki operációs rendszerré fejlődött, amelyre alapozva az Autodesk a különböző mérnöki területekre szakmai alkalmazásokat fejlesztett ki. Ezek a szakmai alkalmazások is nyitottak, melyeket a felhasználó a saját munkastílusához, igényeihez igazíthat az erre szolgáló eszközökkel. Ezeken túlmenően a világon független fejlesztők tízezrei fejlesztenek speciális alkalmazásokat az AutoCAD alaprendszerben, felhasználva elsősorban az AutoCAD részét képező fejlesztő eszközöket, az AutoLISP-et, VisualLISP-et, DCL-t, Visual Basic Application-t, ObjectARX-et. Az AutoCAD alkalmazások az objektumorientált szoftverfejlesztési technológiával fejlesztett legnagyobb alkalmazások közé tartozik. Ezek mellett külső szoftverekkel is fejlesztenek AutoCAD alkalmazásokat.

Az Autodesk termékek ilyen mértékű dominanciája azzal is magyarázható, hogy folyamatosan figyeltek a nagy tömegben elterjedt PC-k teljesítőképességére és az operációs rendszerekre. Az első verziókat már IBM XT gépeken, MS DOS operációs rendszer alatt lehetett futtatni. Készültek ezen kívül UNIX és Windows operációs rendszer alatt futó verziók is.

Az Autodesk termékek dinamikus fejlődése minőségileg új szakaszba lépett a korábbinál szorosabb együttműködést eredményező, Microsoft-tal kötött exkluzív szerződéssel. A szerződés aláírása óta az Autodesk ma már csak a világ legelterjedtebb operációs rendszere, a Windows alá fejleszt.

A világ első és negyedik legnagyobb szoftverfejlesztő vállalkozásának együttműködése olyan szoftverfejlesztői kapacitás és tőke koncentrációt jelent, amely belátható ideig meghatározó

szerepet fog játszani a szoftver és a CAD világ fejlődésében. A műszaki gyakorlat és az ezt szolgáló műszaki szakoktatás területén ezért célszerű Autodesk szoftvereket alkalmazni.

Az Autodesk termékek sikerében fontos szerepük volt a marketing szakembereknek is. Kiknek is jutott volna eszükbe másoknak, mint az Autodesk és persze a Microsoft illetékeseinek, hogy a szoftvertermékeik elnevezésével kihasználják a 2000. év reklámértékét. Az 1999-ben megjelent szoftvereik, az AutoCAD 2000, AutoCAD Map 2000, AutoCAD Mechanical 2000 illetve Office 2000 üzlettartalmát könnyű megérteni. Ezek a multinacionális óriáscégek akkor eldöntötték, hogy ezeket a szoftvereket fogjuk használni a 2000. év után is. Az Autodesk szoftverek fejlesztési üteme napjainkban is töretlen. Már 2003. első félévében forgalomba hozták az AutoCAD 2004-et és a rá alapozó szakmai alkalmazásokat.

Eredményes marketingfogásnak bizonyult a különböző Series szoftvercsomagok összeállítása, közöttük az Inventor Series 7, amely tartalmazza az AutoCAD 2004 az AutoCAD Mechanical 2004, az Autodesk Mechanical Desktop 2004 és Inventor Professional 7 szoftvereket. Ez egyetlen integrált termékben biztosít rugalmas, magas színvonalú eszközöket az adott tervezői környezetben felmerülő 2D és 3D gépészeti tervezési feladatok megoldására. Napjainkban az Autodesk Inventor Series az egyik legnagyobb példányszámban eladott 3D tervezőszoftver a világon.

Az Autodesk kiváló eszközök széles választékával biztosítja vezető helyét a CAD szoftverek piacán. Jól szemlélteti az alábbi 1. táblázat, hogy az Autodesk a műszaki tervezés minden területét képes kiszolgálni.

Szoftvertermék neve	Alkalmazási terület
AutoCAD	Minden területen használható alaprendszer
Autodesk Architectural Desktop	Építészeti tervezés
Autodesk Building Systems	Épületgépészeti, épületvillamossági, csővezetéki és tűzvédelmi rendszerek tervezése
Autodesk Revit	3D építészeti tervezés
Autodesk Civil Design	Építőmérnöki tervezés (út, vasút, közlekedés)
Autodesk Civil Series	Integrált csomag építőmérnöki tervezéshez
Autodesk Envision	Térképészeti és tervadatok lekérdezése, elemzése, bemutatása
Autodesk Land Desktop	Táj, kert, kultúrmérnöki létesítménytervezés, terepmodellezés, nyomvonalak és telkek kezelése
Autodesk Map	Térinformatika
Autodesk Map Series	Térképkészítés, karbantartás, elemzés
Autodesk MapGuide	Térinformatikai és digitalis tervadatkezelő alkalmazások fejlesztése, kezelése az Interneten, Intraneten és terepen végzett munkához
Autodesk OnSite Desktop	2D és 3D adatelemzés, kezelés a terepen
Autodesk Raster Design	Raszter-vektor konverzió, digitalis fotók feldolgozása
Autodesk Survey	Geodézia
AutoCAD Mechanical	2D gépészeti tervezés
Autodesk Mechanical Desktop	2D és 3D gépészeti tervezés
Autodesk Inventor Pro.	3D gépészeti tervezés
Autodesk Inventor Series	Integrált csomag gépészeti tervezéshez
Autodesk Vault	Adatkezelő program az Inventorhoz
IGES, STEP, VDA-FS translátorok	Gépészeti alkalmazások (CAM)
3D Studio Max	Látványtervezés, grafika, animáció
Autodesk VIZ	Látványtervezés, ipari formatervezés

(1. táblázat)

A felsoroltakon kívül további kiegészítő Autodesk termékek állnak a felhasználók rendelkezésére. Az Autodesk szoftverekhez független fejlesztők által készített programokkal a CAM (Computer Aided Manufacturing), vagyis a számítógéppel segített gyártás területén felmerülő minden igény kielégíthető. Az eredeti angol szoftvertermékeknek 15 nemzeti verzióját készítették el, közöttük a magyar verziókat is.

A mérnöki tervezés szakemberei építészeti, gépészeti és egyéb területeken megjelenése óta rajzok millióit készítették el a program segítségével világszerte. Ez köszönhető annak a szívós fejlesztői munkának is, amelynek révén a programot létrehozó Autodesk mintegy másfél évenként újabb verzióval rukkol elő. Újabban ezt a ciklusidőt is egy évre leszorították, ennek az AutoCAD 2011-es verziója már a hetedik terméke.

4.1. Az AutoCAD fejlődési szakaszai

Az alábbiakban szeretném bemutatni az AutoCAD fejlődési szakaszaiban általam fontosabbnak vélt pontokat a 2004-es verziótól kezdve napjainkig [3]. Megpróbálom belátásom szerint több szempontot vizsgálva felbontani, osztályozni őket. Először a tervezés és modellezés területén bevezetett újításokat vizsgálom meg, utána felsorolom azokat az eszközöket, amelyek megkönnyítették a munkafolyamatokat és egyre inkább felhasználóbarátabb környezetet teremtettek, majd elemzem a programnak a megosztást és közzétételt elősegítő képességeit és kitérek a Microsoft AutoCAD-re történő esetleges behatásaira.

4.1.1. Tervezés és modellezés

Lényeges mérföldkő az AutoCAD 2004-es verziója, amelytől kezdve megszűnt a legfeljebb 256 szín alkalmazását engedő kötöttség. Ez a verzió akár 64 k (65536) szín felhasználását is lehetővé tette, színátmenetes kitöltéseket készíthettünk két különböző szín vagy ugyanazon szín különböző árnyalatai között. A színeket választhatjuk a nyomdászatban elterjedt (például PANTONE) színskálákból is. Így az AutoCAD programon belül, más szoftverek alkalmazása nélkül is minőségi prezentációs grafika állítható elő. A renderelt térbeli izometrikus nézeteket kinyomtathatjuk.

Új lehetőségekkel gazdagodott a straffozási eszköztár. A szokásos rajzobjektumokhoz hasonlóan metszhetők lettek a sraffozási objektumok is. A sraffozást alkalmazhatjuk hézagos

határvonallal rendelkező objektumok esetében is, vagyis a kitöltő minta „nem folyik ki”. Ezeken felül módosíthatjuk a sraffozási határvonalat, kiszámíthatjuk a sraffozási területet, készíthetünk különálló sraffozásokat, újraépíthetjük a sraffozási határvonalat.

A gyorsabb parancsbevitel érdekében a kurzor mellett megjelenő parancssor is használhatóvá vált. Az új, dinamikus adatbevitelnek megfelelően a parancsok paraméterezése a méretvonalak méretjelzéseinek módosításához hasonlóan történhet.

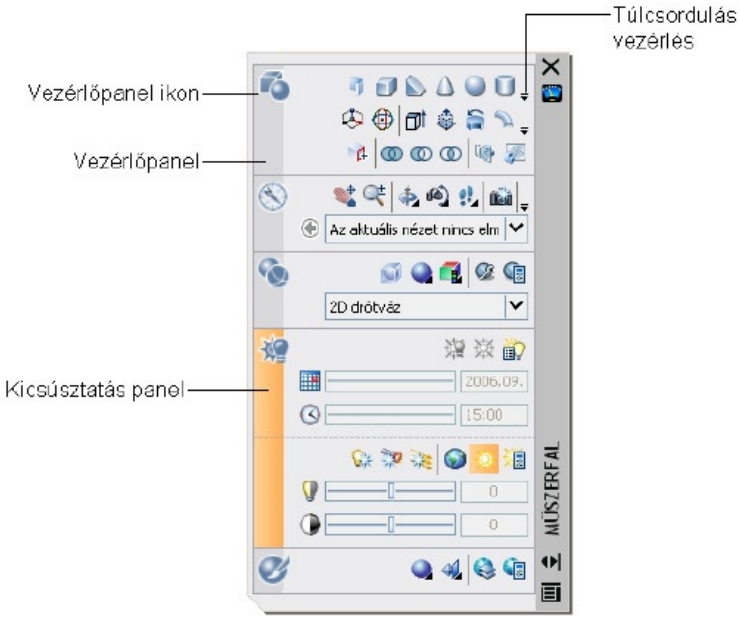
A tervezési munka hatékonyságát jelentősen növeli a blokkadatok kinyerése és az új, dinamikus blokk. Ez utóbbiakat csak az AutoCAD 2006-ban hozhattuk létre, az AutoCAD LT 2006-os változatában csak alkalmazhatjuk az itt készített blokkokat, amelyből mindjárt rengeteg mintát is kapunk. Az AutoCAD LT 2007-ben már szintén létrehozhatunk dinamikus blokkokat. Ezekkel a blokkok parametrikusan illeszthetők be, nem kell például egy hatlapfejű csavart az összes járatos méretben megtervezni, eltárolni, elegendő egyetlen dinamikus példány, amelynek beillesztésekor listából kiválasztva adjuk meg a szabványos méretet vagy elnevezést.

A 2010-es verzióban jelentősen továbbfejlesztették a térbeli modellezést. Az új technikákkal szabad stílusban, lendületes 3D modellek készíthetők (igaz, a korábbiaknál sokkal erőforrás-igényesebben). Új, soklapú hálóobjektumokat (hasáb, henger, gúla, kúp, ék, tórusz, gömb) készíthetünk, amelyek élei egyesíthetők, felületük simítható, tovább darabolható. A program támogatja a 3D nyomtatással készülő prototípusgyártást is.

A parametrikus tervezéshez bevezették a kényszereket, amelyekkel egy rajzelemhez kapcsolt más rajzelemek követik egymáshoz viszonyított helyzetüket, méretüket. A kényszerek létrehozása másfajta, a kényszereket sértő módosítások létrehozását kizárja.

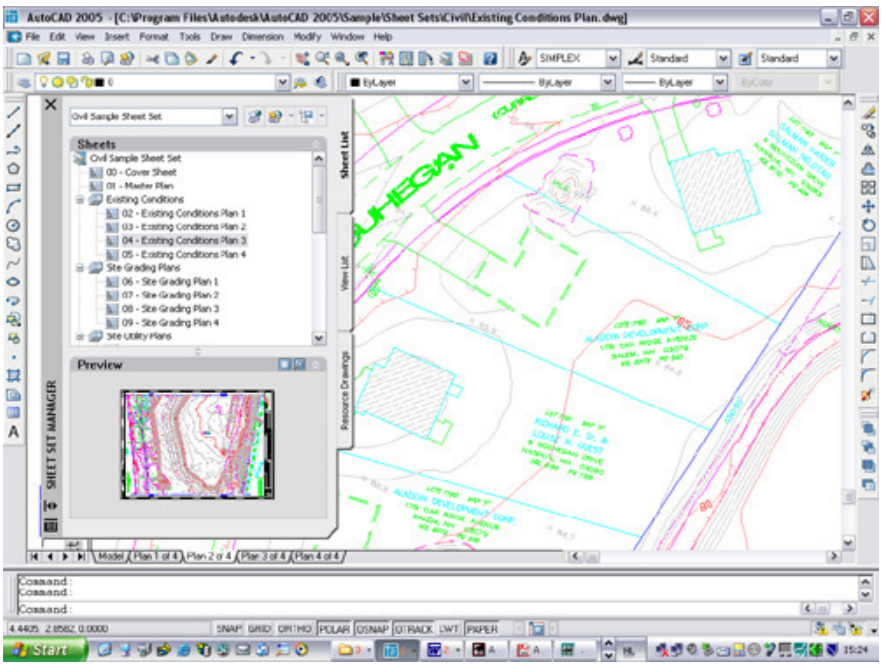
4.1.2. Felhasználócentrikus tulajdonságok

Azért, hogy megkönnyítsék a felhasználók munkáját, folyamatosan fejlesztették, újították a kezelőfelületet, hogy azon a mindennapi munka során leggyakrabban használt elemek könnyen elérhetők legyenek. Az új eszközpaletták testre szabhatók, kiegészíthetők a munkacsoportban használt elemekkel, blokkokkal, kitöltési mintákkal, LISP rutinokkal. Megjelent a műszerfal (1. ábra), mely egy olyan speciális paletta, amely a feladat alapú munkaterülethez kapcsolt nyomógombokat és szabályzókat jelenít meg, főként a 3D modellezéshez, megjelenítéshez és rendereléshez használatos.



(1. ábra) Műszerfal

Létrehozták a lapkészlet kezelőt (2. ábra), amely több, esetleg különböző tervezőktől, szakágaktól származó rajzfájl egyetlen tervezési projekt lapkészletbe foglalását segíti.

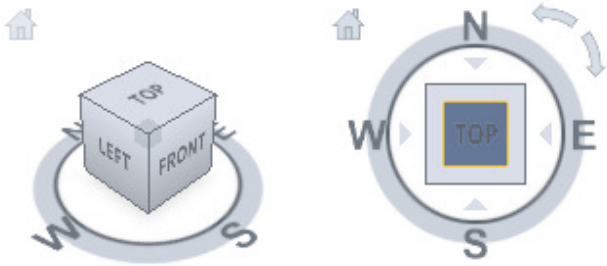


(2. ábra) Lapkészlet kezelő

A különösen komplex projektek esetén, lapkészleten belül alkészleteket is kialakíthatunk. A lapkészletek manuális kezeléséhez a laplista címeit tartalmazó, könnyen frissíthető táblázatot készíthetünk. A lapkészlet rendezéséhez rajzcsoportosító eszközöket kapunk. A lapkészlet teljes egészében közzé tehető, elküldhető e-Küldeményként és archiválható. A lapkészlet kezelő a nézeteket a rajzokhoz hasonlóan kezeli.

Új megjelenítést és navigálást segítő eszközök jelentek meg, a ViewCube (3. ábra) és a Steering Wheel (4. ábra).

(3. ábra) Steering Wheel



(4. ábra) ViewCube

4.1.3. Megosztás és közzététel

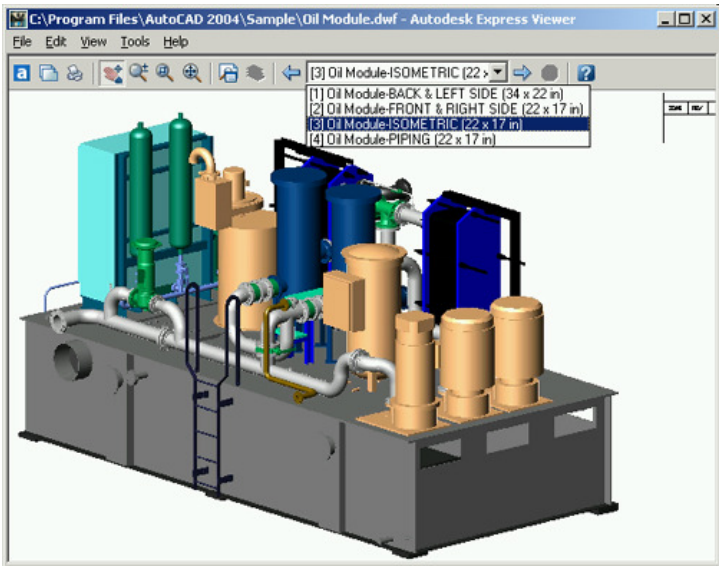
4.1.3.1. DWF

A DWF (Design Web Format) szabvány egy szabadon használható, biztonságos fájlformátum, amit az Autodesk hálózatos terjesztéshez fejlesztett ki [7]. Ebben a formátumban a rajzok mérete kisebb, mint a fele az eredeti méretnek, így az Internetes továbbítás gyorsabb és biztonságosabb. A formátum legújabb verziója a DWF 7. Az újabb verziókat (DWF6-tól) már úgy tervezték, hogy segítségével már teljes rajzösszeállításokat lehessen készíteni. Így egy DWF fájlban több rajznak több elrendezése is jelen lehet egy komplett tervdokumentációt kielégítve. További lehetősége a formátumnak, hogy a fájl jelszóval védhető.

A DWF formátum új lehetőségeit a rajznézet összeállítások közzétételével (Publish Drawing Sheets) lehet elérni (Publish utasítás).

4.1.3.2. Autodesk DWF Viewer

Az Autodesk DWF Viewer (5. ábra) egy kisméretű, hatékony és ingyenes alkalmazás a DWF formátumú fájlok megtekintésére és nyomtatására. Az alkalmazás telepítő fájlja kisebb, mint 2 MB, így akár e-mail-en is továbbítható, vagy letölthető az Internetről is. Telepíthető akár önállóan, akár mint ActiveX vezérlőeszköz a Microsoft Internet Explorer böngészőhöz. Mivel a termék ingyenes, szabadon lehet továbbítani bárkinek, akinek rajzainkat be szeretnénk mutatni. A programmal megnyitható minden DWF formátumú fájl.



(5. ábra) Autodesk Express Viewer

Lényeges az AutoCAD Adobe-féle PDF formátum támogatása. A kimeneti oldalon: az AutoCAD szoftverből a rajzot közvetlenül PDF-fájlokban is közzétehetjük. Ennek köszönhetően a rajzok megosztása rendkívül könnyű a felhasználók legszélesebb köre felé, hiszen az ingyenes Adobe (Acrobat) Reader, mint PDF-olvasóprogram az egyik legjobban elterjedt szoftver a világon.

4.1.4. Kapcsolat és hasonlóság a Microsoft termékekkel

A Microsoft Office programjaihoz hasonló módon, több megnyitott rajz külön elemként is megjeleníthető a Windows tálcáján. Mindezek növelik a rajzkészítés hatékonyságát.

A bekezdésszöveget a Microsoft Word-hoz hasonló helyi szerkesztőben módosíthatjuk, alkalmazhatunk a szövegelemen felsorolás-jelölést és sorszámozást is.

A rajzhelyreállítás kezelő a Microsoft Office programjaiban megszokott módon, a fatális rendszerösszeomlások, áramszünet esetén nyújt segítséget a rajz utolsó (szerkesztés közbeni automatikus mentéssel rögzített) változatához történő visszatérésben.

A táblázatokkal kapcsolatos igen fontos újítás volt, hogy a táblázatadatokat a közismert Microsoft Excel táblázataiból csatolva illeszthetjük be, így az adatkapcsolat biztosított a táblázat és a rajz között. Bármely módosítás egyszerűen átvezethető a két fájl között. Az összes csatolt adat egyszerűen frissen tartható és szinkronizálható. Nagy jelentőségű az új Adatkiemelés varázsló, amellyel a rajz objektumainak (blokkokat, attribútumok is) adataiból kigyűjtött tulajdonságadatokat Excel munkalaphoz csatolhatjuk, vagy exportálhatjuk. Az oszlopok átrendezhetők, elrejthetők, tartalmuk sorba rendezhető.

Az AutoCAD 2009-es verziójában teljesen átdolgozták a felhasználói felületet. Ennek leglényegesebb eleme a Microsoft Office 2007-ben megjelent szalag, gyakorlatilag egy, a menüsor szerepét átvevő, a címsor alatt rögzített paletta, amely a legfontosabb parancsokat tartalmazza. A Microsoft alkalmazásaival szemben viszont itt a szalag testre szabása egyszerű, sőt, akik ragaszkodnak a korábbi megoldáshoz, könnyen elérhetik a hagyományos menüt is.

Ugyancsak ebben a verzióban átdolgozták a kommunikáció központot is. A Microsoft programjaiban bevett gyakorlatot követi a műveletrögzítő, amelynek segítségével a gyakran ismétlődő műveletsorok parancsállományba rögzíthetők. Ezek a makrók aztán később elővehetők és újrafuttatásukkal jelentősen meggyorsítható a munka.

4.2. AutoLISP és Visual-LISP

Az AutoLISP a Lisp programnyelvnek egy változata, melyet az AutoCAD számítógéppel segített tervezőrendszer számára fejlesztettek ki. Az AutoLISP az AutoCAD programon belül használható és arra szánták, hogy makrókat lehessen készíteni vele a program testreszabása érdekében. Az AutoCAD program maga is tartalmaz AutoLISP-ben megírt részeket. Az AutoLISP-en keresztül el lehet érni az AutoCAD teljes adatállományát, változóit és parancsait, ezeket módosítani lehet és önálló, AutoCAD-en belül futó alkalmazásokat lehet készíteni vele. Az AutoCAD klónok egy részében is működnek az AutoLISP makrók. Az AutoLISP értelmezőt

használ, így lehetőség van az AutoCAD parancssorából soronkénti végrehajtásra is használni, mint a legtöbb interpreterrel rendelkező nyelv esetében, sőt egy AutoCAD parancs során az adatok helyébe kiértékelhető AutoLISP kifejezéseket is lehet használni.

Egyes Autodesk alkalmazásokban is használható az AutoLISP, ilyen például az Autodesk Map 3D és az Autodesk Architectural Desktop. Az AutoCAD olcsó, korlátozott képességű változatán, az AutoCAD LT-n az AutoLISP nem működik. Az AutoLISP legnagyobb hátránya, hogy az AutoCAD környezetén kívüli eszközöket (adatokat és programokat) csak igen korlátozott módon lehet elérni vele.

A Visual-LISP az AutoLISP lényegesen továbbfejlesztett változata, ez integrált fejlesztő rendszert (IDE), debuggert, és compilert is tartalmaz. Ezt az eszközt a Basis Software cég fejlesztette és az AutoCAD 2000-től a program részévé tette az AutoLISP helyett. Ez a szoftver amellett, hogy a régi forrásnyelvi programok és egyéb eszközök használatát is lehetővé teszi, VBA-szerű hozzáférést ad az AutoCAD objektum modelljéhez, reaktorjaihoz és általános ActiveX szolgáltatást nyújt.

5. AutoLISP

Az AutoLISP néhány fontos tulajdonsága:

- AutoCAD rendszer hatékonyságát növelheti;
- Rutin feladatokat automatizálhat;
- Módosítani lehet rendszerváltozókat;
- Módosítani lehet rajzokat, rajzelemeket;
- AutoCAD legrégebbi programozói környezete;
- Sok AutoCAD parancs maga is AutoLisp függvény.

Az AutoLISP megtalálható az AutoCAD minden egyes változatában, és minden új munkaszesszió indításakor automatikusan betöltődik [10]. Az AutoLISP utasítássor egy kerek zárójel-pár (zárójel-párok) közé írt karakterek sorozata. A zárójelben található karakterek sorozata az úgynevezett szimbolikus kifejezésekből, vagy röviden s-kifejezésekből áll. A s-kifejezések az AutoLISP programok alap utasításai, alapvetően zárójelek közé zárt listák

formájában használjuk őket. Amikor az AutoLISP s-kifejezésekre bukkan, azokat felméri és kiértékeli, a kiértékelés eredményét pedig visszafordítja az AutoCAD parancssorába.

A programozás során az egyik leggyakoribb hiba, amit elkövetünk, hogy nem megfelelő számban párosítjuk a zárójeleket, vagyis nem követ minden nyitott zárójelet megfelelő számú bezárt zárójel.

5.1. Atomok és listák

Két alapvető AutoLISP objektum létezik: atom és lista. Az atomok egyszerű, a listák pedig komplex elemek. Ezek az AutoLISP alapelemei.

- Listák - atomok sorozata zárójelek között.
- Atomok - "minden, ami nem lista" és a nyelv eszközeivel nem bontható tovább
  - lehetnek konstansok (számok, karaktersorozatok, stb.) vagy szimbólikus atomok (változók)
- Speciális atomok - pl: nil (hamis érték, üres lista), t (igaz érték)

Egyszerűen eldönthetjük, hogy egy elem atom vagy lista. Ha egy elem zárójelben van, akkor lista, ha nincs zárójelben, akkor atom.

A listák kerek zárójelek között elhelyezett, szóközökkel elválasztott atomok listája. A listák egymásba ágyazhatóak és az egymásba ágyazás szintjeinek nincs korlátja. A listák lehetnek adatlisták, 2D vagy 3D pontlisták, függvénylisták.

Példa listára: (11 (2 3) (4 (5 6)))

Listák kiértékelése esetén első lépésben az AutoLISP a lista első elemét elemzi. Minden kiértékelendő lista első elemének egy függvénynek kell lennie, ha ez nem így van, hibaüzenetet küld. A listában a függvényt követő elemek a függvény argumentumai. Második lépésben az AutoLISP a függvényt értékeli ki, mint utasítássorozatok egymásutániságát, harmadik lépésben sorra kiértékeli az argumentumokat, negyedik lépésben, ha már nincs kiértékelendő argumentum, akkor az AutoLISP interpretor befejezi a függvény utasítássorozatát, például egy összeadással a + függvény esetében, és visszafordítja annak értékét. Az AutoLISP interpretor mindig balról jobbra végzi el az értékelést.

5.2. Hozzárendelés

A hozzárendelés az a specifikus AutoLISP folyamat, amely során egy szimbolikus atomhoz (változóhoz) egy értéket kötünk hozzá. Az érték-hozzárendelés ideiglenes, mivel ugyanahhoz a változóhoz adott pillanatban egy más érték köthető hozzá. A változó típusát így nem szükséges meghatározni.

A hozzárendelés a setq függvénnyel végezhető el:

(setq szimb1 kifej1 szimb2 kifej2 … szimbn kifejn),

ahol a szimb1, szimb2, …, szimbn változók, a kifej1, kifej2, …, kifejn pedig atomok vagy listák.

Példa:

(setq szam 78) - a “szam” értéke ”78”
(setq szam “karaktersorozat”) - a “szam” értéke “karaktersorozat”

5.3. Az AutoCAD pontok tárolási formája

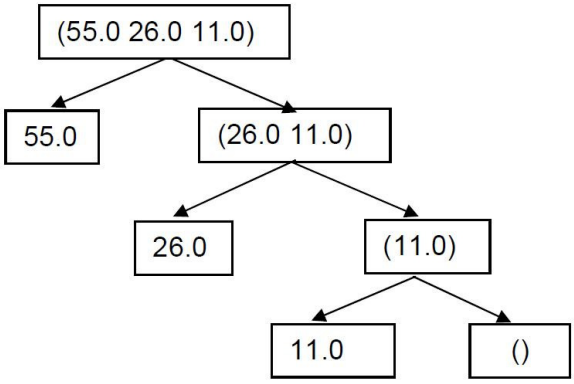
Az AutoCAD program a saját adatbázisában Descartes-féle koordinátarendszert használja a két- és háromdimenziós pontok leírására. Az AutoLISP az AutoCAD pontot egy valós számokból álló listával írja le.

Az AutoCAD-ben egy háromdimenziós pont három, egymástól független számból áll, amelyek az X, Y és Z tengelyre vetített távolságok. Nagyon sokféleképpen adható meg egy pont koordinátája, abszolút, relatív, poláris vagy object snap formában, az adatbázisban azonban ez csupán három szám formájában van tárolva. Ez a pont AutoLISP-ben egy három, valós számból álló lista (például: (1.0 2.0 3.0)), ahol a számok sorra az X, Y és Z koordináták.

5.4. A lista elemeihez való hozzáférés

Egy AutoLISP lista a számítógép memóriájában bináris faként (6. ábra) van jelen. A bináris fa bármely ágazata egy csomópont. A csomópontja két ágra bomlik, egyik gyereke a lista első elemét (lista feje) fogja tartalmazni, másik gyereke lesz a lista fej nélküli része.

Példa:



(6. ábra) AutoLISP lista a számítógép memóriájában

Az AutoLISP a car és cdr függvények segítségével el tudja érni a fa mindkét ágát. A car függvény a lista első elemét, a cdr függvény a lista első elem nélküli részét adja vissza. Mindkét függvény az argumentumban szereplő listát nem bontja le véglegesen, csupán visszatéríti a fent említett részeit.

5.5. AutoLISP és az AutoCAD utasítások

Az AutoLISP egyik óriási előnye, hogy interaktív módon meghívható a kiértékelési modul bármely AutoCAD utasítás közben vagy beírva az AutoLISP utasítást, az rögtön látható az AutoCAD programban. Az AutoLISP-en belül lehetőségünk van bármelyik AutoCAD utasítást használni. Ezt a command függvény segítségével érhetjük el. A command függvény argumentumai AutoCAD utasítások, amelyek karaktersorok és változókként vannak megadva. Példa:

```
Command: (command “line” “1, 1” “5, 5” “”)
Command: nil
```

A kiértékelés eredménye egy 1,1 és 5,5 kordinátájú pontok közötti egyenes lesz. A nil-el jeleztük a befejezést, a command függvény utolsó argumentuma, a “”, pedig egy ENTER-nek felel meg.

Egy ehhez hasonló példa, egy másik megvalósításban, háromdimenziós pontok segítségével a következőképpen néz ki:

```
Command: (setq pont1 (list 1.0 1.0 1.0) pont2 (list 5.0 5.0 5.0))
Command: (command “line” pont1 pont2 “”)
Command: nil
```

Az AutoLISP-nek léteznek olyan függvényei, amelyek lehetővé teszik az interaktív adatbevitelt. Ekkor szünetel a kiértékelés és a felhasználó adatbevitelére várakozik. Ezek a függvények get előtaggal kezdődnek és utótagként a bekért adat típusával. Ilyenek a getpoint, getreal, getdist, getstring, stb függvények. Példa:

```
Command: (getpoint “Kerem az elso pontot:”)
Kerem az elso pontot:1,1
(1.0 1.0 0.0)
```

5.6. Saját AutoLISP függvények létrehozása

A létező függvények mellett lehetőség van a felhasználó által definiált függvények elkészítésére, amelyben használhatók a már létező belső vagy a felhasználó által már definiált függvények. Mindezt a defun függvény segítségével valósíthatjuk meg, amelynek szintaxisa a következő:

(defun <szimb> (<arg lista> / <helyi valtozok> <kifejezesek…>,

ahol a <szimb> helyen kell megadni a definiálni kívánt függvény nevét, az <arg lista> helyén az argumentum listát kell megadni, majd opcionálisan egy törtjel után a függvényt egy vagy több lokális szimbólum követheti. Ha a függvénynek nem adunk meg argumentumot, sem pedig lokális szimbólumot, akkor a függvény nevét egy üres zárójelpárnak kell követnie. A defun függvény az általa definiált függvény nevével tér vissza. Példák defun függvényre:

```
(defun fug (x y)…) két argumentuma van
(defun fug (/ a b)…) két lokális szimbóluma van
(defun fug (x / ideigl)…) egy argumentuma és egy lokális szimbóluma van
(defun fug ()…) sem argumentuma, sem lokális szimbóluma nincs
```

Beépített függvények vagy szimbólumok nevét soha ne használjuk saját függvény nevének definiálására, mivel ebben az esetben a beépített függvény többé nem lesz elérhető.

Egy speciális esete a defun függvénynek az, amikor a létrehozott függvényt használni lehet, mint bármilyen AutoCAD utasítást zárójel használata nélkül. Ekkor a definiálás során a függvény neve elé egy "c:" jelet teszünk. Ebben az esetben a létrehozandó függvény nem használhat argumentumokat. Példa:

```
(defun c:bre ()
  (setq bp (getpoint „Valassza ki a torespontot !”))
  (command “Break” bp „@”)
)
Command:bre
Valassza ki a torespontot!
Command:nil
```

5.7. Listakezelő függvények

A listakezelő függvények ismerete elengedhetetlen ahhoz, hogy az AutoLISP-el dolgozzunk. Ezek közül néhány alapvető függvény, amelyre érdemes kitérni a length, nth, apply és mapcar.

A length függvény egy lista elemeinek a számát egészként adja vissza. Szintaxisa: (length <lista>).

Példa:

```
Command:(setq lista1 (list 1 jj lp “piros” 0.1) - lista létrehozása
Command:(length lista1)                          - lista elemszámának lekérdezése
6                                                - eredmény
```

Az nth függvény kiértékelésként egy lista n-dik elemét adja vissza. Szintaxisa: (nth <arg> <lista>), ahol az arg egy egész szám 0 és n-1 között, ha a lista hossza n.

Példa:

```
(setq lista2 (list “ttt” 78 56 8.2))
(nth 0 lista2)
“ttt”
(nth 2 lista2)
56
```

Az apply függvény egy argumentumlistát ad át egy függvénynek. Szintaxisa: (apply <fuggveny> <lista>). Példa:

```
(apply ‘+ ‘(1 2 3))
9
```

A mapcar függvény a fv kiértékelésének eredményével tér vissza oly módon, hogy a listák elemeit a lista1-től a listan-ig egyenként a függvény argumentumaiként kezeli. Szintaxisa: (mapcar <fv> <lista1>... <listan>). A listák számának egyeznie kell a fv által igényelt argumentumok számával. Példa:

```
(setq a 1 b 2 c 30)
(mapcar '1+ (list a b c))
(2 3 31)
```

5.8. Elágazások

Az AutoLISP-ben a programelágazások kezelésére az if és a cond függvények szolgálnak. Az if függvény szintaxisa:
(if <feltetel-kif> [akkor-kif] [egyebkent-kif])

Az if függvény feltételesen értékeli ki a kifejezéseket. Ha a feltetel-kif nem nil, akkor az akkor-kif kifejezés kerül kiértékelésre, ha viszont nil, akkor az egyebkent-kif kifejezés. Az egyebkent-kif argumentum el is hagyható, ekkor, ha a feltetel-kif értéke nil, az if függvény nil értékkel tér vissza. Példa:

```
(if (= 77 3) "IGEN!!" "nem.") eredménye "nem."
(if (= 6 (+ 4 2)) "IGEN!:") eredménye "IGEN”
(if (= 1 (+ 9 4)) "IGEN:!") eredménye nil
```

Mivel az if függvénynek csupán három argumentuma lehet, több argumentum kiértékeléséhez használhatjuk a progn függvényt, amely lehetővé teszi az egyes logikai elágazásokban akármilyen nagyszámú kifejezés egy argumentumba gyűjtését. Példa:

```
(if (> 3 2)
(progn
(setq wer 12)
(command „line” pt1 pt2)
(command „zoom” „e” „”)
)
(setq wer 789)
)
```

A cond függvény az AutoLISP elsődleges feltételes függvénye. Szintaxisa: (cond (<feltetel1> <eredmeny1>...) (<feltetel2> <eredmeny2>...)...)

Ez a függvény tetszőleges számú listát fogad el argumentumként. Minden allista első elemét kiértékeli egész addig, amíg valamelyik értéke nem különbözik nil-től. Ha van ilyen, akkor ennek az allistának a feltételt követő kifejezéseit kiértékeli és az utolsó kifejezés értékével tér vissza. Amennyiben az allistából hiányoznak az eredmény kifejezések, azaz csak a feltétel kifejezést tartalmazza a lista, akkor ennek a kiértékelésével tér vissza. Tulajdonképpen a cond függvény esetszétválasztásos (case típusú) függvénynek felel meg. Példa:

```
(cond ((= s „I”) 1)
((= s "i") 1)
((= s "N") 0)
((= s "n") 0)
(T nil)
)
```

5.9. Ciklusszervezés AutoLISP-ben

Az AutoLISP ciklusfüggvényei a repeat, while és a foreach. A három függvény között formailag és tartalmilag lényeges különbség van.

A legegyszerűbb ciklusfüggvény a repeat. Szintaxisa:

(repeat <hany> <kifejezesek…>)

A függvény a kifejezéssorokat annyiszor ismétli, amennyi a hany argumentum értéke.

A foreach segítségével egy lista minden elemét kiértékelhetjük egy kifejezésre. A foreach függvény szintaxisa:

(foreach <nev> <lista> <kifejezes…>)

Működése: A lista minden elemét hozzárendeli a nev-hez és minden elemére kiértékeli az összes kifejezést. A foreach függvény az utolsó kifejezés kiértékelésének eredményével tér vissza. Például:

```
(foreach lovak '(“EZ” “EGY” “PELDA”) (print lovak))
“EZ”
“EGY”
“PELDA” “PELDA”
```

A legkomplexebb ciklusfüggvény a leggyakrabban használt while. Szintaxisa:

(while <feltetel> < kifejezesek … >)

A while ciklusfüggvény kiértékeli a feltetel kifejezést. Ha ez nem nil, akkor kiértékeli a kifejezesek-et, majd újból kiértékeli a feltetel kifejezést. Ezt addig folytatja, amíg a feltetel kifejezés értéke nem nil. A while függvény az utolsó kifejezésnek a legutolsó értékével tér vissza. Példa:

```
(setq x 1)
(while (<= x 5)
(sajatfuggv x)
(setq x (1+ x))
)
```

5.10. Rajzelemek módosítása az asszociációs listák segítségével

Egy AutoLISP függvény a subst használatával, az AutoCAD rajzelemeket meg tudja változtatni. A subst segítségével az asszociációs listatípust megváltoztathatjuk. A függvény szintaxisa:

(subst <ujelem> <regielem> <lista>)

Ez a függvény eredményül a lista egy olyan másolatával tér vissza, ahol a listában szereplő régi tételek (regielem) minden előfordulását lecseréli az új tételre (ujelem). A subst a változatlan formájú listával tér vissza, ha a listában nem talál regielem-et.

```
(setq lista1 '(a b (c d) b))
(subst 'XX 'b minta)              - eredménye ( A XX ( C D ) XX )
(subst 'XX 'z minta)              - eredménye ( A B ( C D ) B )
(subst 'XX '(c d) minta)          - eredménye ( A B XX B )
(subst '(XX rr) '(c d) minta)     - eredménye ( A B ( XX RR ) B )
(subst '(XX rr) 'z minta)         - eredménye ( A B ( C D ) B )
```

5.11. Függvények ábrázolása AutoLISP segítségével

Az AutoLisp segítségével 2D vagy 3D függvényeket ábrázolhatunk. A megjelenítés egyik lehetséges megoldása a line használata. Ez azt jelenti, hogy egy görbét egyenes szakaszokkal közelítünk meg, amelyet tulajdonképpen bármilyen pontossággal elvégezhetünk, hiszen mint tudjuk, minél rövidebbek a szakaszok, annál pontosabb lesz a görbe. Műszaki szempontból hihetetlen pontosságú finomítást érhetünk el, akár 10^{-15} mm-es egyenes szakaszokkal is megközelíthetünk egy görbét.

A line művelet helyett célszerűbb a polyline-t használni, ha az új rajzelemek további felhasználhatóságát tekintjük, mivel ezt a rajzelemet további műveletekben, akár bonyolult felületek generálásánál is könnyen használhatjuk.

Ez a folyamat egy példán keresztül könnyen átlátható (lásd Függelék).

Adott egy másodfokú függvény: $y = ax^2 + bx + c$

Egy AutoLISP programot kell írnunk, amely kiszámolja az $ax^2 + bx + c = 0$ egyenlet gyökeit, ha valósak, ha komplexek, akkor kiírja azt, majd ábrázoljuk a függvényt úgy, hogy a szélsőértéke látható legyen.

Első lépésben bekérjük az a, b és c számokat egy adatok nevű függvény segítségével:

```
(defun adatok ()
(getreal "Kerem az a egyutthatot:")
…)
```

Miután beolvastuk az adatokat, készíthetünk egy másik függvényt, amely kiszámolja az egyenlet gyökeit, ha ezek valósak. Ez legyen pl. a gyokok függvény:

```
(defun gyokok ()
(setq delta (… itt kiszámoljuk a b2 – 4ac -t)
```

Ez után a delta függvény segítségével eldöntjük, hogy milyen típusú gyökei vannak az egyenletnek, és ha valósak, akkor kiszámoltatjuk és kiíratjuk őket a képernyőre.

```
(cond
((> delta 0)
(… x1, x2 … =)
(print x1)
(print x2))
((= delta 0)
(x1 = x2 = …)
(print x1=x2=…))
((< delta 0)
(print "Komplex gyokok!"))
```

Majd a szelsoertek függvénnyel kiszámoltatjuk a szélsőértéket:

```
(setq szelsoertek (- (/ b (* 2 a))))
)
```

A harmadik lépésben egy olyan függvényt készítünk, amely kirajzolja az ábrázolni kívánt függvényt bizonyos határok között. A határokat úgy kell megválasztanunk, hogy a függvény csúcsa, ahol a szélsőértéke is található, valahol középen legyen. A határok az xbal és xjobb értékei lesznek: xbal = szelsoertek – (hossz/2), illetve xjobb = szelsoertek + (hossz/2)

Szükségünk van továbbá egy lépéstávra, lepes, amely alapján a függvény pontjait állapítjuk meg.

A függvényünk ábrázolását polyline segítségével akarjuk megvalósítani, így szükségünk lesz az egymás utáni pontok kiszámítására. Ezt egy while ciklus segítségével valósítjuk meg, amely az egyik határtól (xbal) a másik határig (xjobb) működik az x koordináta segítségével és lepes lépésközzel haladva. A ciklus feltétele tehát x < xjobb lesz. A ciklus minden lépése során behelyettesítjük x értékét a függvénybe, így megkapjuk y-t és ezzel a függvény egy pontját. Az így kapott pontokat tárolnunk kell egy listában. Ahhoz, hogy a polyline rajzolásához a listát használni tudjuk, a következő formájúnak kell lennie:
((x1 y1) (x2 y2) … (xk yk) … (xn yn))

Ha a listát megépítettük, akkor ki kell rajzolnunk a megoldást. Ezt a következőképpen oldhatjuk meg:

```
(defun rajzolas ()
(setq xbal…)
(setq xjobb…)
(setq lepes (getreal “Kerem a lepest:”))
(setq x xbal)
(while (< x xjobb)
(setq y =…)
(setq pontlista (cons (list x y) lista))
(setq x (+ x lepes))
)
(command “pline” (foreach pont pontlista (command
pont)))
(setq pontlista (reverse pontlista)))
```

A pontlista nevű változóban tárolt pontok alapján egy polyline rajzolódik ki, amely éppen az adott függvény ábrája. Végül, utolsó lépésként, hogy ne kelljen egyenként futtatni ezeket a függvényeket, összefoglalhatjuk egy új definiált függvényben.

```
(defun fuggvenyabrazolas (/ xbal xjobb szelsoertek
pontlista)
(adatok)
(gyokok)
(rajzolas))
```

Ha 3D függvények ábrázolását akarjuk megvalósítani, hasonló módon járhatunk el, csupán a polyline helyett a 3dpoly AutoCAD parancsot alkalmazzuk.

6. 3ds Max

Az Autodesk 3ds Max, korábbi nevén 3D Studio Max (röviden 3ds Max), egy 3 dimenziós modellező és aminációs program, amelyet az Autodesk Media and Entertainment fejlesztett ki.

A 3ds Max erőteljes, integrált 3D modellezést, animációt, renderelést és kompozitálást tesz lehetővé, amelyekkel a művészek és tervezők gyorsabban készíthetik el produkcióikat.

6.1. Története

Az eredeti 3D Studio terméket a Yost Group készítette DOS-os platformra (1988) és az Autodesk publikálta (1990) [11]. A 3D Studio Realease 4 után a terméket újraírták Windows NT platformra és átnevezték 3D Studio Max-nak. Ezt a verziót is a Yost Group készítette és a Kinetix publikálta, ami ugyanakkor az Autodesk Media and Entertainment (média és szórakoztatás) divíziója volt. Az Autodesk megvásárolta a terméket a 3D Studio Max második verziójának kiadásánál és átvette a termék fejlesztését. Azóta számos verzió jelent meg, majd a nyolcadik kiadásnák a termék nevéhez hozzáfűzték ismét az “Autodesk” logót.

6.2. Parametrikus modellezés és 3ds Max

A 3ds Max egyik különlegessége a parametrikus modellezés, mely azt jelenti, hogy a tárgyakat és a rajtuk végzett műveleteket parametrikusan, matematikai formában írja le [12]. A program másik érdekessége, hogy ötvözte a poligonális és a spline alapú modellezést.

Poligonális modellezés során a program a tárgyakat pontokból és azok között feszülő felületekből építi fel, ezek koordinátáit és paramétereit tárolja.

Spline alapú modellezés során a tárgyakat vektorokból építjük fel.

Visszatérve a parametrikus modellezéshez, ahol minden tárgy és a rajta végzett művelet matematikai formában kerül rögzítésre. Vagyis például egy henger létrehozásánál a program nem a felületét alkotó pontokat írja le, hanem csak a henger középpontját, alapkörének sugarát és a magasságát. Megjelenítéskor a program ezekből számítja ki a poligonális tárgyat. Ezzel a módszerrel könnyen megváltoztatható bármely tárgy bármely paramétere bármikor, anélkül, hogy

a tárgy bármilyen torzulást szenvedne. A parametrikus modellezés legnagyobb előnye, hogy rendkívül egyszerűvé teszi az animációkészítést, hiszen az alkalmazott módosítókat és értékeiket is matematikailag tárolja, tehát ezek bármikor utólag is módosíthatók, törölhetők.

Modifier Stack

A Modifier Stack (módosítóverem) az alkalmazott módosítók előzménylistája [1]. Ha utólag szeretnénk valamilyen paramétert megváltoztatni, csak jelöljük ki a tárgyat és a Módosítások panelen a Stack-ben megjelenő módosítók közül csak ki kell választanunk a kívánt módosítót és már meg is tudjuk változtatni az értékeket. A Stack-ben tárolt módosítók alulról felfelé követik egymást, azaz alul van maga a tárgy, s felette a használat sorrendjében az egyes módosítók.

6.3. A 3ds Max főbb sajátosságai

MAXScript

A MAXScript a 3ds Max programok beépített szkriptnyelve, a program teljes funkcionalitású vezérlőnyelve, amely a következőkre használható:

- A program használatának főbb vonatkozásait, mint a modellezés, animáció, anyagok, renderelés szkriptelésére.
- A program parancssoros ablakból történő interaktív irányítására.
- A felhasználói felületet kiterjesztésére vagy lecserélésére.
- Szkriptelt plug-in-t írni egyéni geometriához, módosítóhoz, rendereffekthez, stb.
- Szkriptelt saját import/export eszköz írására.
- Kötegelt feldolgozású eszközök létrehozására, mint amilyen a kötegelt renderelés, stb.
- MAXScript parancsokként történő rögzítésére, amit tevékenykedünk a programon belül.
- A szkriptek eltárolására a jelenetfájlon belül, így azok minden alkalommal lefuthatnak, amikor szükség van rájuk (például renderlés előtt, objektumok kijelölése után, stb.).

A MAXScript nyelv speciálisan a 3ds Max programhoz kifejlesztett nyelv, egyszerű szintaxissal, hogy a nem programozó képzettségű grafikusok is könnyen boldogulhassanak vele [2].

A MAXScript programozási nyelvet a fejezet végén részletesebben elemzem.

Character Studio

A Character Studio egy beépített kiegészítés, amely a 3D Studio Max 4-es verziójáig egy különálló mudul volt. A Character Studio segíti a felhasználókat a csontvázalapú animációban. A rendszer úgy működik, hogy egy előre elkészített (beépített) csontvázat ("Biped" rig) hozzárendelünk az animálni kívánt karakterünkhöz. A Character Studio motion capture (az a folyamat, amikor bizonyos mozgásokat rögzítenek, majd ezt a mozgást egy digitális modellhez rendelik) és görbemódosítási eszközei ideálissá teszik karakter animációkhoz. A "Biped" objektumoknak még sok más hasznos képességei vannak, amelyek automatizálták a mozgási (járási) ciklusok, mozgási útvonalak és a másodlagos mozgások (objektumok visszahatása az elsődleges karakter mozgására) létrehozását.

Scene Explorer

A Scene Explorer a Windows böngészőhöz hasonló böngésző a 3ds Max-hoz, mely segítségével könnyen kezelhetővé és átláthatóvá tehetjük összetett, sok objektumot tartalmazó jeleneteinket. A Scene Explorer lehetővé teszi bármilyen típusú objektum vagy tulajdonság rendezését, keresését vagy megszűrését egy jelenetben. A 3ds Max 2008-as verzióhoz adták hozzá és képes volt az objektumok meta adatait is megjeleníteni, kezelni.

DWG Import

3ds Max lehetővé teszi DWG fájlok összekapcsolását és importálását. A 3ds Max 2008-as verzió fejlesztéseinek köszönhetően nagyobb jeleneteket is tudunk importálni, több objektummal.

Texture Assignment/Editing

A 3ds Max-ban az anyagok határozzák meg a felületek tulajdonságait - színüket, csillogásukat, textúrájukat, fényvisszaverő - képességüket, és így tovább. A 3ds Max számtalan lehetőséget és megoldást kínál a textúrák használatához. Saját textúrákat hozhatunk létre, könnyen módosíthatjuk őket a Material Editor segítségével. Textúrák "drag-and-drop" szerű hozzárendelése az objektumokhoz is lehetséges.

General Keyframing

Ha az animációs technikákat próbálnánk meg összefoglalni, a következő válfajokat különíthetjük el:

- Kulcs animáció
- Programvezérelt animáció
- Összetett animáció
- Motion capture

A leggyakrabban használt technika a „keyframe” (kulcs) animáció. Ekkor a mozgást kulcspozíciók megadásával határozzuk meg. Ezen pozíciók között a program számítja ki, vagyis interpolálja az animációs görbét. A felhasználónak mindig lehetősége van az interpoláció paraméterezésére, így lehetőségünk van a mozgás ritmusát, dinamikáját és puhaságát befolyásolni.

A 3ds Max-ban a kulcspoziciókat kétféle képpen adhatjuk meg, kézzel és automatikusan. Ezek mellett lehetőségünk van a kulcsokat kivágni, másolni és beilleszteni, így nagyon egyszerűen tudunk animálni. Az animációs pályagörbéket a Nézetablakból is megtekinthetjük és módosíthatjuk.

Kényszerített Animációk

A 3ds Max-ban minden objektumhoz és animált paraméterhez az animálható viselkedésüket leíró animációvezérlők vagy megszorítások vannak rendelve. Az animációvezérlők és megszorítások sokban hasonlítanak egymáshoz. A fő különbség közöttük, hogy a vezérlők közvetlenül hatnak az objektumok viselkedésére, míg a megszorításoknak valamilyen külső információra, például a jelenetben szereplő alakzatra vagy egyéb objektumra van szükségük.

A kényszerített animáció fontos szerepet tölt be az animálásban. Az animált karaktereket pl. arra tudjuk kényszeríteni, hogy egy előre megrajzolt pályán, görbén mozogjanak, melyik irányba nézzenek mozgás közben a megfelelő időben, stb.

Skinning

A csontváz és a figura teste közötti összeköttetést nekünk kell meghatároznunk. Mivel a valóságban a bőr, azaz a figura felszíne nem közvetlenül a csont mozgását követi (gondoljunk a bonyolult ín- és izomrendszerre), nekünk kell a csontokhoz a felszín egyes darabjait hozzárendelni. Ez a hozzárendelés az ún. skinning.

A munkának ez a fázisa gyakran nehezebb, mint maga az animálás.

A rossz skinning eredménye az, hogy a hajlatoknál „gyűrődések" jönnek létre, azaz pl. a felkar mozgásánál a mellkas egy darabja is elmozdul. A 3ds Max súlyokkal dolgozik, az egyes csontokat kiválasztva egy ecsettel adjuk meg azt, hogy a felületre mennyire erősen hasson a csont elmozdulása.

Csontvázak és Inverz Kinematika (IK)

A megmozgatni (animálni) kívánt modellhez a topológiája alapján egy csont/ízületrendszert rendelünk - az úgynevezett „rigging" eljárás során - a virtuális marionett különböző irányítókat kap, azt az animátor így manipulálni tudja.

A számítógépes karakteranimáció leginkább egy merev darabokból álló bábu animálására hasonlít. Csontokat és őket összekötő ízületeket hozunk létre, amelyek a valódi csontvázhoz hasonlóan mozgatják a felületeket.

A 3ds Max-ban két lehetőség van a csontozatok, azaz a figurák mozgatására:

- Forward kinematika
- Inverz kinematika

Forward kinematika

Lehetőségünk van a figurák „hagyományos" mozgatására. Először a karok, lábak felső csuklóit állítjuk be és egyesével haladunk a csontrendszer utolsó csuklói, az ujjak felé.

A gyakorlott, hagyományos technikákon nevelkedett animátorok szinte kizárólag ezt a megközelítést alkalmazzák, hiszen így a mozdulatok legapróbb részleteit is kezükben tarthatják.

Inverz kinematika

Ez a technika sokkal kényelmesebb és hatékonyabb, mint az előbb említett. Az animátornak nincsen más feladata, mint a csontrendszer utolsó csontját mozgatni, a többiről a számítógép gondoskodik.

A gyakorlatban ez az jelenti, hogy elég a figurának a kézfejét mozgatni, a könyök és a váll mozgását a program kikövetkezteti. Komplex mozgásokat szinte csak ezzel a technikával lehet elkészíteni. Egy sétáló figura animálásához elegendő a láb- és a kézfejeket mozgatni.

Integrált Cloth Solver

A haj szimuláció után a ruha szimuláció az egyik számítástechnikai szemmel legnehezebb terület. A 3ds Max rendelkezik integrált ruha-szimulációs motorral, ami lehetővé teszi, hogy majdnem minden 3D objektumot ruhává alakíthassunk vagy újat alkossunk.

A 3ds Max Cloth kiegészítésben, az első lépés a kiterített szabásminta készítése, amiben segít a Garment Maker (GM) módosító. A GM az egyszerű spline szabásmintát helyettesíti a ruha szimulációhoz szükséges poligon hálóval, illetve itt definiálhatjuk az összevarrt éleket. A Cloth módosító a lelke a rendszernek, itt adhatjuk meg a ruha objektumot és választhatjuk ki az ütközéshez használt geometriát. A ruha tulajdonképpen egy külső és belső erők által torzított poligon felület. Ezek az erők pedig a gravitáció, az anyaghajlítási, nyúlási belső erői és a karakter által keltett erők. A karakter esetén a programnak a ruha egyes pontjainak és a karakter felületének ütközését kell vizsgálnia, ezért fontos, hogy szinkronban legyen ezen objektumok felbontása. A rendszer a geometria deformációja mellett különböző anyagjellemzőket is kezel, például gyűrődési és élképződéseket, amelyek döntően befolyásolják a fizikai viselkedést.

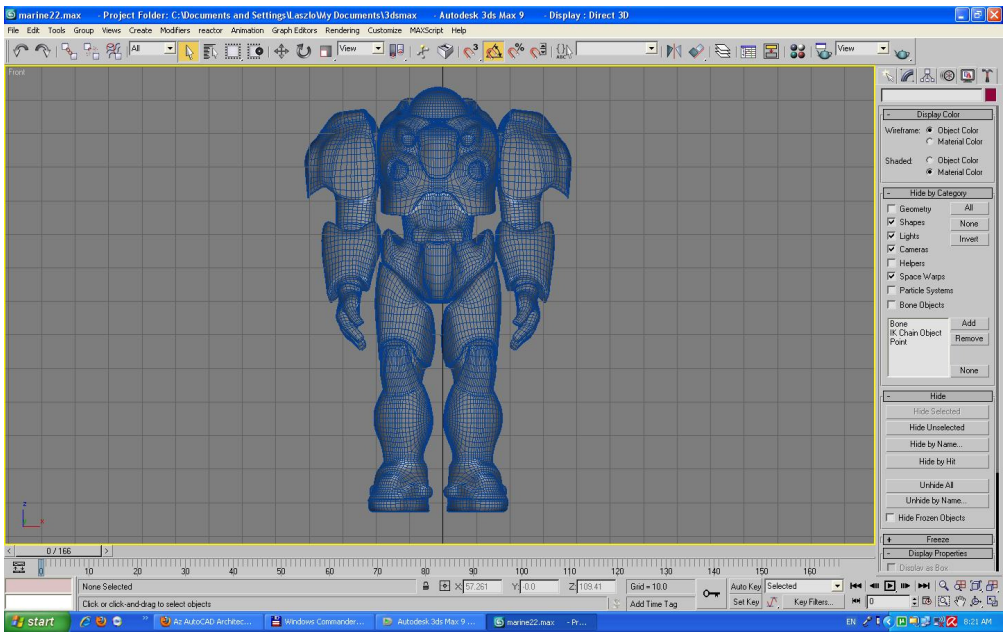
Integráció az Autodesk Vault-al

Az Autodesk Vault beépülő modul, amely része a 3ds Max programnak, egyetlen helyen gyűjti össze a 3ds Max eszközöket, lehetővé téve a fájlok automatikus követését, valamint a folyamatban lévő munka kezelését [13].

Egyszerűen és biztonságosan oszthat meg, kereshet és használhat fel újra 3ds Max-eszközöket nagyléptékű gyártási vagy megjelenítési környezetekben.

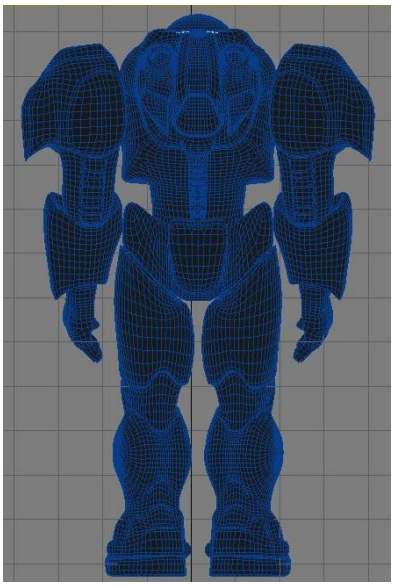
6.4. Egy 3ds Max példa

Ahhoz, hogy jobb képet tudjunk alkotni a 3ds Max-ról, úgy gondoltam bemutatom az egyik modellemet, mégpedig a legelsőt, amit a 3ds Max-al készítettem. Miért is kezdtem el modellezni és akartam megtanulni a 3ds Max használatát? Azt hiszem, magával ragadott az a hihetetlen látvány, amit a modellezés nyújtani tud, ami az évek elteltével egyre valósághűbb lesz és az a tulajdonsága, hogy milyen sokoldalú és hány területen használják (tervezés, film, játékok). Az alábbi képek (7.-9. ábra) egy kitalált jövőbeli űröltözék drótvázas modelljét ábrázolja.

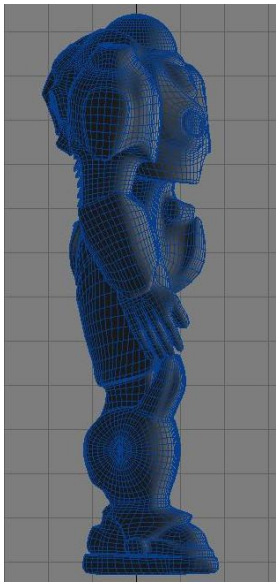


(7. ábra) 3ds Max modell, előnézet

A rajzot, amiről a modellt készítettem, az interneten böngészve találtam. Kezdőként és lelkesen nagy fába vágtam a fejszém első projektem elkészítéséhez, mivel nem egyszerű, hanem humanoid, ember formájú alakzatot választottam megvalósításra. Ahhoz, hogy egy modellező egy ilyen modellt el tudjon készíteni, rendelkeznie kell némi embertani ismeretekkel is és az arányokat is be kell tudni tartania. Úgy gondolom, nekem ezt végül többé-kevésbé sikerül is megvalósítani.



(8. ábra) 3ds Max modell, hátul nézet



(9. ábra) 3ds Max modell, oldal nézet

Látható, hogy a modell nagy részletességgel lett kidolgozva. A megvalósított modell több mint 27000 pontot és 56000 poligont tartalmaz. Nagy részletességgel kidolgozott modellekre van szükségünk például animációs film készítésénél, prezentációknál, gyártási modellek elkészítésénél, viszont ez nagymértékben megnöveli a rendereléshez szükséges időt és nagy gépigénye van. Éppen ezért más területeken, mint például számítógépes játékok modelleinél, minél alacsonyabb poligonszámú modellek készítésére törekszenek, a valósághű hatást más eszközök segítségével próbálják elérni, mint például részletesen kidolgozott textúrák vagy fényhatások.

6.5. 3ds Max vagy 3ds Max Design?

A két szoftver változat között, néhány eltérést leszámítva nincs különbség [14]. Egyedül a 3ds Max rendelkezik fejlesztőkörnyezettel (SDK). A 3ds Max Design szoftverben kézhez kapunk egy Lighting Analysis Assistant nevű segédeszközt, valamint a súgó teljesen építész/mérnöki példákra lett átdolgozva. Ezen kívül a két szoftver teljesen azonos, tökéletesen futnak rajta a plugin-modulok, teljesen kompatibilisek egymással. Lényeges szempont, hogy egyazon számítógépre nem telepíthető fel mind a két változat. Ezen információk birtokában könnyű a döntés: ha film, reklám, játékfejlesztés területen dolgozunk, vagy 3ds Max szoftverhez fejleszteni

szeretnénk plugin modulokat (nem Maxscript-ben), akkor 3ds Max, ha műszaki feladatokra használnánk a szoftvert, akkor 3ds Max Design-t fogjuk választani. Valószínűleg a két szoftver a jövőben sem fog külön technológiára épülni, csak a különböző célú kiegészítő modulokban térnek majd el egymástól.

6.5.1. Lighting Analysis Assistant (LAA)

Kizárólag 3ds Max Design 2009 funkció. A mental ray képkiszámítás alatt működő eszköz, integrált egyesített paneleken a pontos fényméréshez a 3ds Max Design jelenetet elemzi. A panel a jelenetben kiemeli félkövér karakterekkel a hibás beállításokat, ellenőrzi, hogy megfelelő-e a fényvisszaverődés számítás, ellenőrzi, hogy csak fotometrikus fényforrásokat és fizikai anyagokat használunk-e, továbbá javaslatot tesz a hibák kezelésére.

6.5.2. Érvek az Autodesk 3ds Max Design mellett

- **Kiváló adatkapcsolatok**

  Szinte mindegyik Autodesk programhoz (AutoCAD, AutoCAD Civil 3D, Revit Architecture és Autodesk Inventor) képes kapcsolódni. Támogatja a DWG és FBX fájlformátumokat, ezzel is csökkentve az utómunkát. Revit és AutoCAD jelenetekből átveszi a fény és anyag beállításokat is!

- **Fejlett modellezési eszközök**

  A közismerten erős poligonmodellezés egyre fejletebb eszközökkel bővül. Az új Graphite eszköztár megkönnyíti a szabad formák modellezését és felanyagozását.

- **Rugalmas látványtervezés**

  Quicksilver hardveresen gyorsított képszámító segít a magas minőségű látványtervek szükséges előnézeteinek elkészítésében. A képszámítási folyamat hálózatba kötött számítógépek között korlátlanul szétoszthető.

- **Prezentációs eszközök**

  Megjeleníthetjük a terveink mögötti tartalmat valósághű látványelemekkel, úgy, mint füst, tűz, víz vagy drapériák.

- **Azonnali termelékenység**

  Telepítés után a szükséges eszközök azonnal rendelkezésre állnak, nem kell kiegészítőket és külső eszközöket beszerezni.

- **Megosztott anyagtárak**

  Az Autodesk mérnöki és látványtervező szoftverei közös anyagtárat használnak, így adatcserekor nem kell az anyagokat újra és újra beállítani.

- **Megvilágítás elemzés**

  Exposure megvilágítás elemzővel jól mérhetjük és szemléltethetjük a természetes és mesterséges megvilágítás hatását az épülettervezés kezdeti szakaszában is.

Összegzés

Nem kétséges, hogy a két termék közel azonos. Ha az újdonságokat vesszük figyelembe egyértelmű, hogy a műszaki látványtervezés jelentősen profitál a fejlesztésekből. Aki Autodesk VIZ szoftverrel rendelkezik, lehetősége van kedvezményesen frissíteni a 3ds Max Design termékre és az újdonságok mellett, az Autodesk VIZ-ből eddig hiányzó részecskerendszert, valós idejű fizikai szimulációs modult, alobjektum animációt és karakter animációt is kap az új szoftverhez.

6.6. Az Autocad és 3D Studio Max kapcsolata

Ahhoz, hogy minél gyorsabban és hatékonyabban elvégezhessük munkánkat, nem hagyhatjuk figyelmen kívül a két program együttműködésével járó előnyöket a tervezés, modellezés folyamatában. Bizonyos dolgokat könnyebb elvégezni, megtervezni az egyik program segítségével és folytatni a munkát a másikkal, mint például hihetetlenül pontos alaprajzokat készíthetünk Autocad-ben, amit aztán 3D-ben a 3ds Max segítségével megrajzolunk. Lehetőségünk van az egyik programban készült modellek, adatok, tervek másikba történő átvitelére.

Ezt a kapcsolatot két kiválasztott Autodesk programon szeretném szemléltetni: Autocad Architecture 2011 és 3ds Max 2011.

6.6.1. Az Autocad Architecture 2011 és a 3ds Max 2011 kapcsolata

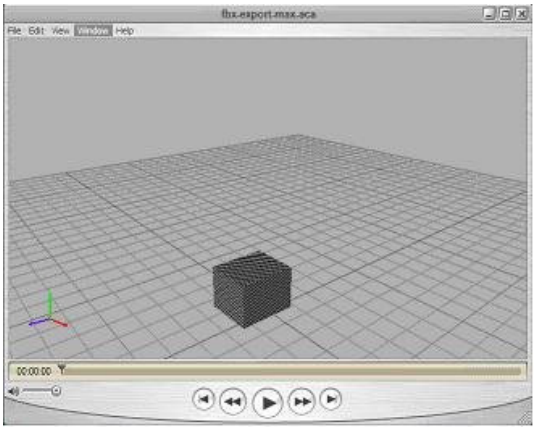
Egyre több építész használja az Autocad Architecture programot. Az Autocad Architecture (röviden ACA) az Autocad szoftver építészek számára készített változata. A kifejezetten az építészek számára készült eszközökkel hatékonyabban hozhatók létre épülettervek és dokumentációk. Automatizálhatók a fárasztó rajzolási feladatok, és csökkenthető a hibák száma.

Korábban is lehetett kísérletezni az ACA és a 3ds Max között, hogy a legjobb eredményt hozzuk ki a modellekből, de most jött el az az idő, amikor minden feltétel adott ahhoz, hogy az ACA programban elkészített rajzunkat a legegyszerűbben vigyük át a 3DS Max programba és ott a legtöbbet hozzuk ki belőle, anélkül, hogy különböző trükkökre lenne szükségünk.

Alapesetben a 3DS Max program az építész rajzok átvitelére a File Link Manager használatát támogatta és támogatja most is. Viszont egy új lehetőség is képbe került, hisz az ACA2011-hez is elérhetővé vált az FBX export, amit a 3ds Max már korábban is képes volt importálni, de az ACA-ba csak most került bele.

6.5.2. FBX

Maga az FBX formátum az Autodesk saját fejlesztésű adatátviteli/tartalom átviteli technológiája, ami alapvetően azért lett létrehozva, hogy a különböző stúdiók, akik különböző Autodesk szoftvereket használnak, de adott esetben egy munkán dolgoznak, azok is könnyedén megoszthassák egymással a digitális tartalmakat.



(9. ábra) FBX Quick Time Plyerben megnyitva

Még egy Plug-in-t is készített az Autodesk, amivel a QuickTime Player segítségével meg tudjuk nézni az FBX kiterjesztésű fájlokat (9. ábra), sőt körbe is tudjuk nézni, és ha az animációt is tartalmaz, akkor azt le is tudjuk vele játszani.

6.5.3. Autodesk Material Library

Az új renderanyag könyvtár gyakorlatilag egy önálló kis alkalmazás, amelyet közösen használnak az Autodesk alkalmazások. Ilyen a Revit, az AutoCAD/ACA, a 3DS MAX, Inventor, Maya stb. Arra szolgál, hogy az egyik alkalmazásban felanyagozott objektum átvíve a másik alkalmazásba ugyanúgy jelenjen meg, ahogyan az a kiinduló alkalmazásban létre lett hozva. A mi esetünkben, ha valamit felanyagozunk az Autocad Architecture-ben és azt beolvassuk a 3ds Max-ba, akkor ne kelljen ismét felanyagozni az objektumokat, hanem ugyanazt lássuk, mint az ACA-ban.

Ehhez az új Anyagkönyvtárhoz kapunk egy megújult Anyagböngésző palettát. Ezen a palettán nem csak az anyagkönyvtárban lévő anyagokat tudjuk böngészni, hanem a rajzunkban lévő összes anyagot is tudjuk kezelni.

Ebben az Anyagkönyvtárban anyagok ezreit találjuk, amelyek kategóriákba vannak rendezve és keresni is tudunk közöttük, nem csak a nevük, hanem akár a jellemzőik alapján is. Újabb könyvtárakat tudunk készíteni, azon belül kategóriákat és természetesen új anyagokat is létrehozhatunk.

6.5.4. File Link Manager

A legjobb végeredményt mégis a File Link Manager nyújtja. A File Link Manager szolgál arra a 3ds Max-ban, hogy az Autocad Architecture-ből rajzokat importáljunk be úgy, hogy a változtatás lehetőségét megtartjuk. Azaz, ha változtatunk az eredeti rajzon, akkor a változást jelzi nekünk a Manager, és lehetőségünk van a frissítésre.

Bár a Manager az FBX kiterjesztésű rajzokat is megnyitja, sajnos csak azokat támogatja, amelyek Revit-tel készültek.

Míg korábban Preset (Készletek) között csak a Revit opció volt elérhető, addig most már a 2011-es 3ds Max-ban megtaláljuk az előre beállított készletek között mind a DWG File Saved from AutoCAD, illetve a DWG File Saved from AutoCAD Architecture opciót is.

Ez után lehetőségünk van rá, hogy átméretezzük már a beillesztés előtt a rajzunkat, ha esetleg olyan valakitől kaptuk, aki nem abban a mértékegységben dolgozik, mint mi.

Amit még megtehetünk a beillesztés előtt, hogy a fóliák között mazsolázva kiválasztjuk azokat, amelyeket tartalmukkal együtt szeretnénk beimportálni a 3ds Max-ba. Azt is megtehetjük, hogy ha előrelátóak voltunk és már az ACA-ban lefagyasztottuk azokat a fóliákat, amelyeket nem akartunk viszontlátni a 3ds Max-ban, hogy egyszerűen a Skip all frozen layers opciót használjuk, azaz a lefagyasztott fóliákat kihagyatjuk az importálás során.

Ha sikerült ezt a néhány kis beállítást megtennünk, akkor nincs más dolgunk, mint hogy az Attach this file (Fájl hozzáadása) gombbal becsatoljuk a rajzunkat a 3ds Max-ba.

Amennyiben olyan anyagokat használtunk, amelyek nem találhatóak meg a 3ds Max elérési útvonalában, akkor ezt rendereléskor jelezni is fogja nekünk, viszont, ha az Autodesk Material Library-ből használjuk az anyagokat, nem találkozunk ezzel a problémával.

7. Bővebben a MaxScript-ről

7.1. A MaxScript nyelvi elemei

Azonosítók

Az azonosítók alfanumerikus karakterek és az aláhúzásjel sorozatai lehetnek vagy akár tetszőleges karaktersorozatot is megadhatunk, ha egyszeres idézőjelek közé van zárva, feltéve, hogy nem tartalmaz ilyen idézőjelet [4].

Kulcsszavai közül néhányat felsorolok: about, and, animate, as, at, by, case, catch, stb.

A következő határolójelek és szimbólumok részei a nyelvnek: (,), +, *, -, /, ^, +=, -=, *=, /=, --, =, ;, <eol>, ,, [,], :, ', &, ., {, }, #, ==, !=, <, >, >=, <=, ?, $, ..., .., \.

Fenntartott globális változók

Ide tartoznak a rendszerszinten definiált változók. Ezeket három csoportra bonthatjuk:

- előredefiniált globális változók (ezek csak olvashatóak, például: true, false,white, undefined)
- a 3ds Max rendszer globális változói (például: frameRate, renderEffects)
- a MAXScript rendszer állapotát tükröző rendszerglobálisok (például: editorFontSize, stackLimit)

A fenntartott globális változókat használhatjuk saját változónevekként is, viszont ez nem ajánlott.

Literálok

A MAXScript literáljait három fő csoportba osztjuk: számliterálok (1.), string-literálok (2.) és speciális lierálok (3.).

1. Számliterálok

A MAXScript kétféle számtípust használ: 32 bites előjeles egészeket, illetve egyszeres pontosságú lebegőpontos számokat. Az ezeket jelölő literálok alakja:

[-]{<digit>}[.{<digit>}[(e|E)[+|-]{<digit>}+]
például: 123, 123.45, -0.00345, 1.0e-6, .1

Írhatunk hexadecimális alakban is számokat:

0x{<hex_digit>}+
például: 0x0E

2. String-literálok

Más nyelvekhez hasonlóan, a string-eket “-jelek közé írjuk, használhatjuk a megszokott escape szekvenciákat is (például \n, \”, \t vagy a \x{d} unicode szekvencia hexadecimális számokkal), illetve a %, * és ? speciális jeleknél az alábbiakat: \%, *, \?.

3. Speciális literálok

Annak köszönhetően, hogy ez egy komplex modellező és animációs program vezérlőnyelve számos speciális literált írhatunk, például olyanokat, amelyekkel időt, térbeli pozíciót írhatunk le vagy hivatkozhatunk hierarchián belüli részekre.

a.) Időliterálok

Az idő tick-ek (ketyegések) többszöröseivel írható le. A 3ds Max-ban egy másodperc 4800 ketyegést jelent, vagyis a legrövidebb ábrázolható idő 1/4800-ad másodperc. A másik fontos elem a másodpercenkénti képkockaszám (frame). A másodpercenkénti képkockaszám állítható, ez alapértelmezés szerint 30 képkocka/másodperc.
Az időliterálokat a következő képpen adhatjuk meg:

{<decimal_number>[m|s|f|t]}+ -- perc/másodperc/képkocka/ketyegés
[-]{<digit>}:{<digit>}[.{<digit>}] -- SMPTE percek: másodpercek. frame-szám
[-]<decimal_number>n -- normalizált idő

b.) Útvonalnév literálok

A 3ds Max jelenetében előforduló objektumokat útvonalnevekkel azonosíthatjuk (például egy vonal, egy gömb geometria, egy fényforrás, stb.). Mivel a nevekben használhatunk speciális karaktereket (?, *), egy útvonalnév több objektumot is azonosíthat. Példák:

```
$box01 -- az az objektum, aminek a neve 'box01'
$torzs/felkar/alkar -- hierarchikus útvonal név
$*box* -- minden olyan objektum, aminek a nevében szerepel a 'box'
$torzs/* -- minden közvetlen gyereke a $torzs-nek
$helpers/d* -- minden 'd'-vel kezdődő nevű segédobjektum
```

c.) 2D és 3D pont literálok

Alakja egész vagy lebegőpontos szám értékű kifejezésekből (expr) épül fel.

```
[<expr>, <expr>] és [<expr>, <expr>, <expr>]
```

d.) Tömbliterálok

A tömbök értékek egy rendezett sorozata. Tömbliterálokat kifejezésekből építhetünk, például:

```
#(1, 2, 3) -- 3 egész számot tartalmazó tömb
 #() -- üres tömb
#(1,"foo",#(1.2, -4,#fred),[1,2,3]) -- számot, string-et, beágyazott tömböt és pontot is tartalmazó tömb
```

Megjegyzések

A megjegyzéseket a /* és */ határolóelemek közé kell írni. Ha csak a kezdő tagot tüntetjük fel, a program a fájl végéig kikomentezettnek tekinti a programszöveget.

7.2. Típusok, Típuskonstrukciók

A MAXScript-ben tulajdonképpen alapvető típuskategóriák nincsenek. Minden osztály, a Value-ból származik, amely a rendszer gyökér osztálya. A Value biztosít olyan metódusokat és operátorokat, amelyek bármely osztály számára elérhetők. A MAXScript-ben számos beépített típus van (Basic Data Values, Time Values, Bitmap Values, Stream Values, stb). A struct definiálásával új típust tudunk megadni.

7.2.1. Struct típuskonstrukciók

A struct típuskonstrukcióval megvalósítható a direktszorzat típus. Legegyszerűbb esetben a struct-definíció csupán a tagok neveinek felsorolásából áll. Itt nem adható meg explicit típus, ahogy a nyelvben máshol sem. Értékadáskor létrejön egy példány, amely mezőit név szerint inicializálhatjuk. Amennyiben egy mezőnek ilyenkor nem adunk értéket, a mezőnek a legelső értékadásig undefined értéke lesz.

```
struct person (name, height, age, sex)
bill = person name:"Bill" height:72 age:34 sex:#male
joe = person name:"Joseph" sex:#male
```

Származtatott metódusok

Minden struct a Value osztályból származik. Egyetlen metódus, a copy() kivételével (amely nem származik a Value osztályból), a print(), format(), classOf(), superClassOf(), isKindOf(), == és != metódusok implicit implementálásra kerülnek a Value osztályban. A copy() metódus egy sekélyszintű másolatot hoz létre az adott struct-ból, így például egy hivatkozott tömböt tartalmazó struct másolata ugyanarra a tömbpéldányra fog a továbbiakban is hivatkozni.

Eseménykezelők

A struct-okhoz két esemény is kötődik. Szintaxisuk: on create do <expression> és on clone do <expression>. Amennyiben új pédányt hozunk létre az adott struct-ból vagy meghívjuk az adott példányra a copy műveletet, a hozzájuk kapcsolt kifejezés végrehajtásra kerül. Az eseménykezelők implicit private-ek.

```
struct foo2
( A = 1,
B = 3,
fn error = throw "ZZZ",
on create do format "Struct Created: %\n" this,
on clone do format " Struct Cloned: %\n" this )
f = foo2 ()
```

Private és public tagok

A MAXScript-ben két láthatósági szint adható meg, a public és private, és ezek is csak az általunk létrehozott struct-ok elemeire. Ha nem adunk meg láthatósági minősítőt, a program alapértelmezés szerint public-ként kezel minden mezőt. Private tagok csak a struct példányon belül érhetők el. Private tag elérésének megkísérlése futási idejű hibát vált ki. A két támogatott eseménykezelő (on create do és on clone do) implicit privátok, viszont elláthatók public minősítéssel.

7.3. Utasítások, vezérlési szerkezetek

7.3.1. Értékadás, üres utasítás

A MAXScript-ben az értékadás **<**cél**> = <**kifejezés**>** alakú, ahol a cél lehet változónév, tulajdonság vagy tömbindex. Tulajdonságok és tömbindexek segítségével elérhetjük összetett értékek komponenseit, mint amilyenek a tömbök vagy háromdimenziós pontok. Példák:

```
x = 2 + 2 * 3
y = [1, 2, 3] + pos
z = #(1, 2, "foo", "bar")
$box01.pos = baseObj.pos + (baseObj.radius*[0,0,1]) -- tulajdonság szerinti elérés
z[2] = d.rotation as eulerangles -- tömbindex szerinti elérés
```

Két változóérték felcserélésére beépített függvényt nyújt a MAXScript swap **<**cél**> <**cél**>** szintaxissal.

A MAXScript nyelvben nincs üres utasítás.

7.3.2. Szekvencia és blokk-utasítás

A programszöveg olyan kifejezésekből áll (legalább egyből), amelyeket sortörés vagy “**;**” jel választ el. Blokk-kifejezéseket kerek zárójelpárak között hozhatunk létre. Sok más nyelvtől eltérően egy blokk-kifejezés lehet operandus, vagyis szerepelhet valamely kifejezés részeként.

Egy blokk-kifejezés ilyenkor visszatér egy értékkel, amely az utolsó blokkbeli kifejezés által meghatározott.

7.3.3. Elágazás

A MAXScript-ben a feltétel szerinti elágazás és összetett elágazás (esetszétválasztás) is létezik. A feltétel szerinti elágaztatásnak két alakja van:

if **<**kifejezés**>** then **<**kifejezés**>** [else **<**kifejezés**>**] vagy

if **<**kifejezés**>** do **<**kifejezés**>**

Mindkét formában az első kifejezések logikai kifejezéseknek vagy összehasonlításoknak kell lenniük, vagyis true vagy false értékre kell kiértékelődniük. Az elágazás maga is egy kifejezés, értéke a feltétel igaz volta esetén az igaz ágban, egyéb esetben az else ágban lévő kifejezés értéke. Amennyiben nincs else ág, vagy az if…do… alakzatot használjuk és a feltétel hamisra értékelődik ki, akkor az if kifejezés undefined értéket kap.

Példa if elágazásra: a = (if d == 0 then 0 else a / d)

Összetett elágazás (esetszétválasztás) esetén egy kifejezés értéke alapján tekintjük az egyes eseteket. Ez a kifejezés lehet számliterál, névliterál, string-literál, változó, vagy blokk-kifejezés. Végrehajtáskor először kiértékelődik a szelektor kifejezés, majd az egyes ágakhoz tartozók addig, amíg megfelelően illeszkedőt nem talál a program. Minden címkének összehasonlíthatónak kell lennie a szelektorkifejezéssel. Opcionálisan megadható default ág is. Példa:

```
new_obj = case copy_type.state of
(
2: copy $foo
3: instance $foo
default: reference $foo )
```

A szelektor kifejezés (teszt kifejezés) opcionális.

7.3.4. Ciklusok

A ciklusok között megtalálhatók a normál elöl- és hátultesztelős ciklusok, illetve a léptetős for ciklus is. Az elöl- és hátultesztelős ciklusok szintaxisa:

do **<**kifejezés**>** while **<**kifejezés**>** vagy

while **<**kifejezés**>** do **<**kifejezés**>**

A ciklusmag végrehajtása addig folytatódik, amíg a while utáni kifejezés értéke igaz nem lesz. Hátultesztelős esetben a ciklusmag legalább egyszer végrehajtódik.

A for ciklus szintaxisa: for **<**ciklusváltozó**>** (in **|** **=**) **<**sorozat**>** (do **|** collect) **<**expr**>**

A sorozat attól függ, hogy milyen adatokon szeretnénk használni a ciklust (lásd az alábbi példát). A for ciklus kétféle képpen használható: a do kulcsszó esetén minden lépésben kiértékelődik a kifejezés, a collect kulcsszó használata esetén azonban a kifejezések értékét a program egy tömbben el is tárolja, majd azt, mint a for kifejezés értékeként visszaadja (yield).

```
for i = 1 to 10 do print i -- számok sorozata
for item in table1 do x = x + item.height -- egy tömb elemei
for t in 0f to 100f by 5f do sliderTime=t -- időskálán 5 képkockánként lépdelve
bigones = for obj in $box* -- meg lehet adni útvonalnév-sorozatot is!
  where obj.height > 100 collect obj -- összegyűjtjük a magas téglatesteket egy tömbbe
```

A continue kulcsszó segítségével egy ciklus következő végrehajtására ugorhatunk. Ciklusból való kilépésre használható az exit [with **<**kifejezés**>**] azonban ez nem javasolt.

7.4. Alprogramok, modulok

Egy függvénydefiníció szintaxisa: [mapped] (function | fn) **<**függvény_név**>** { **<**parameter**>** } **=** **<**expr**>**. A fuggvények lehetnek rekurzívak és nem feltétlenül kötődnek osztályokhoz vagy struct-okhoz, lehetnek globálisok is.

Példa:

```
function add a b = a + b
fn factorial n = if n <= 0 then 1 else n * factorial(n - 1)
```

A MAXScript-ben vannak függvényváltozók is és egy függvényt hozzárendelhetünk egy változóhoz, így például tömbben is tárolhatunk függvényeket. Függvény lehet paraméter is, vagyis létezik olyan függvény, amelynek valamely argumentuma vagy eredménye is függvény.

A MAXScript-ben alapértelmezés szerint érték szerinti paraméterátadás van, de lehetőség van referencia szerinti paraméterátadásra is, ezt úgy tudatjuk a programmal, hogy a paraméter neve elé "&" jelet írunk.

7.5. Absztrakt adattípusok

Abban az értelemben, hogy a specifikáció és az implementáció nem választható szét, a MAXScript nem támogatja az absztrakt adattípusok létrehozását, azonban támogatja a reprezentáció elrejtését a láthatóság definiálásával.

7.6. Kivételkezelés

A programozók körében ismert tény, hogy még egy jól megírt program futása során is léphetnek fel hibák. Ezek hatására a program normális működését nem tudja folytatni, ezért a hibákat valamilyen módon kezelni kell.

A try kifejezés szintaxisa MAXScript-ben a következő: try **<**védett_kifejezés**>** catch **<**hiba_esetén_kifejezés**>.** A <védett_kifejezés> végrehajtásra kerül és bármilyen elkapott hiba esetén a vezérlés a <hiba_esetén_kifejezés>-re tevődik. Ha nem fordul elő hiba, a catch utáni kifejezés nem hajtódik végre. Ha a <védett_kifejezés> egy blokk-kifejezés, a blokk-kifejezés a hiba helyén felfüggeszti a futást.

Kivétel dobására a throw() függvény szolgál, illetve a throw <hibaüzenet> [<érték>] alakja. Ha egy catch ágon a kivétel kezelésénél továbbdobjuk azt (argumentumok nélküli meghívással), akkor a kivétel a külső, az aktuális try/catch blokkokat tartalmazó, try blokk kivételkezelő ágra dobodik és ez fog lefutni.

Példa:

```
try
(
  i=10
  try ( i.x=1 ) -- futásidejű hibát generál
  catch
  (
    print "Bad Error" -- hibaüzenet kiírása
    try (i.pos=[0,0,0]) -- szintén generál egy futási idejű hibát
    catch (throw()) -- a throw() külső catch()-re adja a vezérlést
  )
)
catch (print "Really Bad Error Occurred") -- hibaüzenet kiírása
```

A catch részen a kivételt a getCurrentException() függvénnyel kérdezhetjük le.

8. Összefoglalás

Napjainkban szinte minden ember kapcsolatba kerül a számítógépekkel, munkája során, tanulás vagy akár más célból és biztosak lehetünk benne, hogy nagy részének nem idegen az AutoCAD vagy a 3dsMax név. Még ha nem is tudják pontosan mi az, valamelyik név biztos ismerősen cseng nekik. Nem hiába választottam én is diplomamunkám témájának e két program bemutatását és elemzését. Úgy hiszem, diplomamunkám segít jobban megérteni, mi is az a számítógéppel segített tervezés és mit várunk tőle, miért ezeket a programokat érdemes használni, miben tudja munkánkat támogatni.

Nem az volt a célom, hogy el lehessen dönteni, melyik programot használjuk inkább munkánk megvalósítására, mivel elemzésem során próbáltam rámutatni arra, hogy mindkét program megfelelő képességekkel bír tervezési, modellezési munkánk elvégzésére, hanem inkább az, hogy egy teljesebb képet tudjunk alkotni arról, hogy ha a két program képességeit ötvözzük, az AutoCAD pontosságát, precizitását a tervezésben és a 3dsMax képességét ezen tervek valósághű kivitelezésére, megdöbbentő eredményeket tudunk elérni. A két program elemzése során rájöttem, hogy idővel a köztük lévő kapcsolat egyre szorosabbá vált és valószínű, hogy a jövőben ez még szorosabb lesz.

9. Irodalomjegyzék

[1] Ted Boardman: 3ds Max –a 3D világa, Prefact Pro Kft. kiadó, 2007

[2] http://nyelvek.inf.elte.hu/leirasok/maxscript/index.php?chapter=1

[3] Pétery Kristóf: AutoCAD 2010 –Fóliák, tulajdonságok, Mercator Stúdió Kiadó, 2009

[4] http://nyelvek.inf.elte.hu/leirasok/maxscript/index.php?chapter=2

[5] Dr. Varga Tibor: AutoCAD 2004-2008 kezdőknek és haladóknak, Computer Studio Kiadó, Győr, 2008

[6] Pintér Miklós: AutoCAD 2000, Computer Books Kiadó, Budapest, 1999

[7] http://www.hungarocad.hu/main.php?folderID=930

[8] AutoCAD 2007: Felhasználói kézikönyv, 2006 Autodesk, Inc.

[9] Vijay Dugall, CADD Primer, A general guide to Computer Aided Design & Drafting – CADD, CAD, Mailmax kiadó, 1999

[10] Tolvaly-Rosca Ferenc: A számítógépes tervezés alapjai, AutoLISP és Autodesk Inventor alapismeretek, Az Erdélyi Múzeum-Egyesület kiadó, 2009

[11] http://www.the-area.com/maxturns20/history

[12] Aurum-Boca: 3ds Max, Aurum DTP Stúdió kiadó, 1997

[13] http://www.sns.hu/index.php?pg=products&id=26

[14] http://www.3dhome.hu/cv_cikkek/cikk_0802

10. Függelék

Függvényábrázolás példa

```
(defun adatok()
  (setq a (getreal "Kerem az a egyutthatot:"))
  (setq b (getreal "Kerem a b egyutthatot:"))
  (setq c (getreal "Kerem a c egyutthatot:"))
  (princ "(a , b , c) = ")
  (princ "(")(princ a)(princ " , ")(princ b)(princ " , ")(princ c)(princ ")")
  (princ)
)

(defun gyokok ()
  (setq elso (* b b))
  (setq masodik (* 4 (* a c)))
  (setq delta (- elso masodik))
  (princ)
    (
    cond (
   (> delta 0) (progn
                  (setq deltagyok (sqrt delta))
                  (setq x1 (/ (- (* -1 b) deltagyok) (* 2 a)))
                  (setq x2 (/ (+ (* -1 b) deltagyok) (* 2 a)))
                  (princ "x1 = ")(princ x1)(princ ", x2 = ")(princ x2)(princ)
                )
         )
       (
   (= delta 0) (progn
                  (setq x1 (/ (* -1 b) (* 2 a)))
                  (setq x2 x1)
```

```
                (princ "x1 = x2 = ")(princ x1)(princ)
              )
      )
      (
   (< delta 0) (progn
                (princ "x1 es x2 komplex gyokok")(princ)
              )
      )
     )
    (setq szelsoertek (- (/ b (* 2 a))))
)

(defun rajzolas()
  (setq hossz (getreal "Kerem az abrazolasi hosszt: "))
  (setq xbal (- szelsoertek (/ hossz 2)))
  (setq xjobb(+ szelsoertek (/ hossz 2)))
  (setq lepes (getreal "Kerem a lepest: "))
  (setq x xbal)
  (while (< x xjobb)
      (setq y (+ c (* b x) (* x x a)))
      (setq pontlista (cons (list x y) pontlista))
      (setq x (+ x lepes))
  )
  (command "pline" (foreach pont pontlista (command pont)))
  (setq pontlista (reverse pontlista))
)

(defun fuggvenyabrazolas (/ xbal xjobb szelsoertek pontlista)
  (adatok)
  (princ "\n")
```

```
  (gyokok)
  (if (or (> delta 0) (= delta 0))
    (progn
      (princ "\n")
      (rajzolas)
    )
  )
)
```

Buborék példa

```
(setq oldCmdecho (getvar "cmdecho")
      oldLayer      (getvar "clayer")
      )
(defun c:bubble()
  (setq cap (getdist "\nKerem a buborek atmerojet:"))
  (command "_.layer" "_new" "TAL_BUBBLE" "_set" "TAL_BUBBLE" "")
  (setvar "cmdecho" 0)
  (draw_bubble)
  (setvar "cmdecho" oldCmdecho)
  (setvar "clayer" oldLayer)
  (princ)
  )
(defun draw_bubble (/ aks pp aks_hand acd_en bh aks_pt1 aks_pt2
                      aks_ang dist1 dist2 circlecenter)

  (setq aks (entsel "\nKerem valassza ki a vonalat:")
        pp (cadr aks)
        aks_hand (entget (car aks))
        acd_en (cdr (assoc 0 aks_hand))
        )

  ;Ha nem vonalat valasztottunk kilep
  (if (/= acd_en "LINE")
        (progn
          (princ "\nA valasztott elem nem vonal!")
          (quit)
        )
        (princ)
  )
```

```
  (setq bh (getstring "\nKerem a buborek cimkejet:"))
  (setq aks_pt1 (cdr (assoc 10 aks_hand))
        aks_pt2 (cdr (assoc 11 aks_hand))
        )
  (setq aks_ang (angle aks_pt1 aks_pt2))

  ;Megvizsgaljuk a vonal melyik vegpontjahoz van kozelebb a kivalasztas
  (setq dist1 (distance pp aks_pt1)
        dist2 (distance pp aks_pt2)
         )

  (if (< dist1 dist2)
    (setq circlecenter (polar aks_pt1 (+ pi aks_ang) (/ cap 2.0)))
    (setq circlecenter (polar aks_pt2 aks_ang (/ cap 2.0)))
    )

  ;Megrajzoljuk a buborekot
  (command "_.circle" circlecenter (/ cap 2.0))
  ;Beirjuk a szoveget
  (command "._text" "_j" "_middle" circlecenter (* cap 0.6) 0 bh "")
)
```