Debreceni Egyetem
Informatikai Kar

A SZÁMÍTÁSTECHNIKAI SZÓRAKOZTATÁS KÉPI MEGJELENÍTÉSÉNEK FEJLŐDÉSE AZ „ŐSIDŐK”-TŐL NAPJAINKIG

Témavezető:
Ledeczky Gábor
számítástechnikai munkatárs

Készítette:
Ignácz Norbert
programozó matematikus hallgató

Debrecen, 2007

Bevezetés

Számítástechnikai szórakoztatás. Meglehetősen tág fogalom, mégis az első dolog, ami az ember eszébe jut, mikor ezt a kifejezést hallja, az a számítógépes játék. A képi megjelenítés fejlődését legjobban ezeken keresztül lehet bemutatni, hiszen az aktuálisan legmodernebb grafikai megoldások mindig a játékokban elevenednek meg. A számítógépes játékprogramokat az alábbi kategóriákba sorolhatjuk:

Kalandjátékok:

- szöveges
- állóképes
- animált

Szerepjátékok (Role-Playing Game)

Stratégiai játékok:

- körökre osztott stratégiák (TBS – Turn Based Strategy)
- valós idejű stratégiák (RTS – Real Time Strategy)
- egyéb stratégiák (társasjátékok, kereskedelmi stratégiák)

Szimulátorok:

- autószimulátorok
- motorszimulátorok
- repülőgép-szimulátorok
- fiktív szimulátorok (űrrepülés)
- egyéb szimulátorok (tengeralattjáró, harckocsi)

Sportjátékok:

- labdarúgás
- kosárlabda
- jégkorong
- küzdősportok
- golf
- tenisz
- egyéb sportok (biliárd, sakk)

Akciójátékok:

- „lődd le mindet” (shoot ’em up)
- „győzd le mindet” (beat ’em up)
- platformjáték
- „első személyű lövöldözős” (FPS - first-person shooter)
- „harmadik személyű lövöldözős” (TPS - third-person shooter)

Ezek csupán az alapkategóriák, melyek az évek során keveredtek egymással. Így akár beszélhetünk szerepjáték-elemekkel megtűzdelt valós idejű stratégiákról, vagy olyan FPS-ről, amiben járműveket lehet vezetni és még sorolhatnánk a különféle hibrid kategóriákat. Az egyes játékstílusok grafikai fejlődését egyenként végigkövetni rendkívül nehéz feladat és jócskán meghaladná e szakdolgozat terjedelmi határait. Ezért a megjelenítés fejlődését az adott korszak legjellemzőbb, illetve legsikeresebb játékkategóriájának egy-egy képviselőjén keresztül szemléltetem.

Célom annak bemutatása, hogy milyen állomásokon keresztül jutott el a játékokban alkalmazott képi megjelenítés az egyszerű karakterektől a mai háromdimenziós grafikai megoldásokig; milyen hatással voltak erre a játékostársadalom aktuális igényei, illetve hogyan fogadták a játékosok a grafikai, valamint játékstílusbeli újításokat, különös tekintettel a monitoron megjelenő erőszakra.

1. Az „ősidők”, a karakteres megjelenítés

Miért is merem „ősidőknek” nevezni a számítógépes játékiparnak ezt a korszakát? Tudjuk, hogy a számítástechnika egészében (nem csupán a képi megjelenítést tekintve) 30 év távlata bizony nagyon hosszú idő. A hardver, a szoftver rohamos léptekkel fejlődik, új mércéket felállítva, aminek eredményeképpen a mai játékokban alkalmazott fotorealisztikus képi megjelenítés ismeretében a 70-es évek kizárólag szöveges felületét joggal nevezhetjük (tisztelettel tekintve arra a korszakra) „ősidők”-nek. Nézzük meg, mi mindennek kellett egy időben megtörténnie ahhoz, hogy ezt a dolgozatot egyáltalán legyen miről megírnom!

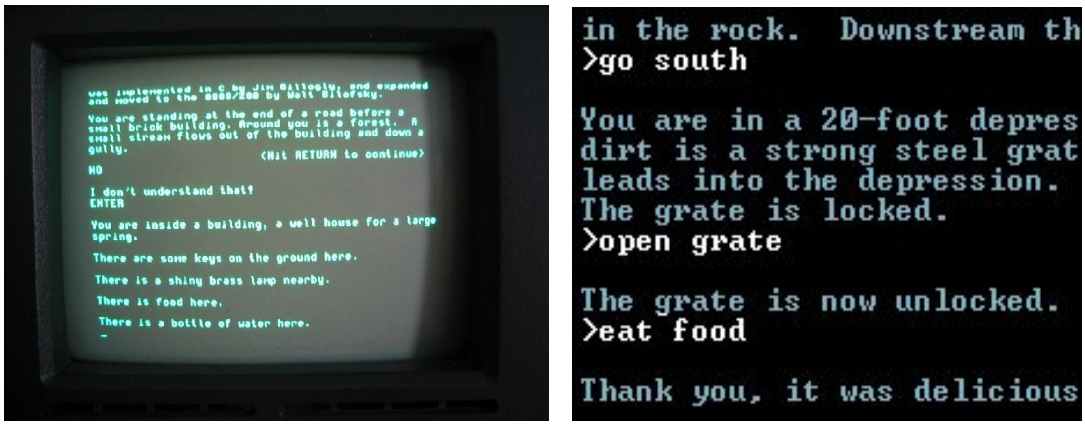
1.1 Az első számítógépes játék, az Adventure

Az első játékprogram megalkotása Will Crowther nevéhez fűződik. Will végzettségét tekintve fizikus volt és ezáltal a számítástechnikában is otthon érezte magát (ugyanis abban az időben – ’50-es évek - még nem volt számítástechnikai szak, a számítógépek a fizikai karhoz tartoztak, mert csak ott voltak olyan emberek, akik tudtak valamit kezdeni azokkal). 1968-tól a BBN-nél (Bolt, Beranek, Newton) dolgozik az ARPANET-en, az INTERNET ősén. A fiatal tudós a munkáján kívül a természet meghódításában leli örömét, szenvedélyesen szereti a barlangászást. Egy túra alkalmával ismerkedik meg egy fizikuslánnyal, Patriciával, akit nem sokkal később feleségül is vett. Will mindig megtalálta a módját, hogy a hobbiját és a munkáját összekapcsolja: a barlangásztúrákon ő volt a térképész, hazatérve az adatokat mindig felvitte a számítógépére és a már működőképes ARPANET segítségével megosztotta azokat a kollégákkal, illetve ők is felvihették saját adataikat. Másik szenvedélye a Dungeons & Dragons szerepjáték volt (aminek szintén jutott szerep az első számítógépes játék megalkotásakor).

Will házassága hamar megromlott, 1972-ben elváltak és Patricia magával vitte két (7 és 5 éves) kislányukat. Crowther kétségbeesett, félt, hogy lányai elidegenednek tőle. Ekkor határozta el, hogy ír egy programot a számukra, amely az általa rajzolt térképeken és barlangászélményeken alapul, hozzákeverve a Dungeons & Dragons szerepjáték elemeit.

Idézzük fel saját szavait: „Az ötletem az volt, hogy ez egy olyan számítógépes játék kell, hogy legyen, amely nem rémíti el a komputerekhez nem értő embereket. Ez volt az oka, hogy úgy írtam meg, hogy a játékos a természetes, beszélt nyelv szavaival irányítsa a játékot egységes parancsok helyett.”

A programot néhány hétvége alatt írta meg FORTRAN nyelven, a munkahelyi DEC PDP10-es számítógépén. Azért esett a választása a FORTRAN-ra, mert úgy gondolta, hogy ez egy egyszerű, hordozható nyelv, ezáltal több platformra is le lehet fordítani. Az ominózus játékot megalkotója Adventure névre keresztelte. Egy szöveges kalandprogramról van szó (szó sincs semmiféle grafikus felületről!), amelynek célja, hogy a játékos felfedezze a föld alatti világot, megtaláljon öt rejtett kincset, miközben olyan természetes akadályokkal kell megküzdenie, mint egy kígyó, vagy a barlangászok hű karbidlámpájának limitált „élettartama”, vagyis ha a játékos nem vigyázott, könnyen magára maradhatott a barlangmélyi sötétben. A program minden lépés után kiírja, hogy hol van a játékos és mit lát, majd várja a parancsokat. Crowther készített egy szövegelemzőt, amely a játékos által beírt egy-két szavas utasításokat („look”, „inventory”, „get key”, „eat food”, „open door”) értelmezte és annak megfelelően cselekedett. *(1. ábra)*



1. ábra Az ADVENTURE futás közben egy Osborne II számítógépen 1982-ben

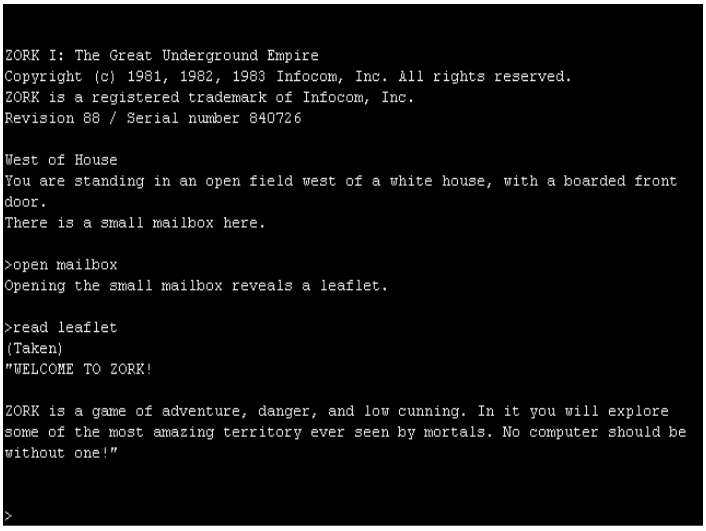
Az Adventure nagyon megtetszett Will lányainak és mindenkinek, aki kipróbálta és hamar elterjedt a szárnyait bontogató hálózaton. Will mégsem fejlesztette tovább, hanem 1976-ban, amikor a Xerox céghez hívták dolgozni, maga mögött hagyta a játékot és vele a hozzá kötődő keserédes emlékeket, majd beletemetkezett az új munkájába.

1.2 Az Adventure követői

Még nincs vége az Adventure történetének, hiszen 1976-ban eljutott a játék a fiatal, tehetséges programozóhoz, Don Woods-hoz, akit lenyűgözött a tény, hogy „a számítógép egy interaktív játékospartner, az ember a számítógéppel játszhat.” A fellelkesült Woods elkérte Crowthertől a forráskódot és hatalmas lendülettel állt munkához.

Woods pár hónap alatt átírta az Adventure-t, megbonyolította a barlangrendszert, elhelyezett benne egy kalózt, néhány kincset, melyek megtalálásáért pontot kapott a játékos (maximálisan 350-et, emiatt sokan azóta is „Crowther/Woods-féle 350 pontos Adventure játék”-ként ismerik a programot). Maga Woods a játékot Colossal Caves-nek nevezte el és hála az ARPANET-nek, az szinte pillanatok alatt elterjedt az Egyesült Államokban. Így született meg az első játékprogram és nyomában az egész számítógépes játékipar. *(Függelék, 47. ábra)*

Rövid idő alatt mások is jelentek meg hasonló játékokkal, hogy csak a két leghíresebbet említsük: 1977 júniusában a Zork *(2. ábra)*, egy trilógia első darabja (sikeréből megalapították az Infocom nevű játékcéget), amelyből az évek során több mint egymillió példányt adtak el, vagy Scott Adams Adventureland *(3. ábra)* című játéka 1978-ban, amely szintén egy sorozat (Classic Adventures) első darabja volt. [1]



2. ábra Zork

A grafikus verzió doboza

Első, szöveges verzió

Grafikus verzió Commodore 64-re

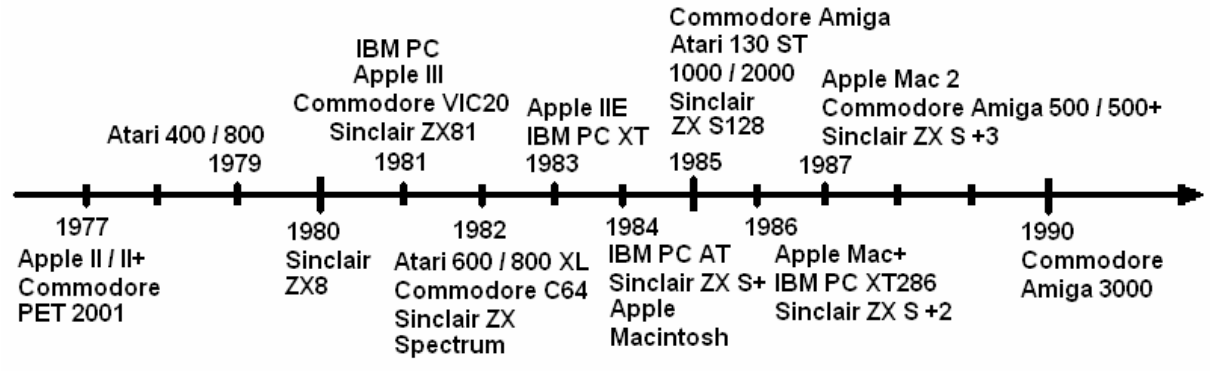
3. ábra Az Adventureland fejlődése

Az ARPANET-nek hála, ezekre az alapokra épült egy új játékstílus, a MUD (Multi User Dungeon, Domain vagy Dimension), amely játékok kombinálták a szerepjátékok, hack'n slash („üsd-vágd") játékok elemeit a chat-szobákkal. Ugyanúgy karakteres megjelenítést alkalmaztak, mint elődeik (persze idővel megjelent a grafikus felület) és rengeteg játékost szögeztek a gépek elé. Hobbiként játszottak a MUD-okkal, egy-egy professzionális fejlesztésű MUD-ért havidíjat kellett fizetni. A mai MMORPG-k (Masszív Többjátékos Online SzerepJáték) a MUD-ok leszármazottjai.

Amint a képeken is látszik, ezek a játékok valóban az ősei a mai háromdimenziós „csodáknak". Itt még azt lehet mondani, hogy a játékos fantáziájára volt bízva, hogy mennyire éli bele magát a játékba, hiszen a megjelenítés még rendkívül kezdetleges volt. Nem volt elterjedt a grafikus megjelenítésre alkalmas hardver sem, sőt kezdetben az ilyen irányú igény sem merült föl, hiszen az ember már annak örült, hogy egyáltalán játszhatott a számítógépével.

2. Kétdimenziós megjelenítés

Az 1980-as évek a kétdimenziós grafikai megjelenítés virágkora. Ahogyan az előző fejezetben írtam, a karakteres megjelenítésű játékok viharos gyorsasággal váltak a játékos kedvű ember, fő időtöltésévé, ami elindította a játékok grafikai fejlődésének máig nem csillapodó hullámát. Az évek során megnőtt az igény a szebb, kifinomult grafikával rendelkező játékok iránt és a ’80-as évekre már a hardver szempontjából is eljött az idő továbblépésre. Szinte évente jelentek meg az egyes platformok új változatai, gazdag „táptalajt” biztosítva a lelkes programozók és immár a grafikusok számára is. *(4. ábra)*



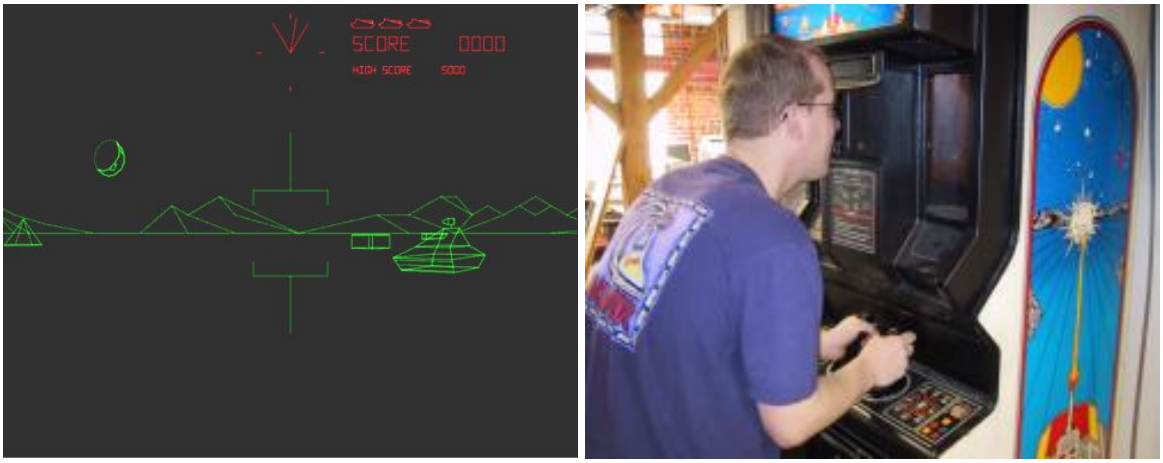
4. ábra A '80-as évek otthoni számítógépei

Kétfajta megjelenítési struktúrát alkalmaztak a kor játékaiban, az egyik a vektorgrafika, a másik a raszter- vagy más néven pixelgrafika (ezen belül az izometrikus-nézetű játékokkal külön foglalkozom).

2.1 Vektorgrafikus játékok

2.1.1 Battlezone

Vektorgrafikát használt az 1980-ban megjelenő Battlezone. Ez egy ún. arcade játék volt (játéktermekben, éttermekben, érmével működő gépen játszható). A ’80-as években számos platformra átírták (DOS, Apple II, Atari ST, Commodore 64, Sinclair ZX Spectrum, Atari XEGS és Atari Lynx), így hamar eljutott az otthonokba is. A vektorgrafika huzalvázas (wireframe) modellezési technikáját használta, ennek (és a csupán két szín – zöld és vörös - használatának) köszönhetően meglehetősen egyszerű volt a képi világa *(5. ábra)*, mégis a magával ragadó játékmenet és az elsőként alkalmazott első-személyű (first-person) nézet megtette a hatását és évekig népszerű játéknak számított.

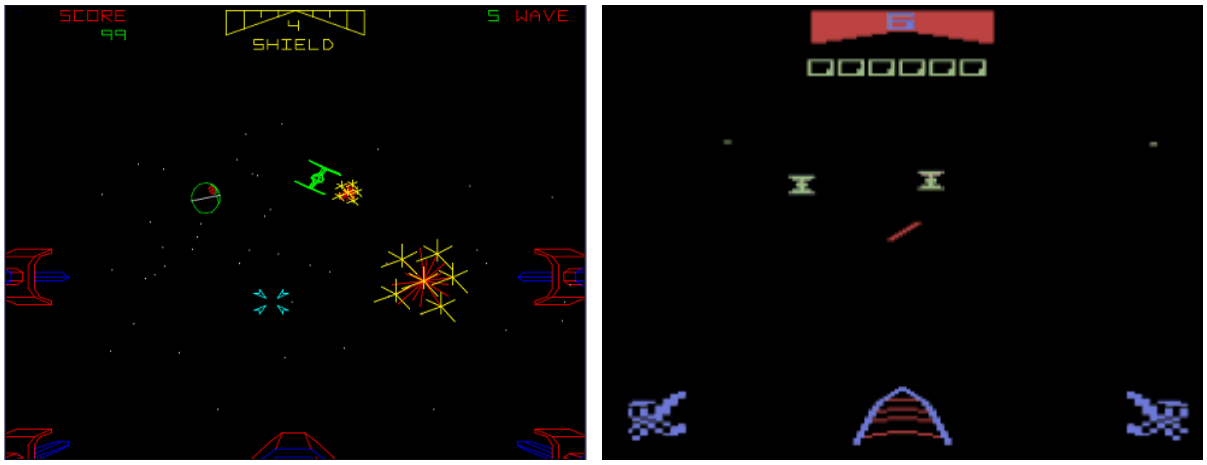


5. ábra A Battlezone játéktermi változata és maga a játékgép

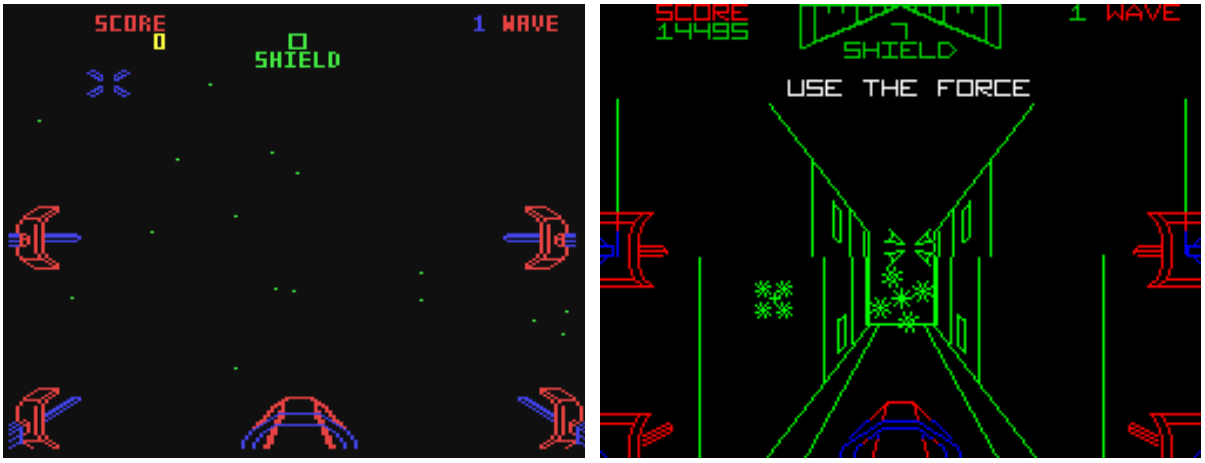
6. ábra A Battlezone Atarin és Commodore 64-en

2.1.2 Star Wars

1983-ban a nagyközönség még jócskán a Star Wars lázban égett, így kézenfekvő volt, hogy születik egy arcade játék ugyanezzel a címmel. Ez a program is vektorgrafikát alkalmazott, de a Battlezone-hoz képest jóval több színben pompázott. A játék célja természetesen a Halálcsillag megsemmisítése volt, mindezt egy X-wing pilótafülkéjéből nézve kellett végrehajtani. Bár nem grafikai vonatkozású, de érdemes megemlíteni, hogy a játék a filmekből vett eredeti hangok digitalizált változatát használta, jelentősen emelve ezzel a játékélményt. A játék sikert aratott és ahogy az ilyenkor lenni szokott, el is készült szinte minden akkori platformra. *(7-8. ábra)*



7. ábra A arcade Star Wars és Atari 2600-ra átírt változata



8. ábra A Star Wars Commodore 64-re és ZX Spectrum-ra

A vektorgrafika tehát leginkább a játéktermi gépeknél hódított és főként akciójátékok készültek a segítségével. Vajon a rasztergrafika mit tudott felmutatni a ’80-as, ’90-es években? Tekintsük át a fontosabb állomásait és meglátjuk, hogy nem is keveset!

2.2 Pixelgrafikus játékok

2.2.1 Mistery House

1980-ban jelent meg az első olyan játék (Ken és Roberta Williams „gyermeke”), amely a szöveget grafikával kombinálja. Ez volt a Mistery House. A grafika volt az egyetlen (de milyen nagy) különbség közte és az addigi szöveges kalandjátékok között. A program Apple II-re íródott, szó sem volt animációról, színekről, hangról, csupán 70 db egyszerű kétdimenziós rajzból állt (melyek Roberta keze munkái). Az Apple II gyenge grafikus képességeit mutatja, hogy a fehér színt a zöld és a lila kombinálásával állította elő, ami középen fehér színt eredményezett, míg a széleken, a függőleges éleknél „átütött” a zöld és a lila. *(9. ábra)*

9. ábra Mistery House

Említésre méltó a játék fejlesztésének története. Meglepő vagy sem, Ken és Roberta ismerték a Colossal Caves című kalandjátékot, majd miután mindketten végigjátszották,

szerettek volna valami hasonlóan jó programot vásárolni. Roberta-nak nem volt baja a szöveges megjelenítéssel, mégis úgy érezte, nagyobb élményt nyújtana, ha képeket is használnának a szöveg mellé. Mivel ilyen nem volt akkoriban a játékpiacon, úgy gondolták, hogy elkészítik maguk az első ilyen játékot. Ken néhány éjszaka alatt megírta a programot az Apple II-jén, majd egy helyi szoftverboltban árulni kezdték. Nem kis meglepetésükre, több mint 10000 példányt adtak el belőle, ami abban az időben rekordnak számított. A pénzből Ken 1980-ban megalapította az On-Line Systems nevű céget, amit 1982-től Sierra On-Line-ként ismerünk (és napjaink egyik vezető cége Sierra Entertainment néven). *(Függelék, 49. ábra)*

2.2.2 King's Quest-sorozat

Szintén a Sierra On-Line nevéhez fűződik az 1984-ben IBM PCjr-re készült King's Quest. A játék az első „animált" kalandjátékok közé tartozott, látványvilágát továbbra is a cég egyik alapító tagja, Roberta Williams álmodta monitorunkra. A korszak hasonló játékaival ellentétben a King's Quest nem egyetlen ember több hetes munkája, hanem hat, teljes munkaidőben dolgozó programozó készítette nem kevesebb, mint 18 hónap alatt. Ezáltal idejének legköltségesebb projektje (több mint 700 ezer dollárba került a fejlesztés). A játék 16 színű grafikát használt, 160x200-as felbontással, de ami igazán kiemelte a többi kalandjáték közül, az az animált karakterek és az interaktív környezet. Míg korábban minden egyes „szoba" egy előre megrajzolt, statikus háttér volt, némi szöveggel kiegészítve, ráadásul a főszereplő sem volt látható, a King's Quest Sir Graham nevű hőse egy teljesen animált kétdimenziós karakter, akivel az iránybillentyűk segítségével járhattuk be a szebbnél-szebb, helyenként animált háttereket. Nem volt többé szükség az iránytű használatára (és az „Észak", „Dél", stb. irányok begépelésére), karakterünket a képernyő egyes széleihez vezetve juthattunk el a következő játékbeli területre. A főhőst továbbra is a megfelelő parancs begépelésével bírhattuk cselekvésre, azzal a különbséggel, hogy például egy ajtó kinyitása esetén, az „open door" szavak hatására nem egyszerűen egy zárt ajtó képe cserélődött le a nyitott ajtó képére, hanem animációt használva, láthatóan kinyílt. A játéknak - köszönhetően az IBM PCjr gyenge fogadtatásának - egy évet kellet várnia az átütő sikerre. Ekkor Tandy 1000-re és több korabeli IBM „klón"-ra is átírták, valamint az Apple II, Amiga, Atari gépek sem maradtak King's Quest nélkül. A játék első része az évek során több kiadást is megért

(11. ábra), ezek leginkább grafikájukban tértek el egymástól. Tekintsünk át néhányat közülük:

- 5. kiadás (1987, PC): átdolgozott grafika, EGA (Enhanced Graphics Adaptor) támogatással
- 6. kiadás (1990, PC): ez a King's Quest úgynevezett „továbbfejlesztett" változata; még mindig 16 színt használ, de a korábbihoz képest kétszer akkora (320x200) képernyőfelbontással, az egeret és a hangkártyát is támogatja
- nem hivatalos újrafeldolgozott (remake) kiadás az AGD Interactive által (2001, PC): 256 színű VGA grafika, beszélő (full-speech) karakterek

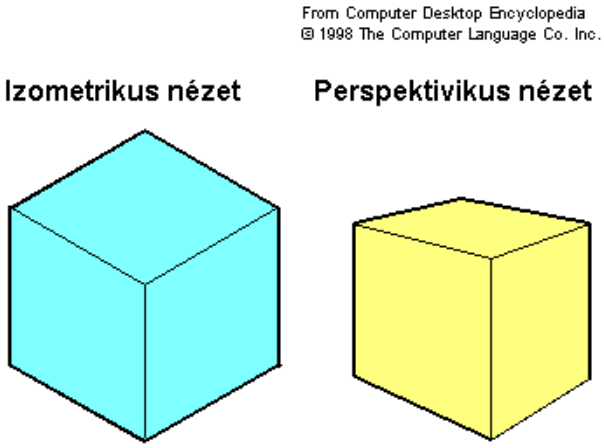
10. ábra A King's Quest I. Tandy 1000-en (bal) és a "továbbfejlesztett", 6. kiadás (jobb)

11. ábra King's Quest I. nemhivatalos "remake" (2001)

1998-ig Ken és Roberta Williams hét folytatással ajándékozták meg a King's Quest kalandjátékok szerelmeseit. *(Függelék, 50. ábra)*

2.3 Izometrikus nézetű játékok

A '80-as '90-es években a hagyományos kétdimenziós megjelenítés mellett előszeretettel alkalmazták egyes játékokban az „izometrikus" leképezést *(12. ábra)*, aminek segítségével lehetőség nyílt látszólag „háromdimenziós" környezet létrehozására két dimenzióban. Mivel ebben az esetben a játéktérbeli objektumok mérete nem változik mozgás közben, így a számítógépnek nem kell a méreteken változtatnia, vagy külön számításokat végeznie a 3D hatás eléréséhez. Így a 8 illetve 16 bites rendszerek is képesek viszonylag nagy „háromdimenziós" területek megjelenítésére.



12. ábra Kocka izometrikus és perspektivikus nézetben

2.3.1. Little Big Adventure

Az 1994-ben napvilágot látott Little Big Adventure az említett izometrikus nézetet használta. A játékban az összes karakter és jármű pedig teljesen háromdimenziós, árnyalt, poligonalapú modellek, amelyek ezáltal teljesen körbeforgathatók és bármilyen irányban elmozdíthatóak voltak (mindez SVGA felbontásban). Kalandjátéknak titulálják, bár tartalmazott akció- és szerepjáték-elemeket is. A játékot folytatás követte 1996-ban, amely grafikailag annyiban különbözött elődjétől, hogy a külső helyszínek immár teljesen háromdimenziós, perspektivikus nézetben jelentek meg, a szobabelsők maradtak

izometrikusak. A fejlesztő Adeline Software mindkét epizód esetében sikert könyvelhetett el, előbbiből 400 ezer, utóbbiból 600 ezer darab kelt el világszerte.

13. ábra Külső terek a Little Big Adventure első és második részében

2.3.2. Diablo-sorozat

Az izometrikus nézetet használó játékok közül talán a leghíresebb, szintén egy sorozat első tagja, a Blizzard gondozásában megjelent 1997-es Diablo. Megalkotta a point-and-click akció-szerepjáték kategóriát, azóta is számos játék „táplálkozik" belőle *(Függelék, 51. ábra)*. A játékban minden helyszín az említett nézetben volt ábrázolva, a főhős nem egyik háttértől a következőig mozgott, hanem a képernyő közepén elhelyezkedő hős „alatt gördült" a környezet (scroll-ozás). A karakterek kétdimenziósak, a 3D-s hatást azzal érték el, hogy 8 különböző nézetből rajzolták meg őket és ezeket változtatta a játék attól függően, hogy milyen irányban haladunk. Különlegessége a Diablo-nak, hogy elméletben nem lehet kétszer ugyanolyan felépítésű pályákon végigjátszani, ugyanis véletlenszerű térképgenerátor hozza létre az egyes játékbeli helyszíneket. Így többszöri újrajátszás után sem haladhatunk emlékezetből, mindig tartogat valami újat számunkra. Ugyanez a helyzet a kalandozás során fellelhető fegyverekkel, páncélzattal is, nem tudhatjuk előre, hogy mit fogunk találni egy-egy legyőzött ellenfelet átkutatva. Ez a megoldás (és persze a beépített többjátékos üzemmód) megalapozta a Diablo sikerét. Világszerte ismerik és azóta is játsszák, ahogyan a 2000-ben megjelenő folytatást is (Diablo 2).

14. ábra Diablo 1 és Diablo 2

3. „Két és fél dimenzió”, avagy majdnem 3D

A 2.5D-s játékok a kilencvenes évek első felében ütötték fel a fejüket a játékpiacon, nem kis feltűnést keltve.

3.1 Wolfenstein 3D

Az első igazán sikeres darab az id Software Wolfenstein 3D című játéka, ami 1992-ben alapjaiban változtatta meg az addigi akciójátékokról kialakított képet, népszerűsítve az „első-személyű lövöldözős” (First-Person Shooter) játékkategóriát. Abban az időben a játékipar még mindig gyerekcipőben járt, a nagy sikereket elért SimCity egy virtuális város felépítésének és pátyolgatásának feladatát rótta a játékosra, míg mások történelmi csatákat vívtak a Civilization-nel – mindezt pörgő játékmenet és brutalitás nélkül. A Wolfenstein 3D olyasvalamit vitt a számítógépes játékok világába, amit ennyire nyersen még senki nem tett meg előtte: az erőszakot. A Wolfenstein 3D két alapelemre építkezett: John Romero (grafikus) brutálissá tette, John Carmack (engine programozó) *(Függelék, 48. ábra)* pedig gyorssá (ez szöges ellentéte az alapkoncepciót ihlető, hasonló című, kétdimenziós Castle Wolfenstein-nek, amiben a hangsúly a lopakodáson volt). A grafikus motor C és x86 Assembly nyelven íródott, arra az ötletre épített, hogy csak azzal foglalkozzon mindig, amit a játékosnak éppen látnia kell. Az engine által használt technika a raycasting (sugárlövés), ami a raytracing-hez (sugárkövetés) hasonló, azzal a különbséggel, hogy csupán egy látótengelyt (line of sight) számol ki, ami a játékos „szeméből” az adott pixel felé halad (ezáltal nem képes olyan effektekre, mint az árnyékvetés (shadowing), tükrözés (reflecting)). Az objektumok megjelenítését sprite-okkal oldották meg, (szintén a sebesség érdekében), textúrázott falak voltak a játékban, de talaj és plafon nem, valamint a magasságkülönbségeket sem tudta kezelni az engine (tehát minden szoba ugyanazon a szinten helyezkedett el). Egy egyszerű 1 dimenziós mélységi buffert használt annak eldöntésére, hogy melyik falat hogyan kell kirajzolni, illetve az egyes objektumokat felépítő sprite-ok méretét milyen mértékben kell változtatni (scale). Fel-le nézelődésről szó sem eshetett, ez az engine még nem volt képes erre. Végsősoron Carmack-nak sikerült elérnie a célját: egy olyan grafikus motort alkotott, ami a játék stílusának leginkább megfelelt. A rendkívül gyors, magával ragadó játékmenet

elkápráztatta az addig egyáltalán nem ehhez szokott játékostársadalmat, ezt a többi játékfejlesztő csapat is észrevette és sorra jöttek a „klónok”. *(Függelék, 52. ábra)*

15. ábra Wolfenstein 3D

A Wolfenstein 3D-nek először egy shareware (szabadon letölthető) verziója terjedt el az interneten mindössze 10 pályával és csak később jelent meg hivatalos kiadása, amely további 50 missziót tartalmazott.

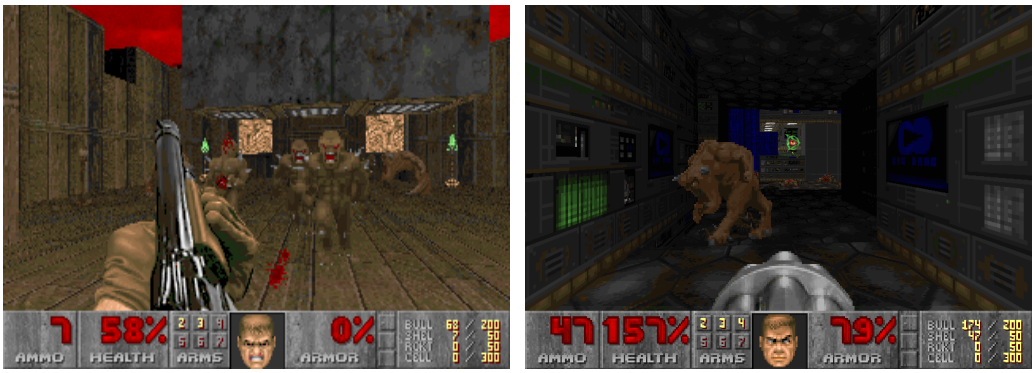
3.2 Doom

A Wolfenstein 3D sikerét követően John Carmack nem tétlenkedett és már 1992-ben hozzákezdett egy új grafikus motor, a Doom-engine kifejlesztéséhez, míg a csapat többi tagja a Wolfenstein 3D-hez készítette a Spear of Destiny című kiegészítést. A Doom hangulatát, kinézetét a korabeli Alien és egyéb fantasztikus- illetve horrorfilmek ihlették, konkrétan egy tengerészgyalogost alakított a játékos, akire különféle pokolbéli szörnyek támadnak. Mindezt a Carmack által létrehozott engine meggyőzően vitte a képernyőre *(16. ábra).* A következő pontokban a Doom-engine által bevezetett újításokat mutatom be:

- magasságkülönbségek kezelése
- nem merőleges falak, azaz két fal már bármilyen szögben kapcsolódhat egymáshoz (a Wolfenstein 3D-ben a falak négyzetes rács mentén haladtak, így kizárólag 90°-os szögben kapcsolódhattak), így változatosabb helyszínek hozhatóak létre

- fegyver mozgása séta, vagy futás közben (a Wolfenstein 3D-ben a játékos fegyvert tartó „karjai” a képernyő közepéhez voltak rögzítve), ezzel sokkal dinamikusabbnak hatott a játék
- teljes texture mapping minden felületen (azaz a padló és a plafon is textúrázott)
- különböző megvilágítású pályák (a Wolfenstein 3D-ben minden terület teljesen kivilágított volt, ugyanazzal a fényerő (brightness) értékkel), melyek nem csak az egyedi látványt adják meg a Doom-nak, hanem nagymértékben hozzájárulnak az atmoszféra megteremtéséhez, sőt a játékmenetre is hatással vannak – a sötétséget a félelemkeltés eszközeként addig még egy játékban sem alkalmazták
- interaktív környezet (mozgó platformok, hidak, lépcsővé átalakuló padló)

Carmack-nak néhány trükköt kellett alkalmaznia, hogy a korszak otthoni gépein a játék megfelelő sebességgel fusson. A legfontosabb ezek közül, hogy a Doom pályái valójában nem háromdimenziósak, belső reprezentációjuk síkban történik, a magassági különbségeket csak később adja hozzá az engine. Ezáltal korlátai is vannak a grafikus motornak: például nem lehet egy szobát egy másik fölé helyezni. A pályák kétdimenziós belső ábrázolásának viszont előnye is van: nagyságrendekkel gyorsabb a renderelés. Erre az úgynevezett BSP (Binary Space Partitioning) módszert használja, aminek lényege, hogy a pályát egy bináris fára osztja fel: a gyökér a teljes pályát jelenti, az egyes csomópontokban az aktuális térrészt kettészelő hipersík van, a levélelemek az így kialakult térrészek. Az algoritmus a gyökérelemből kiindulva a pályának csak azon részeit rajzolja ki, amelyeket ténylegesen „lát” a kamera, ezzel jelentős számítási időt spórol meg. A nézelődés továbbra sem lehetséges, a magasabban lévő ellenfelek eltalálásához automata célzást használt a játék.



16. ábra Doom

A Doom szintén az interneten hódított először, 1995-re (mikor a hivatalos verzió a boltokba került Ultimate Doom címen) már 10 millió számítógépre volt felinstallálva a játék a shareware-verziója. A Doom mindenhová eljutott, a legtöbb problémát az olyan munkahelyeken okozta, ahol az alkalmazottak egymás ellen vívott Doom deathmatch-ei (többjátékos mód) leterhelték a cég számítógép-hálózatát. Természetesen folytatások is születtek, a második rész alig egy évvel az elsőt követően, a Doom 3-ra viszont várniuk kellett a játékosoknak, egészen 2004-ig. Nem kétséges, hogy a Doom mérföldkőnek számít a számítógépes játékok történetében (az engine-t számos fejlesztőcég megvásárolta, így születtek pl. a Heretic, és Hexen c. játékok), az elkövetkezendő években (sőt, akár manapság) megjelent hasonló játékokat egyszerűen csak „doom-klón”-ként emlegetik. *(Függelék, 53. ábra)*

3.3 Duke Nukem 3D

Valószínűleg a Doom sorozat sikere inspirálta Ken Silverman-t, aki a kilencvenes évek közepén előállt saját fejlesztésű grafikus motorjával, a Build-engine-nel, amit a 3D Realms nevű cég meg is vásárolt a Duke Nukem 3D című FPS-éhez. Az engine nagyon sok tekintetben a Doom motorjához hasonlít: világának belső reprezentációja kétdimenziós, felépítéséhez 2D-s alakzatokat (sector) használ és a pályák objektumokkal való benépesítéséhez szintén egyszerű sík objektumokat (sprite) alkalmaz. Később adja hozzá a magassági komponenseket, hiszen minden sector-nak különböző mennyezet- és padlómagassága lehet.

A Build grafikus motort az teszi különbözővé a Doom-engine-től, hogy sokkal összetettebb, rugalmasabb világ hozható létre vele. A sector-ok valós időben manipulálhatók; alakjuk, magasságuk, dőlési szögük bármikor megváltoztatható anélkül, hogy a renderelési információt újra kellene számolni. Ezáltal lerombolható környezetet alkalmazhatnak a játékokban (ezt a Duke Nukem 3D még nem tudta, csak az egy évvel későbbi - szintén Build-engine épülő - Blood). Továbbá a sector-okban a falaknak, a padlónak speciális karakterisztikát lehet adni, azaz egy ilyen speciális padlóra lépve a játékost az engine egy másik sector-ba vitte át. Ez gyakorlatban annyit tesz, hogy a játékos „leeshetett” egy lyukon át egy nagyobb szobába, beugorhatott a vízbe (a víz felszíne alatt egy másik sector-ba érve), vagy liftet használhatott. A Duke Nukem 3D az első két és fél dimenziós játék, amelyben

nézelődni lehet (bár ez még nem valódi fel-le nézés). Az „y-shearing” nevű módszerrel éri el a megfelelő hatást: megnöveli a függőleges felbontást (y) és egy ablakot hoz létre az adott területen, majd ezt az ablakot fel-le mozgatva a felfelé, illetve lefelé nézés illúzióját teremti meg a játékos számára. A perspektíva szempontjából csupán a horizontális távolságot veszi számításba, azaz akárhogyan is nézelődik a játékos, a függőleges vonalak mindig függőlegesek maradnak. *(17. ábra, bal oldal)*

17. ábra Duke Nukem 3D

A Build-engine későbbi verziói támogatták a voxelek használatát is. Ez szintén csak a Blood-ban jelent meg, ott a lőszeresdobozok és néhány tereptárgy már nem sprite-okkal, hanem voxelekkel volt megjelenítve. *(18. ábra)*

18. ábra Voxeles tereptárgy a Blood-ban

A Duke Nukem 3D nem csak engine-jében, de hangulatában is különbözött az őt megelőző Doom-tól. Ebben a játékban is szörnyekkel harcolt a főhős, de az egészet humoros megvilágításba helyezte, szó sem volt félelemkeltésről. A játékos egy napszemüveges „macsót” alakított, aki a játék folyamán sorozatosan (nem kevésszer obszcén kifejezésekkel) kommentálta a képernyőn történteket. Hosszú lenne felsorolni, hogy hány, akkoriban kasszasiker mozifilmet parodizált a játék; a konkurens FPS-ek, a Doom, valamint a még fejlesztés alatt álló Quake sem maradt ki az élcelődésből. Ez nem is lett volna baj, de a játékot a helyenként pornográf tartalma miatt a kritikusok sorozatosan támadták és számos országban csak cenzúrázva jelenhetett meg. Ennek ellenére sikeres lett, egy kiegészítő csomagot készített hozzá a 3D Realms (ez volt az Atomic Edition) és 1997-ben bejelentette a Duke Nukem Forever című folytatást, amit 10(!) éve fejlesztenek és már régen nem hiszi senki, hogy valaha is elkészül.

4. Háromdimenziós megjelenítés

A fejlődés nem állt meg, a kilencvenes évek második felére már elterjedt a CD-ROM, a Windows 95, egyre gyorsabb hardverek láttak napvilágot. Ezek mind szükségesek voltak ahhoz, hogy az újdonságra szomjazó játékostársadalmat minél igényesebben tudják kiszolgálni a fejlesztőcégek. Lehetőség nyílt a valóban háromdimenziós virtuális világok létrehozására, útjára indult a 3D-s forradalom. Az akciódús FPS játékokba belekóstoló közönséget egyre kevésbé kötötték le a kalandjátékok (mely játékkategória a ’80-as évek egyik legkedveltebbje volt), elsősorban FPS-t akartak, csak szebbet és jobbat, mint az addigiak. A 3D alkalmazása nem feltétlenül volt hasznos egyes játékstílusokat tekintve: a legtöbb stratégiai játék kezelhetetlenné vált, a kalandjátékok túlságosan sterilek lettek a háromdimenziós megjelenítés miatt, nem volt meg bennük a korábbi kézzel rajzolt elődeik varázsa. Ez ahhoz vezetett, hogy a kalandjátékok szépen lassan szinte teljesen eltűntek a játékpiacról. A 3D abszolút nyertesei az akciójátékok, azon belül is az FPS-ek, ez az a játékkategória, ami a professzionális háromdimenziós megjelenítéssel a leginkább képes elhitetni a játékossal, hogy teljes mértékben részese a képernyőn történteknek.

4.1 Future Shock

A Bethesda Softworks szintén filmes témát (konkrétan a Terminátor sorozatot) választott a Future Shock című játékához, amely FPS 1995-ben az elsők között használta a háromdimenziós környezetet és az egérrel történő valódi (nem „y-shearing”) nézelődést. A játékmotor neve Xngine, mely támogatta a teljes textúrázást, a phong-árnyalást, valamint a valós idejű fényforrásokat. Míg a környezet, az épületek és az ellenfelek egyaránt háromdimenziósak voltak, a felvehető tárgyak, fegyverek és a pályák dekorációja a már kissé „megkopott” sprite-ok alkalmazásával kerültek megjelenítésre. Egy évvel később, 1996-ban elkészült egy hivatalos kiegészítő, Skynet címmel. Az Xngine nem esett át jelentős ráncfelvarráson, csupán annyiban tért el, hogy már támogatta a 640x480-as felbontást is a

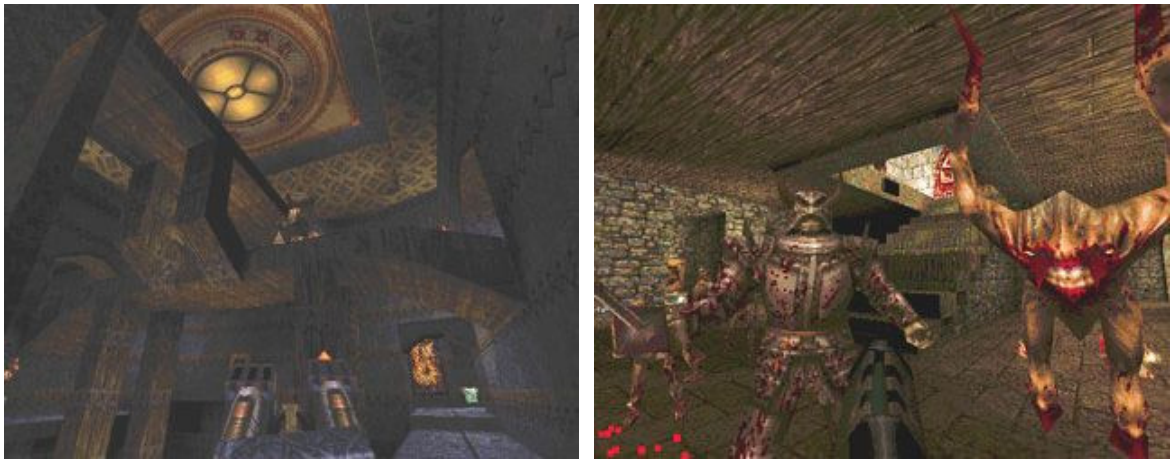
320x200 mellett és ezt az opciót a Skynet feltelepítése után a Future Shock-nál is lehetett használni.

19. ábra Future Shock és a Skynet

4.2 Quake-sorozat

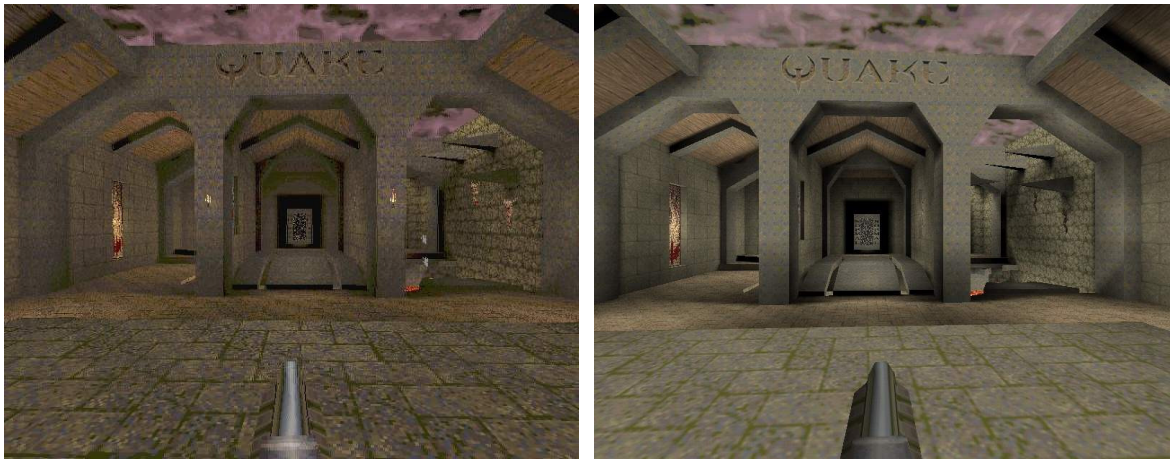
Talán nem meglepő, hogy az első teljesen háromdimenziós FPS John Carmack, valamint az id Software nevéhez fűződik. Ez a játék az 1996-ban megjelent *Quake*, ami megalkotójához híven ismét ámulatba ejtette a világot. Az alapverzió még se a DirectX, se az OpenGl API-kat nem használta, de ezek nélkül is (szintén Carmack „hagyomány") olyan látványt nyújtott, amilyet addig még egy játékban sem láttak. A Quake-engine által alkalmazott technikák - mint ahogyan az a Doom esetében is történt – később minden FPS-ben alapkövetelménnyé váltak:

- 3D-s modellek alkalmazása a játékos és az ellenfelek megjelenítésére
- a pályákat már nem kétdimenziós térképpel, hozzáadott magassági adatokkal hozza létre, hanem ténylegesen 3D-s területként vannak tárolva (így már nincs szükség torzításra nézelődéskor)
- lightmap-ek használata a falakon és a nem mozgó objektumokon, goraud-shading a mozgó objektumokon (kis számításigényű árnyalási eljárás alacsony poligonszámú modelleken)



20. ábra Quake

Az 1996 végén megjelent *VQuake* elsőként támogatta a hardveres gyorsítású renderelést, de kizárólag a Rendition Verité chipset-et használó grafikus kártyákkal (ami nem volt igazán elterjedt akkortájt). Amellett, hogy a VQuake sokkal gyorsabban futott, az első, szoftveres renderelésű verziót látványban is lekörözte. A 16 bites színmélység, a bilinear filtering, a még szebb, dinamikus fények és a bekapcsolható anti-aliasing (élsimítás), mind hozzájárultak a látványorgiához. Carmack rájött, hogy ez mit sem ér, ha csak kevés játékos tapasztalhatja meg. Ezért az id Software 1997 januárjában elkészítette a *GLQuake*-et, ami a mellékelt meghajtóprogram segítségével a kor legelterjedtebb grafikus hardverén, a 3dfx Voodoo 1 chipset-tel szerelt videókártyákon gond nélkül futott. A GLQuake nevéből adódóan az OpenGl 3D API-ját használta, a grafikus kártya raszterizált, így nem terhelte ezzel a CPU-t. A GLQuake olyan további effekteket produkált a VQuake-hez képest, mint a tükröződések, átlátszó vízfelület, kezdetleges árnyékok. *(21. ábra)*



21. ábra Quake szoftveres rendereléssel és a GLOuake

A Quake, VQuake és a GLQuake egyaránt DOS operációs rendszer alatt futottak, de fel voltak készítve Windows 95 használatára is. Csupán a Windows NT alapú operációs rendszereken nem működtek. Mivel a Windows operációs rendszerek már széles körben ismertek és használtak voltak a kilencvenes évek második felében, az id Software érdeke volt, hogy megírják a WinQuake-et, amely már az összes akkori Windows rendszeren működött. A játék ezen verziója már használta a DirectDraw, DirectSound és DirectInput API-kat, de továbbra is OpenGL alapú maradt.

A Quake sem kerülhette el a végzetét, sorozat lett belőle. A második és harmadik részt az id Software fejlesztette, míg a Quake 4 (2005) már a Raven Software munkája (és a Doom 3 motorját használja).

Az 1997-es *Quake 2* csak nevében és a továbbfejlesztett Quake-engine tekintetében folytatása az első résznek. A fejlesztett engine már a robbanásokat is 3D-s objektumként jelenítette meg, részletesebb modelleket és kezdetleges vertex-animációt használt. A játék már megjelenésekor tartalmazott OpenGl támogatást a szoftveres renderelési opció mellett. Az engine egyes komponenseit dinamikus könyvtárakban tárolták, ezáltal a forráskód nyilvánosságra hozása (2001) után a rajongók könnyebben módosíthatták a grafikus motort úgy, hogy közben a fő komponensek érintetlenek (és nem utolsó sorban működőképesek) maradtak. A Quake 2 engine - az id Software korábbi FPS-eihez hasonlóan - is a BSP eljárást használta a virtuális világ létrehozására.

A *Quake 3 Arena* (1999) egyértelműen a többjátékos módra lett kihegyezve. Tartalmazott ugyan egyjátékos módot, de az nem szólt másról, mint végigküzdeni 22 deathmatch pályát, gépi ellenfelekkel. A játék az előző rész motorját használta, de a kód túlnyomó részét át- vagy újraírták. Ezúttal szoftveres renderelési támogatás nem volt, a játék OpenGl kompatibilis grafikus gyorsítókártyát igényelt. Az engine a karaktermodelleket goraud-shading eljárással világította meg és árnyalta, míg a pályák megvilágítási módját a játékos opcionálisan megadhatta (lightmap ill. goraud-shading). Színes fényeket használt, melyek a modelleken is látszottak (a maga idejében az ilyen minőségű megvilágítás nagyon is előrehaladott technikának számított). A Quake 3-engine háromféle árnyékot volt képes megjeleníteni: az első, egy - a szélei felé halványuló - fekete kör a modell lábainál, míg a másik kettő vetett árnyék a modellnek megfelelő alakkal. A különbség az utóbbi kettő között, hogy az egyik egyszerű, átlátszatlan fekete árnyék, míg a másik áttetsző fekete (depth-pass stencil shadow). A Quake 3 már használta a magas szintű shader nyelvet, ködöt is meg tudott jeleníteni és elsőként alkalmazta az ívelt felületeket. Vertex alapú animációja segítségével mozgatta a modelleket, külön animációt használva a fejre, torzóra és a lábakra.

22. ábra Quake 3

A Quake legalább olyan fontos állomása a számítógépes játékok történelmének, mint a Doom. A GLQuake-kel indult a 3dfx Voodoo chipset-es videókártyáinak térhódítása, később

szinte minden valamire való háromdimenziós játékba beépítették ezen hardverek támogatását. A mai napig játszanak a Quake-kel (vagy folytatásaival), többjátékos módja annyira népszerű, hogy világbajnokságokat rendeznek a rajongóknak. Nem kevés cég megvásárolta valamelyik Quake epizód motorját, amit aztán továbbfejlesztve saját játékának alapjaként használt. *(Függelék, 54-55. ábra)*

4.3 Doom 3

1999 novemberében (a Quake 3 kiadását követően) Carmack kutatásokat kezdett végezni egy új 3D-s engine kifejlesztése érdekében. Kezdetben a Quake 3 engine-jére támaszkodott és csak a grafikával foglalkozó részt cserélte le, így sokkal könnyebb dolga volt az első lépések megtételekor. Ez lehetővé tette azt is, hogy Carmack több, különféle megközelítésre épülő teszt-engine megírásával foglalkozhasson és így a gyakorlatban legjobban működő technológiát választhassa. Mint utólag kiderült, a shadow volume számítása bár korrekt, de elég költséges az egész játéktérre vonatkozóan és hiányosságai (pl. alfa teszttel nem működik) miatt nem a legjobb választás. A kutatás másik fontos területe az emberi látás volt, ennek jellegzetességeit kiismerve lehetett meghatározni az új engine-nel kapcsolatos legfontosabb elvárásokat és irányelveket.

4.3.1 Célkitűzések

Az első célkitűzés a megvilágítási rendszer egységesítése volt. Ezidáig két fő eljárás állt rendelkezésre: a *vertex alapú árnyalás* lehetővé tette a fényviszonyok változásának követését, viszont csak részletesebb modelleken mutatott jól. Így általában csak a játékokban szereplő karaktereken használták, valamint a mozdítható tárgyakon, járműveken. A másik megoldást a *lightmap*-ek jelentették: egy második textúra minden egyes felülethez, amely a megvilágítottságot tartalmazza. Ezzel a módszerrel a vetett árnyékokat is meg lehetett jeleníteni, ugyanakkor a textúrákat valós időben frissíteni szinte lehetetlen feladat volt, hiszen ennek a számítási igénye a jelenet komplexitásával arányosan növekszik. A legtöbb engine valamilyen hibrid megoldást használ: a pályák fényellátottságát lightmap-ekkel oldják meg, a karakterek és tárgyak pedig vertex alapú megvilágítást kapnak. A második gond összefüggött a lightmap-ek használatával, ezek leszámoltatása, valamint az engine sebességét jelentősen

megnövelő láthatósági információk eltárolása ugyanis jelentős feldolgozási idővel járt (az egyszerű PC-ken ez napokban volt mérhető). Carmack végül olyan megoldást dolgozott ki, amellyel egyszerre oldhatta meg mindkét kérdést. A Doom 3 engine legfontosabb újítása tehát az, hogy a megvilágítás szempontjából minden felület egyenrangú – a pálya minden eleme, az összes tárgy és karakter ugyanazokat az eljárásokat használja. Emellett a játéktér teljesen dinamikus, bármikor elmozdítható egy komplett fal, vagy szoba is.

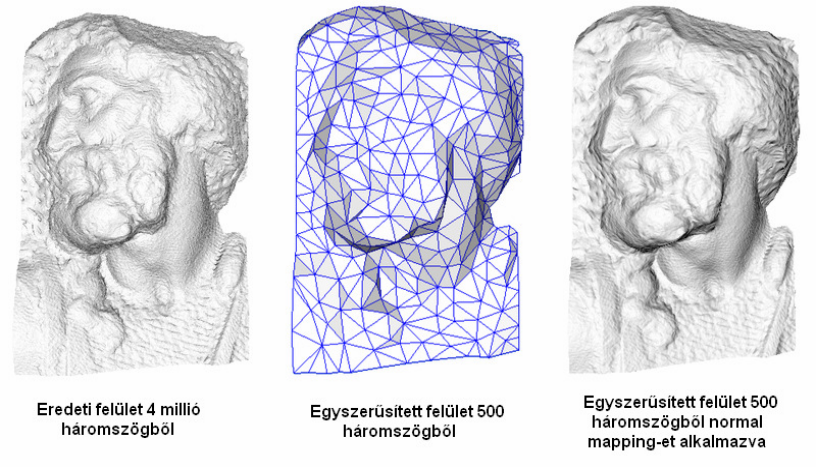
4.3.2 Fények és árnyékok

A fényviszonyok megjelenítéséhez a Doom 3 engine stencil-es shadow volume-okat használ. Egy-egy ilyen idom lényegében az adott fényforrás árnyékában lévő terek összességét jelenti; nagyon egyszerű példával élve egy sík négyszög shadow volume-ja egy csonka gúla. Ezeket az idomokat az engine valós időben képes létrehozni a jelenetben található tárgyakból, így a megvilágítás és az árnyékok teljesen valósághűek és szabadon változhatnak. A shadow volume-ok generálása viszont már elméletben sem egyszerű feladat: első lépésben a fényforrás szemszögéből nézve azonosítani kell az árnyékvető tárgyakat és meg kell keresni azoknak a körvonalait. Ezeket a sziluetteket a térben a fényforrástól eltolva jönnek létre a shadow volume-ok. A gyors és megbízhatóan működő algoritmus kidolgozását tovább nehezítik olyan korlátozások, hogy például ez az idom nem nyúlhat a végtelenségbe; bár szerencsére minden fényforrás hatása elenyészik egy bizonyos távolságon túl. (Az infinite shadow volume ezt a problémát kiküszöböli, itt a *far clip plane* úgymond a végtelenben van). Az eljárás másik fontos jellemzője pedig az, hogy a tárgyak poligonszámával arányos a sebessége – tehát az elviselhető sebességhez igen szűkmarkúan kell adagolni a poligonokat és nincs lehetőség sem a *Truform*, sem a *displacement mapping* használatára. Ezeket ugyanis a hardware számolja, a Doom3-ban pedig minden vertexet a CPU transzformál, hogy a shadow volume-hoz meghatározza a poligonokat. További hátrányt jelent az, hogy az árnyékok szélei mindig pengeélesek, ami egyáltalán nem valósághű. A soft shadow ezt a problémát hivatott kiküszöbölni; egyik módszere a shadow volumera épül, de annál még költségesebb.

4.3.3 Pályák és karakterek

Mivel a fények szempontjából minden tárgy egyenrangúnak számít (legyen az a pálya része, a játékos fegyvert markoló keze, vagy éppen egy szembejövő ellenfél), emiatt a vertex

alapú árnyalás alkalmazását Carmack kezdettől fogva elvetette és inkább a *pixelenkénti árnyalás*sal kezdett el komolyan foglalkozni. Ezzel a módszerrel viszonylag alacsony poligonszámú tárgyak esetében is látványos eredményeket lehet elérni, ráadásul lehetővé tette a *bump mapping* használatát. Itt jött egy újabb zseniális megoldás Carmack-tól: a bump-hoz szükséges *normal map* segítségével a shadow volume-ok miatt kényszerűen alacsony poligonszámú (lowpoly) modellek megjelenítését jelentősen fel lehet javítani. Ehhez minden egyes tárgynak és karakternek el kellett készíteni egy igen részletes, akár 5-600 ezer poligonból (!) álló változatát, majd erről a részletes felületről mentették el az információkat egy textúrába és ezt a nagyon részletes textúrát „húzták” rá a lowpoly modellre. Az eredmény a gyakorlatban a várakozásoknál is jobban néz ki, ugyanakkor vannak hátrányai is. Ezek közül a legnagyobb az, hogy a tárgyak körvonalai szögletesek maradnak, hisz a hasonló jellegű *displacement mapping*-gel ellentétben ez a módszer nem növeli meg a geometria részletességét. A másik, járulékos nehézséget pedig az jelenti, hogy minden tárgyat kétszer is el kell készíteni: egy minél optimálisabb lowpoly modellre és egy nagyon részletes, sok munkaórát igénylő változatra is szükség van.

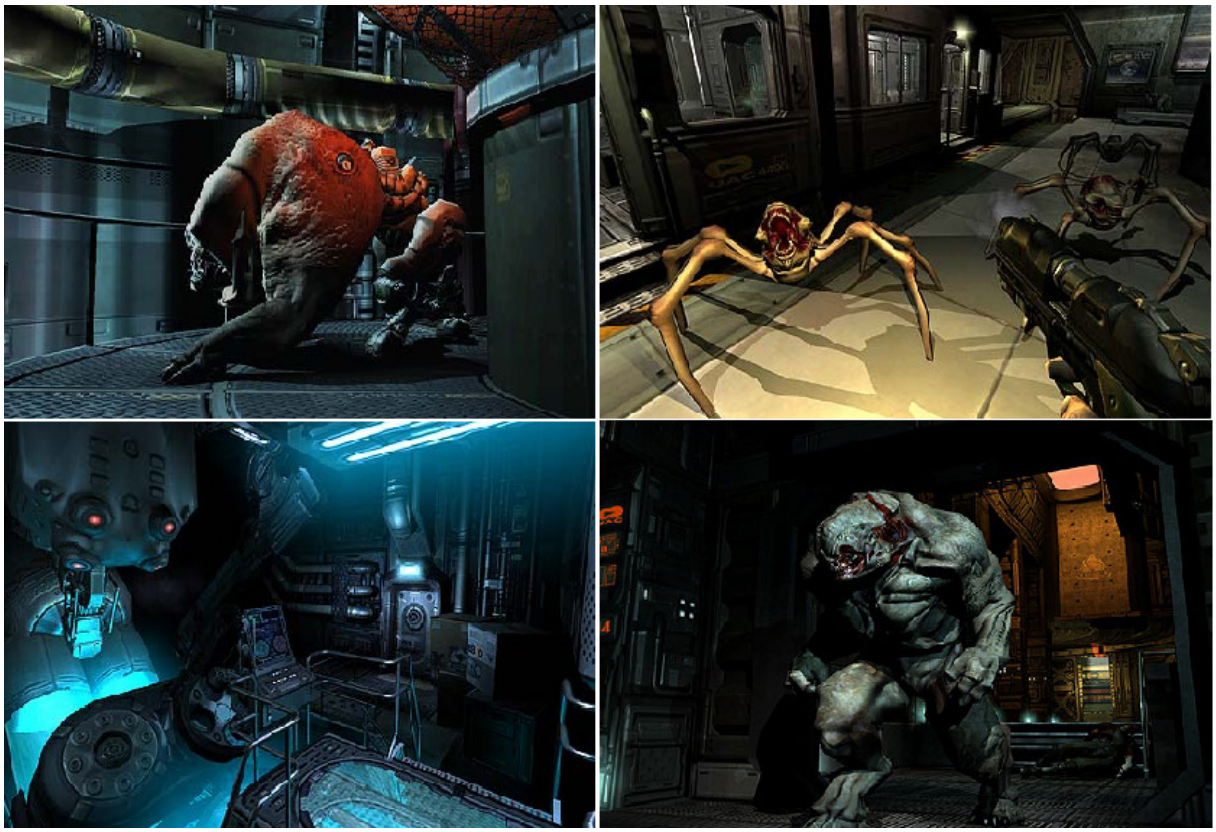


4.3.4 Az engine működési elve (menetek)

Több menetben rajzolja ki a 3D-s jelenetet, az egyes menetek tartalma a frame bufferben kerül összekombinálásra. Első menetben a textúrák és fények nélkül számolja ki a sima poligonokat, valamint tölti fel a *Z-buffer*t - ennek a tartalmát a további menetek már nem fogják felülírni. Ez az első kör viszonylag gyors és kevés memóriasáv-szélességet igényel a textúrák hiánya miatt. Ezután minden egyes fényforrásra megismétli a következő két menetet: először kiszámítja a shadow volume-okat és a renderelés során a *stencil buffer* tartalmát megnöveli azoknál a pixeleknél, amelyek árnyékba kerültek. Utána „ráfesti” a fény hatását a tárgyakra, de csak azokra a pixelekre, amelyeknél a stencil buffer értéke 0. Ebben a

menetben kerülnek fel a textúrák és a bump mapping is. A stenciles menet az egymást igen gyakran átfedő shadow volume-ok miatt igen sok poligonból állhat, amelyek nagy része ráadásul takarásban van. Ezért aztán kifejezetten nagy szerep jut az optimalizálásoknak, a nem látható felületek kiiktatásának – bár lényegében az egész stenciles menet láthatatlan, hiszen a frame buffer tartalmát nem változtatja meg. A textúrák hiánya miatt ez sem igényel nagy sávszélességet, viszont nagymértékben fogyaszthatja a *fill rate*-et. Fordított a helyzet a „fényező” menetekkel, ahol ugyanis az engine pixelenként 6-7 textúrát igényel. [5]

A fentiekből következik, hogy a Doom 3-nak igen erős grafikus hardverre van szüksége a gördülékeny futáshoz. Az átlagos rendszerkövetelményként meghatározott GeForce3 szintű videokártya 1999-ben még ijesztőnek tűnt, de a játék megjelenésekor (2004) már a GeForce4 is csak középkategóriának számított.



23. ábra Doom 3

4.4 FarCry

A CryTek fejlesztőinek FarCry című játékában többféle árnyékvetést is kombinálnak, kültereken a shadow map-et, belterekben a shadow volume-ot. Mivel a Doom 3 előtt jelent meg, így technikailag ez a meghatározóbb – bár a közmegítélésben nem kapott akkora elismertséget, mint amekkora a Doom 3-at megelőzte. A játék érdekessége, hogy a grafikus motorja - a CryEngine - eredetileg az nVidia Geforce 3 videókártya képességeit bemutató technológiai demó volt és csak később döntöttek úgy, hogy játékengine-ként használják. A CryEngine kiválóan ötvözi az összes olyan technikát, melyeket külön-külön előtte máshol már alkalmaztak (környezetet visszatükröző vízfelület, polybump normal mapping, shadow map, shadow volume) *(24. ábra)*. Az 1.2-es verziója már támogatja a 2.0-ás vertex- és pixel shader-eket, az 1.3-as CryEngine pedig már a shader model 3.0-át is használja, ami tovább javítja az amúgy sem mindennapi látványt. *(25. ábra)*

Talán a kiadást megelőző kevés reklámnak (és a már küszöbön álló Doom 3-nak) is köszönhető, hogy a FarCry viszonylag hamar feledésbe merült, a játékengine-t is csupán egy cég licenszelte – persze valószínűleg nem is adják olcsón.

24. ábra FarCry

25. ábra Pixel shader 2.0 és 3.0 a FarCry-ban

4.5 Crysis

A most készülő Crysis ezzel ellentétben már vezető szerepet játszik, ami többek között annak is köszönhető, hogy maximálisan kihasználja a DirectX 10-ben rejlő lehetőségeket. A játék már a CryEngine 2-őt használja; nézzük meg miket kínál a grafika szempontjából:

- Renderelés: kültéri és beltéri technológiák együttes alkalmazása, DirectX 8/9/10 támogatása
- Animációs rendszer: csontvázalapú animáció kombinálása előre felvett mozdulatokkal, különös figyelmet fordítva az emberi animációra (szemek mozgása, egyenetlen felületeken való haladás, arcmimika, természetes átmenetek az egyes animációs fázisok között – pl. a gyalogló karakter fokozatosan kezd el futni)
- Shader-ek: valós idejű pixelenkénti árnyalás, érdes tükröződő felületek, fénytörés, volumetrikus ragyogás effekt, animált textúrák, csillogó felületek
- Terep: fejlett magasságtérkép-rendszert használ poligonredukcióval a hatalmas, realisztikus környezet létrehozására, a látótávolság akár 2 km is lehet

- Voxel objektumok: segítségével olyan geometriákat lehet létrehozni, amelyeket a magasságtérkép-rendszer nem támogat; ilyenek pl. a barlangok, sziklafalak, kanyonok; a voxelek szerkesztése legalább olyan egyszerű, mint a magasságtérképé és ráadásul a renderelése gyors
- Fények és árnyékok: előre kiszámolt árnyékok kombinálása kiváló minőségű valós idejű vetett árnyékokkal - dinamikus fény-árnyék a külső tereken *(27. ábra)*; nagyfelbontású, korrekt perspektívájú és volumetrikus „puha-árnyékok” - realisztikus belső terek
- Köd: volumetrikus, látótávolságot befolyásoló – atmoszférát, hangulatot teremt
- Polybump 2: a FarCry-ban alkalmazott normal mapping eljárás módosított változata, rendkívül részletes felületek létrehozására alkalmas, támogatja a displacement mapping-et *(26. ábra)*

A Crysis a közeljövő egyik legtöbb, elsősorban grafikai újdonságot felvonultató játéka lesz (a másik a szintén még fejlesztés alatt álló Unreal 3 *(Függelék, 57. ábra)*). Az alábbi képek zavarba ejtően részletesek, ám igazán mozgás közben lehetne látni, hogy a karakterek mennyire „emberinek” hatnak. A CryTek tisztában van mindezzel és néhány hete áruba bocsátotta a CryEngine 2 motort, így csak idő kérdése, hogy egy hasonló grafikai színvonalú játékot jelent be valamelyik fejlesztőcég.

26. ábra Karakterek a Crysis c. játékból

27. ábra Kültér a Crysis c. játékban

Íme a látvány 2007-ben, amit a programozók napjaink grafikus hardvereinek támogatásával a monitorunkra képesek varázsolni. Ez – valljuk be őszintén – már nincs messze attól, hogy összekeverjük a valósággal. *(Függelék, 58. ábra)*

5. Egyedi technológiák, érdekességek a számítógépes játékok történetéből

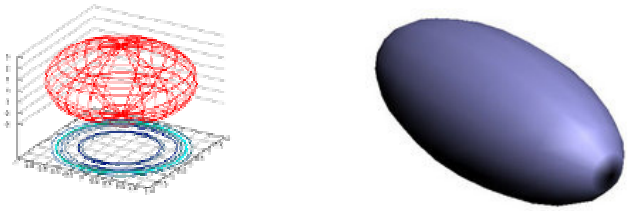
Külön fejezetet szenteltem néhány olyan különlegesség bemutatására, amely játékok megjelenésük idejében feltűnést keltettek egyedi grafikai megoldásaikkal, olyan technológiát alkalmaztak, amit előttük még egy programban sem láthatott a nagyközönség.

5.1 Ecstatica

Az 1994-ben IBM PC-re készített Ecstatica ugyan játékmenetbeli újításokat nem hozott, hiszen ebből a szempontból a klasszikus Alone in the Dark-ra erősen hasonlított, mégis a játék grafikáját tekintve mindenféleképpen a különlegesség kategóriájába tartozik.

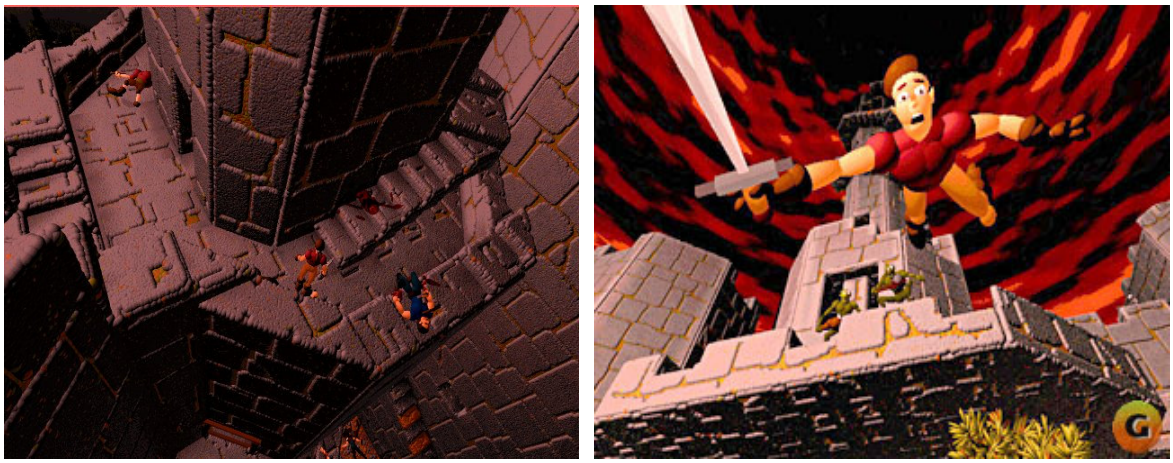
Az Ecstatica világát a 256 színt használó, ellipszoid *(28. ábra)* technológián alapuló grafikus engine rajzolta a képernyőnkre. Ez azt jelenti, hogy a játékban minden karakter (legyen az a főhős, vagy bármelyik ellenfél) ellipszoidok láncolatából állt, ennek ellenére mégsem néztek ki furcsán, sőt, kifejezetten élethűnek hatottak és a nagy előd (Alone in the Dark) poligonokból felépülő modelljeihez képest szebbek is voltak. A karakteranimáció megvalósítása szintén dicséretet érdemel, a karakterek életszerűen mozogtak, támadtak (a főhős tudott lopakodni, gyalogolni, futni, vívni és ha súlyosan megsebesítették, vonszolta magát). A környezetet, a növényzetet szintén ellipszoidok sokasága alkotta, ami addig soha nem tapasztalt, egyedi látványt kölcsönzött a játéknak *(29. ábra).* A kameranézet rögzített volt, hősünkkel képernyőről képernyőre vándorolhattunk ebben az érdekes világban. Pontosan a képernyőként változó fix kamera tette nehezen játszhatóvá az Ecstaticát. (képzeljük el, ahogy menekülünk egy ellenfél elől, az új képernyőn viszont már szemből látjuk magunkat, de ahogy megszoknánk, rögtön madártávlatból mutatják hősünket…)

Mindezek ellenére 1997-ben elkészült a folytatás, ami már SVGA-ban pompázott. Az ellipszoid technikát meghagyták *(30. ábra)*, de az olykor frusztráló játékmeneten sajnos nem javítottak.



28. ábra Ellipszoid huzalvázas és 3D renderelt képe

29. ábra Ecstatica



30. ábra Ecstatica 2

5.2 Outcast

Egy másik említésre méltó alkotás az 1999-ben debütáló Outcast című akció-kaland játék, a belga illetőségű Appeal fejlesztő cég munkája. Grafikus engine-je a vektorgrafikát

tárgyaló fejezetben említett voxeleket alkalmazta, igaz nem elsőként (a NovaLogic már a ’90-es évek közepén ezzel a technikával készítette a Comanche és a Delta Force sorozatokat), de kétségkívül ez a legszebb az összes hasonló technológiával felvértezett játék közül.

A voxel-engine tisztán szoftveralapú, nincs szüksége hardveres gyorsításra a grafikus kártya részéről. Azért választották ezt a módszert, mert így az íves felületek renderelése nem lassítja le a játék sebességét (szakzsargonnal élve: „nem szaggat a játék”), ráadásul a voxelekkel sokkal részletesebb, életszerűbb tájakat tudtak megjeleníteni. Ugyanilyen kidolgozottságú környezetet poligon alapú engine-nel még hardveres gyorsítással támogatva sem lehetett volna megfelelő sebességgel mozgatni a kor számítógépein. A voxelek köztes megoldást jelentettek a szoftveres és a hardveres renderelés között, kiváló minőségű képet tudtak produkálni anélkül, hogy szükség lett volna bármilyen különleges grafikus hardverre. Végeredményben korát meghaladó látvánnyal ajándékozta meg a játékosokat az Outcast, köszönhetően az olyan effektek használatának (hangsúlyozom: szoftveresen!), mint a depth-of-field, bump mapping, anti-aliasing *(31. ábra)*, melyek a kor hardveres gyorsítást használó grafikus kártyáit is nagy valószínűséggel két vállra fektették volna. Azonban mindennek ára van, a voxel-engine szépségének is: mivel nem használ hardveres gyorsítást (nem kell a grafikus kártyára költeni), így kizárólag a CPU-ra támaszkodik. Ahhoz, hogy a legnagyobb felbontásban (ami csupán 512x384), minden effektet bekapcsolva élvezhetően fusson, a kor leggyorsabb, 5-600 MHz-es Pentium III-as processzoraira volt szükség és nem ártott, ha az „átlagos” felhasználó számítógépében 128 Mbyte memória lapult. Ezt persze még a kilencvenes évek végén sem sokan engedhették meg maguknak, így az Outcast vegyes érzelmeket keltett a játékosok körében: akiknek volt megfelelő gépük hozzá, azok éltették, akiknek nem, azok csalódottan letörölték a játékot… majd néhány év múlva (mikor a Pentium III-at már fillérekért adták) kipróbálták teljes pompájában és akkor már elismerően bólintottak.

Az Appeal folytatni szerette volna a játékot, de a megosztott sikernek köszönhetően annyira kis bevételt hozott az első rész, hogy kénytelenek voltak leállítani a folytatás fejlesztését. Próbálkoztak a kiadótól anyagi támogatást szerezni, de kérésük süket fülekre talált. A cég tönkrement, az Outcast 2-re pedig csak néhány kép emlékeztet. *(32. ábra)*

31. ábra Outcast

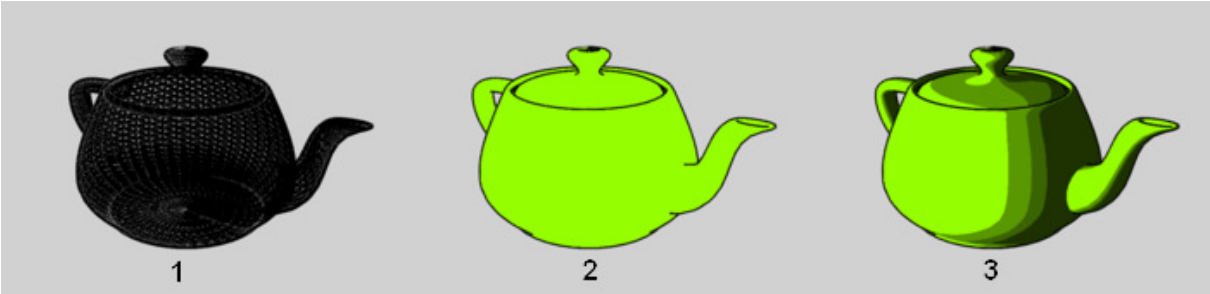
32. ábra Az Outcast 2 már poligonokat használt (volna) voxelek helyett

5.2 XIII

Végül, de nem utolsó sorban, nézzünk meg a Ubisoft cég XIII című játékát, ami egy olyan technikát alkalmaz, aminek nem az a célja, hogy realisztikus képet állítson elő, hanem pont az ellenkezője, a rajzfilm-, képregényszerű megjelenítés.

A módszert cel-shading-nek nevezik és első alkalmazása nem is a PC –hez, hanem a játékkonzolok világához köthető. Ez leginkább a konzolok (Dreamcast, Playstation) fix hardverfelépítése miatt van így, ugyanis a cel-shading szerény hardverkövetelmények mellett látványos eredményt képes előállítani. A cel-shading elnevezést a hagyományos kétdimenziós animációk készítésekor használatos celluloid lapok, az ún. cel-ek után kapta az eljárás. Míg a végeredmény olyan egyszerűnek hat, mintha csak kézzel rajzolták volna, maga a módszer eléggé bonyolult. A cel-shading folyamat egy ugyanolyan 3D-s modellel kezdődik, mint minden más renderelő eljárás esetén. A különbség a modell képernyőre való kirajzolásakor

fog kitűnni. Az engine az objektum színeinek csak néhány árnyalatát használja, ezzel „lapos” (flat) kinézetet ad neki. A modell kontúrvonalainak előállításához (hogy úgy tűnjön, mintha tussal lenne rajzolva) a backface-culling eljárás inverzét használja, azaz szándékosan a nem látható (takarásban lévő) háromszögeket jeleníti meg és húzza ki fekete vonalakkal. Ezt többször is megismétli, hogy a kontúrvonalak vastagok legyenek a kész képen. Így előáll az objektum fekete-árnyalatú sziluettje. Ezután a backface-culling-ot már a hagyományos értelemben használja, hogy elkészítse az árnyalást és „felhúzza” az objektum esetleges textúráit. Végül a kép a Z-buffer-en keresztül áll össze.



33. ábra 1. A nem látható felületek vastag vonallal kihúzva (sziluett), 2. Egyszerű textúra az objektumon, 3. Árnyalás (shading)

A XIII volt az első olyan játék PC-re, ami ezt a technológiát használta. 2003-ban jelent meg, az azonos című francia képregény adaptációja, tehát kézenfekvő volt, hogy a cel-shading a legmegfelelőbb a képi világ megvalósításához. Ezt sikerült is nagyszerűen kiviteleznie a Ubisoft programozóinak, a játék hűen adta vissza a képregény hangulatát. Az átvezető animációk is az engine-nel készültek, folyamatosan fenntartva azt a hatást, hogy a játékos egy képregény lapjain kalandozik. Egyaránt használta a „kép a képben” effektet és az onomatopoeia-kat (hangutánzó szavak, szócsoportok, melyek függnek a hang forrásától, pl. click, tap, bang, ka-boom), tovább erősítve a képregény hangulatot. *(34. ábra)* A játékot a szaklapok pozitívan értékelték, mégis az elvártnál jóval kisebb bevételt hozott a Ubisoft konyhájára. [3]

34. ábra XIII

6. Technikai háttér

Az előző fejezetekben tárgyalt módszerek megértéséhez elengedhetetlen a két- és háromdimenziós grafika egyes fogalmainak ismertetése, valamint annak bemutatása, hogy a korszak játékfejlesztői milyen technológiákra építhettek és azok milyen új megoldásokat kínáltak számukra.

6.1 Kétdimenziós grafikai fogalmak

6.1.1 Vektorgrafika

Ez az egyik lehetősége annak, hogy a programozó képet állítson elő a rasztergrafikus képernyőn. Gyakorlatilag minden korszerű térbeli renderelő eljárás a síkbeli vektorgrafika valamilyen kiterjesztésén alapul. Háromdimenziós (korábban 2D-s) lebegőpontos koordináta-rendszert használ, lehetővé teszi a geometriai pontosságú szerkesztést és transzformációkat. (Alkalmas a mérnöki és a tudományos munka támogatására.) Absztrakt modelltérbeli tárgyakkal dolgozik; ezek olyan önálló objektumok (entitások), melyekkel műveleteket lehet végezni a képernyőn való megjelenítéstől függetlenül is. A grafikus objektumokat adatbázisban tárolják, lehetővé téve az egyes testek, tárgyak modelljeinek egyedi visszakeresését és az ezek közötti kapcsolatok rögzítését és kimutathatóságát.

A vektorgrafikus modellezés fajtái:

Huzalvázas modellezés: (Nem teljes értékű geometriai modellezés – nem tartalmazza a valós test leíráshoz szükséges összes geometriai és csatlakoztatási (topológiai) információt.) 3D-s geometriai alakzatokat csúcsaival és éleivel jellemzi, a modell csak a csúcsokat és az ezekhez rendelt összekötő éleket tartalmazza. Előnye, hogy számítógépes megvalósításuk algoritmusigénye a többi geometriai módszernél lényegesen kisebb. Problémája: egy huzalváz-modellnek több test is megfelelhet. Nem mindig tehető különbség a tömör és üreges között, és a testet határoló felületek görbültségét sem tudjuk kezelni.

Palástmodellezés (b-rep = boundary - representation): A geometriai objektumokat a vektorgrafikus modelltérben határoló felületeikkel (beleértve e felületek csatlakoztatására vonatkozó adatokat is) jellemezzük. Geometriai jellemzői szerinti osztályozása:

1. ha a palástot képező lapok síkbeli sokszögekből állnak, akkor poliéder modelleket kapunk,
2. ha a palástot képező lapok változó görbületű felületfoltok is lehetnek, akkor valósághű palástmodellezésről beszélünk.

Palástmodelleket lépésenkénti szerkesztéssel is létrehozhatunk, a test felületeit egyenként definiáljuk a térbeli csatlakoztatási lehetőségeinek függvényében. Ezek elemi lépéseit Euler-operátoroknak nevezik: pl.: - képezz csomópontot, élet és palástot,
- kapcsolj ki (törölj) csomópontot és élet.

Ezekkel az operátorokkal „konvex poliéderekhez hasonló" testek egyértelműen létrehozhatók. Elterjedten a különböző CAD rendszerekben alkalmazzák.

Testmodellezés: Feltételezzük, hogy a modellezendő objektumok olyan merev testek (a modelltérben mozgatásukkor alakjukat nem változtatják), melyek a palástjukkal határolt teret teljesen kitöltik (merev testek). Két elterjedt testmodellezési módszer:

1. elemi testekkel és ezek közötti szabályos halmazműveletekkel való modellezés,
2. a testek elemi sejtekből való felépítése.

Véges számú elemitest primitívből kiinduló és a modellt a metszet, egyesítés, kivonás és ragasztás halmazműveletek egymás utáni felhasználásával megkonstruáló modellezési módszert konstruktív tömör testmodellezésnek nevezzük (Construktív Solid Geometry = CSG)

A CSG két alkotóeleme:

1. kiinduló tömör testprimitívek készlete, (3D-s primitívekből áll: hasáb, gúla, henger, kúp, gömb).
2. a megengedett halmazalgebrai műveletek eszközkészlete.

A konstruktív tömör testmodellezést alkalmazó vektorgrafikus szoftverek egy speciális parancsnyelvet használnak, mely a műveletek többségét lehetővé teszi:

-konkrét primitív példány létrehozása,

-objektum másolása, törlése, transzformálása,

-objektumok halmazalgebrai metszete, egyesítése, kivonása, összeragasztása.

A CSG-vel létrehozott modellek adatstruktúrájára a bináris fa gráf jellemző, ágcsomópontjai a halmazalgebrai műveletek, levelei a műveletekben résztvevő testek.
A CSG a gyakorlatban jól használható, mert a műszaki tervezés során szükséges testek döntő része előállítható néhány egyszerű geometriai test (pl.: hasáb és henger) megfelelő kombinációjából.

Térfelosztással való modellezés – térfogatmodellezés elemi sejtekkel: Térfogatmodellezésnél egy tömör tárgyat több egymáshoz csatlakozó, de egymást nem metsző kisebb tömör tárgyra, azaz sejtekre bontunk fel. Az elterjedt modellezési módszerek a sejtek két típusát kezelik:

1. a sejtek azonos típusú alakzatok (pl.: hasábok), de méretük egy paramétertől függően változhat,
2. a sejtek azonos típusú és méretű alakzatok, ekkor ezeket (a képponthoz, azaz pixelhez hasonlóan) *voxel*nek (volume pixel) nevezzük.

Azonos formátumú és méretű voxelekkel való kitöltése a modellezendő testnek - amennyiben a voxeleket elegendően kisméretűre választjuk - a test relatíve pontos leírását eredményezi. A leggyakoribb voxeltípus a kocka. A modellezendő objektumokat a voxelekkel úgy írjuk le, hogy minden egyes, a testhez teljes egészében, vagy csak részlegesen hozzátartozó voxel adatait hozzárendeljük a testhez. Ez a számítógéptől nagy tárolókapacitást és processzorteljesítményt igényel.

6.1.2 Rasztergrafika

Alapeleme a pixel, ami a picture element, szó szerint „képelem” kifejezés rövidítése, egy grafikus kép egyetlen képpontját jelöli. Egy rasztergrafikus kép, vagy bitmap, ezen pixelekből épül föl és jelenik meg a monitoron. Ha a képpontok elég sűrűn helyezkednek el, szemünkkel nem pontokat, hanem összefüggő képet látunk (ennek ellenére a képernyőn két látszólag egymást egy pontban metsző egyenesnek egy vagy több közös képpontja is lehet). A geometria szabályai szerint szerkesztésre nincs lehetőség, egy rasztergrafikus kép tartalma csak a teljes kép felülírásával módosítható. A megjeleníthető színek mennyisége alapján az alábbi raszteres képtípusokat különböztetjük meg:

- bittérképes képek (bitmapped image),

Minden egyes képponthoz tartozó színinformációkat 1 biten (1=fekete, 0=fehér) kódoljuk, ezek a képek fekete-fehérek.

- szürkeárnyalatú képek (grayscale image),

Képpontonként 8 biten kódolva – 256 féle fekete-fehér átmeneti színt tartalmazhat.

- színpalettával indexelt képek (indexed color image),

Pixeljeinkhez egy színindex értéket rendelünk hozzá, mely 256 elemű színtáblázatra hivatkozik, melyet *paletta*-nak nevezünk. A színpaletta minden 2, 4, vagy 8 bites indexe egy konkrét színárnyalatot határoz meg egy pixel számára (sőt, a TGA képfájlok támogatják a 16, 24 bites indexelést is). Képenként külön színpalettával indexálhatunk.

- valódi színezetű képek (true color image).

A színtér alapszíneinek megfelelő színcsatornánként adjuk meg az alapszínek intenzitását. Ez RGB vagy CMY színtér esetén 3x8 = 24 bit, CMYK színtér esetén 4x8 = 32 bit megadását jelenti (pl.: RGB alapszín intenzitások keverésével 2^n (n=24), azaz több mint 16 millió (16 777 216) színárnyalatot tudunk megkülönböztetni). A true color-hoz hasonló a hi color kép, ahol 15 illetve 16 biten tároljuk a pixeleket (R5G5B5 vagy R5G6B5).

Léteznek úgynevezett alfás képek is, ez az (általában) átlátszósági információ lehet akár a 15 bites hi color pixelben a 16-ik bit, a 24 bites true color pixelben a 4-ik byte, vagy akár a 8 bites grayscale kép önmaga.

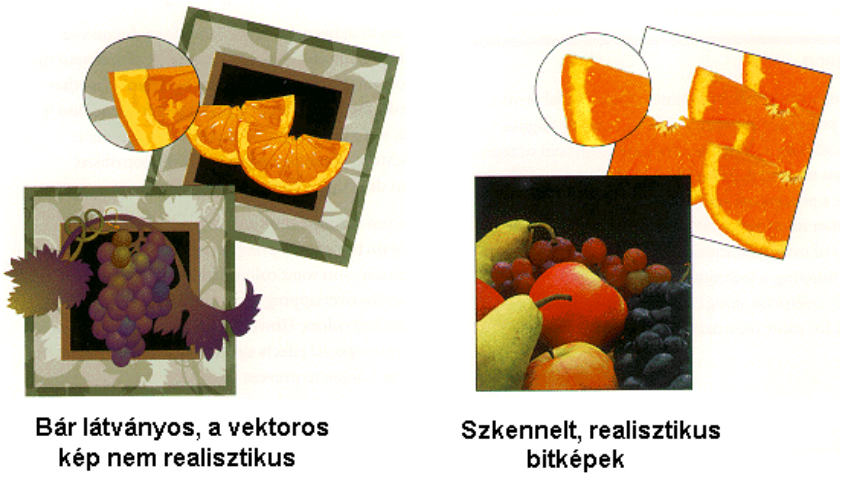
Monitor és videókártya típusok:

- 1980-ban CGA : 320 * 200-as felbontásnál 4 szín , 640*200-as felbontásnál 2 szín
- 1984-ben MDA/HERKULES : 720*348-as felbontásnál 2 szín
- 1984-ben EGA : 640*350-as felbontásnál 16 szín
- 1987-ben VGA : 640*480-as felbontásnál 16 szín, 320*200-as felbontásnál 256 szín
- 1990-ben SVGA VESA szabvány, true color, hi color módok megjelenése

6.1.3 A vektor- és rasztergrafika összehasonlítása

Mind a vektorgrafikának, mind a rasztergrafikának megvannak a maga előnyei és hátrányai. Egy vektoros kép korlátlan méretben nagyítható és kicsinyíthető minőségromlás

nélkül, míg a pixelgrafikus megoldás esetén pl. egy kör nagyított képe nem fog körnek kinézni, sőt a ráközelítéssel a vonalvastagság is megnő. Az egyes módszereknél lényeges összehasonlítási szempont a tárigény is. Mivel a képet leíró matematikai vektorok viszonylag egyszerűen leírhatók és a felhasznált színek számának sincs szerepe (minden objektumot ki lehet tölteni valamilyen színnel, nincs jelentősége annak, hogy milyen szín – ha egy rajzban csak fekete és fehér színeket használunk, akkor is ugyanannyi helyet foglal, mintha az összes objektumnak más színt adunk), ezért általában a vektoros rajzok sokkal kisebb méretűek, mint a velük összemérhető pixelgrafikus képek. Minél nagyobb felbontású egy pixelgrafikus kép, minél több színt használ, annál nagyobb lesz az állomány mérete (erre a problémára a képtömörítés jelent megoldást). Ugyanakkor egy vektoros kép soha nem lesz olyan részletes, fotorealisztikus, mint egy raszteres kép. *(35. ábra)*



35. ábra Vektoros és raszteres kép összehasonlítása

6.2 Háromdimenziós grafikai fogalmak

Alpha-blending: Átlátszó objektumok létrehozását szolgáló módszer. A textúrák tartalmazhatnak egy negyedik színcsatornát az RGB mellett. Ez a negyedik színcsatorna az alfa, amely arról tartalmaz információt, hogy a textúra az adott pontban mennyire átlátszó (a

0.0 érték a teljes átlátszóságot jelenti, míg az 1.0 ennek az ellentettjét). Tehát egy textúra átlátszóságát 0 és 1 között megadott alfa értékekkel szabályozhatjuk. Az alfa csatorna ezen felül használható olyan effektek előállításához is, mint például objektumok felületének tükröződése (reflectivity), fényvisszaverődése (specular intensity).

Fill rate: A grafikus kártya által másodpercenként előállított és megjelenített pixelek száma. Újabb grafikus kártyák esetében megapixel/másodpercről, sőt gigapixel/másodpercről beszélhetünk.

Higher Order Surface (HOS): Magasabbrendű felület. A professzionális grafikában már régóta használnak a poligonoknál jóval bonyolultabb geometriát; ilyenek például a NURBS felületek vagy a Bézier patch-ek. Ezeket a rendereléshez poligonokra kell felbontani, azaz tesszelálni kell. Nagy előnyük, hogy jóval kevesebb adattal írhatók le, mint az eredményül kapott poligonos geometria – ha a tesszeláció a videochip-en történik, akkor a busz terhelése jelentősen csökkenthető, ugyanakkor a poligonszám növelhető. Ezek a geometriák a játékfejlesztésben nem túl célszerűek, mert a grafikusoknak elég körülményes velük dolgozni.

Lightmap: Adott 3D engine-hez tartozó, felületek megvilágítási adatait tartalmazó struktúra. Mindig előre kiszámított és általában statikus objektumok esetén használják. A Quake volt az első játék, amely alkalmazta a látvány javítása érdekében.

Near/far clip plane: A clip(ing) plane azt határozza meg, hogy egy adott objektumnak mennyire közel (near), illetve mennyire távol (far) kell lennie ahhoz, hogy kikerüljön a kamera látómezejéből (tehát mikortól nem kell renderelni az objektumot).

Stencil buffer: Extra buffer, a Z-buffer (mélység-buffer) mellett. Ha a Z-buffer 24 bites, akkor a fennmaradó 8 bitet a stencil buffer foglalja el. Legegyszerűbb esetben a renderelést „határolja” be. Egy sokkal bonyolultabb alkalmazása kihasználja a stencil buffer és a Z-buffer közötti szoros kapcsolatot (például a stencil értékek automatikusan nőnek/csökkennek a mélység-teszten megbukó, illetve átmenő pixelek esetén). Leggyakrabban árnyék és tükröződések megjelenítésére használják a 3D-s alkalmazásokban. A Direct3D és az OpenGl egyaránt támogatja.

Textúra: A 3D-s modell kinézetét, felszínét meghatározó, arra ráfeszített többrétegű mintázat. A textúra sokféle formátumú lehet (akár monokróm is).

Vertex alapú- és pixelenkénti árnyalás: A két eljárás között a legfontosabb különbség a megvilágításhoz szükséges egyik információ, a felületre merőleges, ún. normálvektor kezelése. Vertexárnyaláskor a videókártya csak a tárgyak alkotópontjainál határozza meg ezt a vektort és utána az egyes poligonok felületén található pixelek renderelésekor interpolációval számítja ki a megvilágítottságot (ezt a folyamatot gyorsítja a T&L módszer „Lighting" része). A pixelenkénti árnyalás esetében viszont egy textúrából, a normal map-ből olvassa ki ezt a vektort és a szükséges számítások elvégzéséhez a pixel shader-t használja. Ez az eljárás tehát számításigényesebb, továbbá minden tárgyhoz szükség van további egy textúrára, viszont sokkal szebb eredményt ad.

Z-buffer: A 3D-s grafikában a kép mélységi (z) koordinátáinak kezelésére szolgál. Egyfajta megoldás az ún. láthatósági problémára, melynek lényege annak eldöntése, hogy a renderelt kép mely részei láthatóak és melyek nem. Ha egy objektumot renderelünk egy 3D-s kártyával, akkor a létrejött pixelek „mélysége" (z koordinátája) eltárolódik a Z-bufferben. Mátrix reprezentációt használ, melyben a képernyő minden egyes pixeléhez tartozik egy elem. Ha renderelés során egy objektum adott pixele a mátrixban már „foglalt" helyre kerülne, akkor a grafikus kártya összehasonlítja a két pixel mélységét, a megfigyelőhöz közelebbit választja és elmenti a Z-bufferbe, a régi értéket felülírva. Végeredményben a grafikus kártya helyesen jeleníti meg a képet: a közeli tárgyak takarják a hátrébb lévőket (ezt nevezik Z-culling-nak). Az átlátszatlan felületeknél ez jól működik, viszont az átlátszó poligonokat mélység szerint sorba kell rendezni, hogy a blendelés korrekt sorrendben történjen.

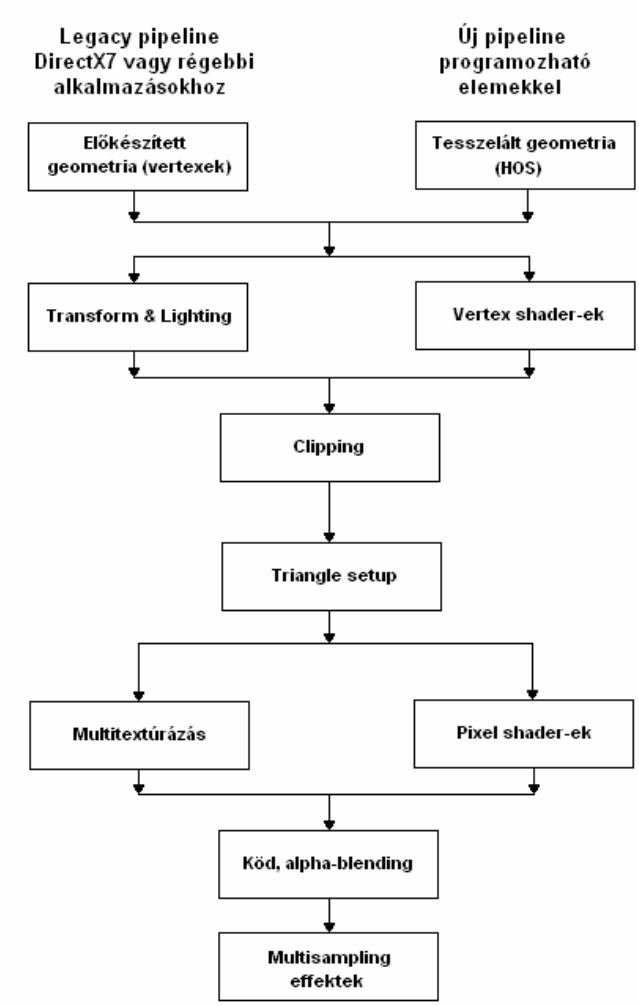
6.3 Programozói felületek (API)

6.3.1 DirectX

A Microsoft által készített programozói felület, amely azon felül, hogy kezeli a beviteli és hálózati eszközöket, lehetővé teszi, hogy a programok kihasználják a számítógép fejlett grafikus, 3D animációs és zenei képességeit. 1995-ben mutatták be először, mára

nélkülözhetetlenné vált multimédiás alkalmazások futtatásához Windows környezetben. A két- illetve háromdimenziós grafika megjelenítéséért a DirectDraw (2D) és a Direct3D nevezetű API-k felelősek. 2001-től összevonták ezeket, az új név DirectX Graphics lett. A Direct3D komponens valójában a DirectX 3.0-ás verziójától használható és az 5.0 magasságában kezdték ezt az API-t előtérbe helyezni a Glide rovására. Az első igazán látványos újítást (a vertex- és pixel shader-ek alkalmazását) a háromdimenziós grafikai megjelenítésben a 8.0-ás verzió hozta. Ahhoz, hogy lássuk a technikai különbségeket, vizsgáljuk meg a 7.0-ás verziót!

Először is nézzük meg, hogyan zajlik az adatok feldolgozása. A feldolgozás menete általában sok ismétlésből áll és a legtöbb részfolyamat ezen belül szintén sokszor kerül végrehajtásra. Ezért általában pipeline-ként (futószalagként) szokás kezelni *(36. ábra)*, amelyet két fő részre tagolnak: a geometriai feldolgozásra és a raszterizálásra.



36. ábra DirectX 8 grafikus pipeline

A geometria feldolgozásának első szakaszát (a játékos viselkedésére az egész 3D-s világnak reagálnia kell) általában a CPU végzi, bár manapság már léteznek olyan fizikai gyorsításra képes kártyák (Ageia PhysX), melyek a CPU helyett elvégzik a fizikai szimulációhoz szükséges számításokat. A következő szakasz a háromdimenziós geometria létrehozása vertexekből (pontokból). Ezeknek a vertexeknek a különböző tulajdonságai változnak a fent említett eseményektől függően és ezek a pontok határozzák meg azokat a poligonokat, azaz sokszögeket, amelyeket aztán a kész képen láthatunk (a pipeline-ban a poligonok háromszögekből épülnek föl). A vertexeket számhármasokként ábrázoljuk X, Y és Z értékek szerint, ahol X, Y, Z a háromdimenziós koordinátarendszerünk tengelyeit jelképezik, a három szám tehát az egyes vertexek pozícióját jelöli. Azonban a 3D-s grafikában többféle „tér" (és koordinátarendszer) létezik: van először is a tárgyak saját tere (object space), a 3D-s programok így tárolják (többek között a megfelelő pontosság érdekében) az objektumokat. Ahhoz, hogy további számításokat végezhessünk, a tárgy vertexeit át kell transzformálni, sorrendben először a világ 3D-s terébe (world space). Itt kerülhet sor például a tárgy megvilágítására, ami a legtöbb esetben a vertexek beszínezésével történik. Ezután, az emberi látásra jellemző perspektivikus hatás érdekében a vertexeket ismét transzformálni kell, mégpedig a kamera terébe (camera space). Végül a kiszámítandó képet az X és Y dimenziói által alkotott kétdimenziós térbe (image space) transzformálva, néhány további számítás után megkapjuk az egyes vertexek helyét a kész képen. Ha ezek a képen kívül vannak, úgy szét kell vágni a háromszögeket. Ezek után lehet hozzáfogni a vertexek által felépített poligonok raszterizálásához, azaz kiszínezéséhez. A fenti folyamat egyes fázisai általában ugyanazok minden egyes vertexre, így jó lehetőség kínálkozott a transzformációs és megvilágítási (Transform & Lighting) műveletek hardveres gyorsítására. Erre az első megvalósítás a DirectX 7.0-ás verziójában jelent meg, azonban az implementáció nem sikerült eléggé egyértelműre. A legfontosabb baj a fix felépítés volt: a programozó csak azokat az eljárásokat használhatta, amelyeket az API lehetővé tett, így például csak a Direct3D megvilágítási modelljének számításait lehetett gyorsítani.

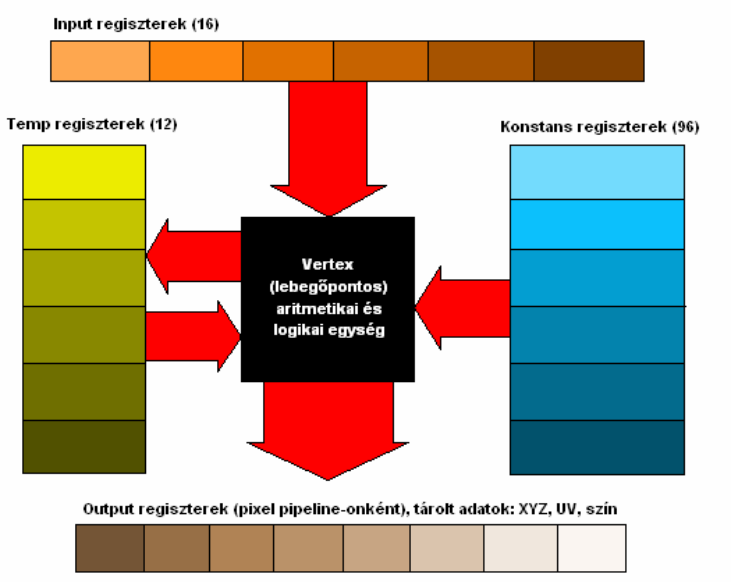
A 3D-s pipeline másik fele a raszterizálás, azaz a kép pixeleinek kiszámítása. Ez megint csak nagyon hasonlóan megy végbe minden egyes pixelre, így a hardveres támogatás szintén adja magát (erre az első jó példa a 3dfx cég Vooodoo1 nevű videókártyája volt). A pixel színének egyik fő komponense általában textúrából származik, a többi pedig a háromdimenziós jelenetbeli információkra épül. Ilyen például a vertex color (ami lehet a

megvilágítás eredménye, netán a grafikusok által előre megadott), vagy éppen a köd. A raszterizálás folyamatának gyorsítása már évek óta alapkövetelmény, így az egyes fázisok szorosan kapcsolódnak a hardver felépítéséhez. A raszterizálást a pixel pipeline végzi, amely sorra veszi a geometriai egységtől (CPU, avagy a hardver) kapott poligonok által elfoglalt pixeleket. A pipeline része a TMU (Texture Management Unit, textúrázó egység), amely az egyes vertexekhez tartozó információk alapján kikeresi, hogy melyik texelre (a textúra egy pixele) lesz szükség, majd ezt filtereli (szűri) és így előállít egy RGB színértéket. Ezután ez utóbbit különféle más adatok (a vertex színe, köd-effektus) módosíthatják és esetleg ún. alpha-blending-re is sor kerülhet, ha a textúra tartalmazott átlátszósági információt. (Blendelni konstans értékkel is lehet, nem kell hozzá feltétlenül textúra.) A DirectX 7-ben is volt már lehetőség multitextúrázásra; azaz a pixel színének kiszámításakor két textúrát is fel lehetett használni, azokat többféleképpen kombinálva (bár ekkor még csak viszonylag egyszerű műveletekre volt lehetőség). Ilyen effekt az Environment Mapped Bump Mapping, röviden az EMBM – ehhez három textúrára van szükség. A light mapping talán közismertebb és egyszerűbb is – ennél ugye nem a vertexre számol világítást, hanem a poligonon lévő textúrából veszi azt. A pixel pipeline felépítése is meglehetősen kötött volt tehát a DirectX 7 esetén, egyértelmű volt, hogy a programozók nagyobb szabadságra vágytak, amit a 8.0-ás verzió tálcán kínált a számukra.

A DirectX 8 egyik legfontosabb előnye a szabadabb programozhatóság megjelenése volt, bár fontos kiemelni, hogy az API-n magán is sokat javítottak (felépítés, dokumentáció). 3D-s pipeline-ja szintén több fő részre osztható. A geometriai szakasz első új része a tesszeláció; ez a HOS, azaz magasabbrendű felületek poligonokká alakítását jelenti. Ez a gyakorlatban nem terjedt el, játékokban szinte sehol sem használják. Ezután a transzformációs és megvilágítási (T&L) műveletek elvégzése a vertex shaderek feladata, majd különféle egyéb számítások (clipping, backface culling) következnek. A DirectX 7 (multi)textúrázásának funkcióját a pixel shaderek vették át, majd a végén következnek a multisampling-et használó frame-buffer műveletek. Nézzük meg részletesebben mit is takarnak a vertex- ,pixel shader és multisample rendering kifejezések!

Vertex shader: először is, a shaderek tulajdonképpen sajátos utasítások használatával megírt rövidke programok. Ezekkel adhatják meg a programozók, hogy mi történjen a meglévő adatok sorozatával. A DirectX 8 shader nyelv egyszerű utasításokra épül, ami leginkább az assembly-re emlékeztet: rövid utasításokból áll (add, mul, dp3), ezekkel lehet

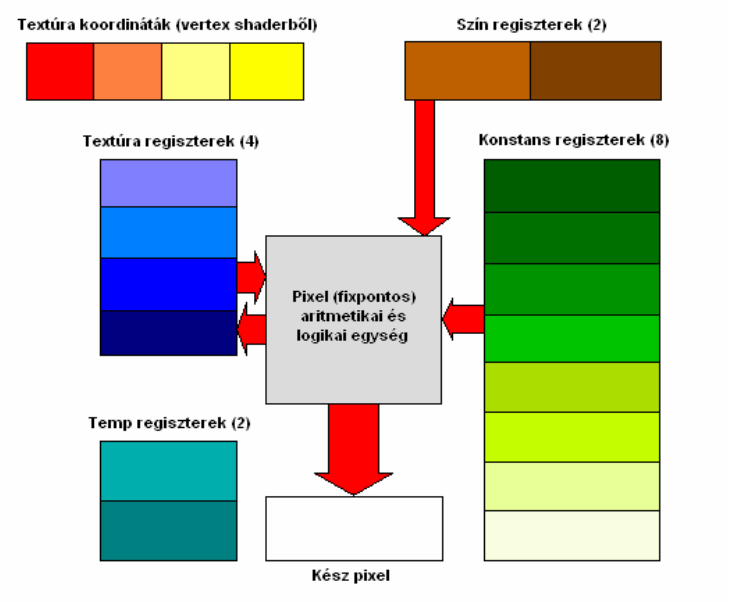
manipulálni jelen esetben a vertexeket. Fontos, hogy egyszerre csak egy vertex dolgozható fel, viszont van lehetőség azonos shadert alkalmazni egy egész sorozat vertexre. Egy vertexhez viszont nagyon sok adat tartozik: először is az XYZ háromdimenziós koordináták, aztán egy vagy több szín, egy vagy több UV textúra-koordináta, átlátszóság és egyebek. A támogatott számítások sora igen hosszú, az alapvető összeadás-szorzás mellett találhatunk skalárszorzat (dot product) és mátrixműveleteket is. A vertex shadert többnyire a vertex pozíciójának, illetve textúra koordináták meghatározására és a megvilágítás kiszámítására, valamint per-pixel lighting esetén annak előkészítésére használjuk. Tehát a vertex shaderek segítségével a T&L megvalósítható (viszont a programozhatóság miatt lehetőség nyílik például saját megvilágítási modellt készíteni), sőt, lehet ún. csontváz-animációt is végezni (a „csontok” mozgását általában a CPU számolja, a vertexek súlyozását számolja a GPU). *(Függelék, 59. ábra)*



37. ábra Vertex shader architektúra

Pixel shader: Ez a rész szolgál talán a legkomolyabb újításokkal és ettől várhatjuk a leglátványosabb eredményeket is. A korábban tárgyalt pixel pipeline ugyanis kiegészült a pixel shader egységgel, amelynek feladata a TMU-k által szolgáltatott texelek feldolgozása. A támogatott műveletek egyik fele a pixel színének kiszámításával kapcsolatos, a többi pedig a textúrakoordinátákkal. Nagyon fontos újdonság az, hogy ezesetben a texeleket már nem csak mint színértékeket kezelik, hanem vektor + skalár adatokként. Azaz, az RGB értékek egy háromdimenziós vektort is megadhatnak, az alfa érték pedig valamilyen kisméretű egész

számot. PS1.x-ben a temporary és color regiszterek komponensei 0 és 1 közti értékeket tartalmaznak.



38. ábra Pixel shader architektúra

Multisample rendering: a DirectX 8-as effekteknek ez a része jobbára a 3dfx öröksége, tulajdonképpen a T-bufferhez hasonló, de annál tágabb lehetőségekkel rendelkező technikáról van szó. Manapság a multiple render target (MRT) nevű módszert alkalmazzák, melynek lényege, hogy egyszerre több felületet is képes renderelni, egy menetben számítva az összes adatcsatorna (pl. szín, mélység) értékeit. [2]

Jelenleg a DirectX legszélesebb körben használt verziója a 9.0, amely lassan 5 éves. Olyan látványos újításokat vezetett be, mint a vertex- és pixel shader 3.0-ás verziója (ld. FarCry, Crysis).

6.3.2 OpenGL

A Direct3D mellett a másik grafikában használt szoftver interfész. Az OpenGL-t (akkor még IrisGl) a Silicon Graphics (SGI) nevű amerikai cég fejlesztette ki, eredetileg saját grafikus munkaállomásainak programozására. Ezek a munkaállomások nagyon gyors grafikus célhardverek voltak, amelyek ultragyorsan végeztek olyan műveleteket, amelyek szükségesek a grafikai számításokban, így a Silicon Graphics számítógépek rendkívül gyorsan végeztek

például mátrixtranszformációkat. Azért, hogy a programokat más rendszerekre is át lehessen vinni, az SGI átdefiniálta az IrisGl-t, és az OpenGL nevet adta neki. Az egyik legfontosabb szempont az OpenGL kifejlesztésénél a hordozhatóság volt, ezért az OpenGL, az IrisGl-el szemben más rendszerek felé is nyitott (OpenGL = Open Graphics Library). Az első OpenGL specifikációt 1992 júl. 1-én mutatták be.

Az OpenGl pár száz eljárásból és függvényből áll, melyek lehetővé teszik 2 és 3 dimenziós grafikai objektumok létrehozását, és ezeken az objektumokon műveletek elvégzését. Lehetőség van görbék és felületek használatára; külön függvények vannak a kontrollpontok megadására, a tesszeláció meghatározására, valamint a tesszelált felület (mesh) kirajzolására. Az OpenGL tehát egy eljárás- és függvénygyűjtemény, melyek 2 és 3 dimenziós geometriai objektumok specifikációját tartalmazzák; ezenkívül olyan eszközöket is nyújt, melyekkel szabályozni lehet ezen objektumok leképezését a képpufferbe, amelyben az OpenGL az eredményként létrejövő képet tárolja. Ennek megjelenítése már az operációs rendszer, vagy az ahhoz tartozó ablakozó rendszer feladata. Itt elérkeztünk egy fontos dologhoz: az OpenGL nem tartalmaz ablakozó rendszert, és nem támogatja az input eszközök kezelését sem, tehát ezeket a dolgokat az adott nyelven a programozónak kell megoldania.

A Unix-os (Linux-os) OpenGL rendszerek grafikus felülete többnyire az X-Window, amely tartalmazza az ablakozást. Windows 95, 98 , XP és NT esetén maga az operációs rendszer szolgáltatja a grafikus felületet. Az OpenGL platformfüggetlenségét az adatbeviteli és megjelenítési rendszertől való függetlenség biztosítja. Az ügyfél-kiszolgáló (kliens-szerver) felépítést követi, ezáltal lehetővé válik, hogy a grafikus alkalmazást futtató, és a végeredményt létrehozó gép egyazon, vagy két különböző gép legyen. A grafikus alkalmazás - mint ügyfél - parancsokat ad az OpenGL kiszolgálónak, amely létrehozza a képet. Mivel a parancsok átadására szabványos protokollt dolgoztak ki, ezért az ügyfél és a kiszolgáló gépek különböző típusúak is lehetnek.

Az OpenGL funkciói:

- a színtér definiálása háromdimenziós (vagy kétdimenziós) primitívekkel
- a nézőpont specifikálása
- megvilágítási modellek alkalmazása
- a megvilágított színtérről árnyalt modell készítése
- árnyékok (fények és anyagok) és textúrák alkalmazása
- atmoszféra effektusok kezelése (pl.: köd)

- anti-aliasing (élsimítás), motion blur (mozgó objektumok körvonalainak elmosása)
- culling, clipping, stencil, depth, blending

Az OpenGL alapfogalmai:

Az OpenGL primitíveket rajzol. A primitívek grafikai alapelemek. Az OpenGL geometriai primitívei a pontok, a szakaszok és a sokszögek (poligonok). A geometriai primitíveket vertexek (csúcspontok, 3D pontok) definiálják. Egy vertex definiálhat egy pontot, egy szakasz végpontját, vagy egy poligon csúcspontját, tehát minden OpenGL geometriai primitívet meg tudunk határozni a vertexeivel. A vertexek struktúrák, melyek tartalmazzák az illető csúcspont térbeli koordinátáit, színét és egyéb adatait. Az OpenGL minden vertexet függetlenül, rendezetten és ugyanúgy kezel. Az OpenGL más struktúrákat is használ, például pixelnégyszögeket, bittérképeket. Ezeket raszterprimitíveknek nevezzük. Fontos, hogy megkülönböztessük a geometriai és raszterprimitíveket, mivel azokat az OpenGL eltérő módon kezeli. Az OpenGL-t állapotautomataként is fel lehet fogni, mivel rendelkezik egy ún. state-tel (állapot). Ezen state tartalmazza azokat az érvényes adatokat, amelyek szükségesek a specifikált objektumok leképezéséhez. Tárolja, hogy pl. a világítás, (azon belül mely fényforrások), az élsimítás, az árnyalás, stb. engedélyezve van-e, vagy le van tiltva. Ezeket az információkat általában egyetlen bit tárolja, ha a bit 1 akkor engedélyezett, ha 0 akkor nem. Az OpenGL-ben minden felhasznált paraméter rendelkezik egy iniciális vagy alapértelmezett (default) értékkel, pl.: az alapértelmezett RGBA szín az (1.0, 1.0, 1.0, 1.0); az alapértelmezett transzformáció és vetítési mátrix pedig az egységmátrix.

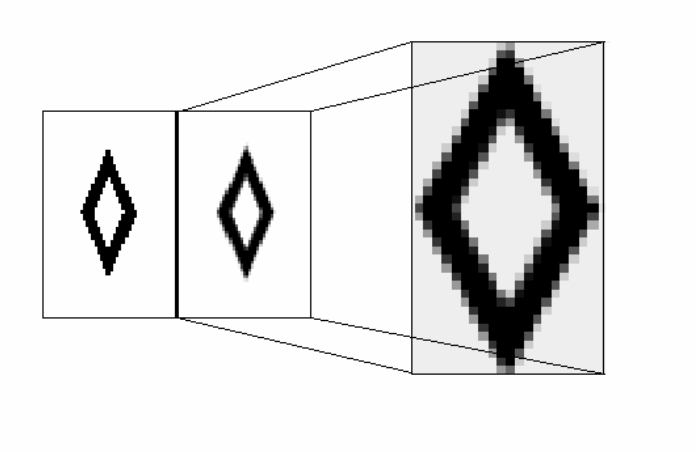
Az OpenGL a megjelenítéskor a Descartes-féle koordináta rendszert használja (Cartesian coordinate system), tehát a bázis olyan vektorokból áll, melyek mindegyike merőleges a többire és egységnyi hosszúak, azaz ortonormált. A koordinátákat a megszokott x, y, z hármassal jelöljük. Mivel a számítógépes grafikában leggyakrabban a jobbsodrású rendszerek használatosak (manapság már a balsodrásúak – x jobbra, y fel, z előre), ezért az OpenGL is ezt használja. Jobbsodrású koordináta-rendszer esetén a (0, 0, 0) pontban van az origó, az x, y tengely pozitív része az origótól jobbra ill. fölfelé található, a z tengely pozitív része a képernyőből kifelé mutat. OpenGL-ben az origó a camera space közepén van, azaz a nézőpont a (0, 0, 1)-ben van. (DirectX-ben a nézőpont az origóban van.)

Az OpenGL kétféle színmódot használ: az RGBA színmódot, illetve a színindex módot. Az RGBA színmódban minden színt négy komponens definiál, a vörös (Red), zöld (Green), kék (Blue), illetve az alfa (Alpha) komponens. Minél nagyobb a komponens értéke,

annál intenzívebben vesz részt a létrejövő színben. Színindex módban minden színt egy lebegőpontos érték ír le és minden ilyen lebegőpontos értékhez hozzá van rendelve három 8 bites érték a memóriában, rendre a három szín intenzitása. Indexelni egész értékkel lehet, de nem használjuk a palettás módot és a videókártyák már textúráknál sem támogatják.

6.4 Látványjavító eljárások, technikák

Anti-aliasing: Tudjuk, hogy kép sok egyforma képpontból (pixel) épül föl. Ha húzunk egy vonalat átlósan a képernyőn, akkor ennek megrajzolásához esetenként kisebb lépésekre volna szükség, mint az elemi képpont, de mivel ez nem lehetséges, a vonalat közelről megvizsgálva „recés" lesz, itt-ott „kiállnak belőle" a kényszerből létrehozott képpontok. Az anti-aliasing (élsimítás, élletörés) funkció a „recés" éleket okozó képpontokat a vonal színe és a háttérszín közötti árnyalatok felhasználásával elmossa. Első ránézésre sokkal rosszabbnak tűnhet az eredmény, mint az eredeti kép, de ha egy kicsit távolabb megyünk, a szemünk nem különbözteti meg élesen az árnyalatokat és így egy szép, sima élű (talán kissé elmosódott) átlós vonalat látunk. A videókártya is ezt az effektust használja, hogy elsimítsa a modellek és textúrák széleit. Legközismertebb fajtája az ún. full screen anti-aliasing (FSAA), azaz teljes képernyős élsimítás *(39-40. ábra).*



39. ábra: A jobboldali alakzaton alkalmaztuk az élsimítást (részletek a nagyított képen)

40. ábra Full Screen Anti-Aliasing (FSAA)

FSAA típusok:

Supersampling: Minden egyes pixel többször kerül kiszámításra, majd a kártya a frame-buffer-be menti az eredményeket és ott kombinálja őket a végső kép előállításához. (GeForce2/Radeon)

Multisampling: Supersampling-tól annyiban tér el, hogy az egyes pixelekhez a kártya csak egy textúra-mintavételezést végez és ezt az adatot újrafelhasználva memória-sávszélességet takarít meg. (GeForce3)

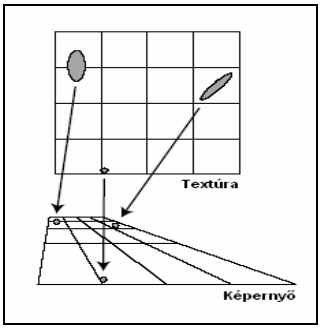
Ordered grid: Az egyes pixelek kiszámításához lerenderelt további minták elrendezése négyzetrácsos jellegű. (nVidia, ATI)

Rotated grid: A minták elrendezése négyzetrácsos, azonban minden egyes pixel mintáit elforgatják a középpont körül, általában körülbelül 20-30 fokos szögben. (3dfx)

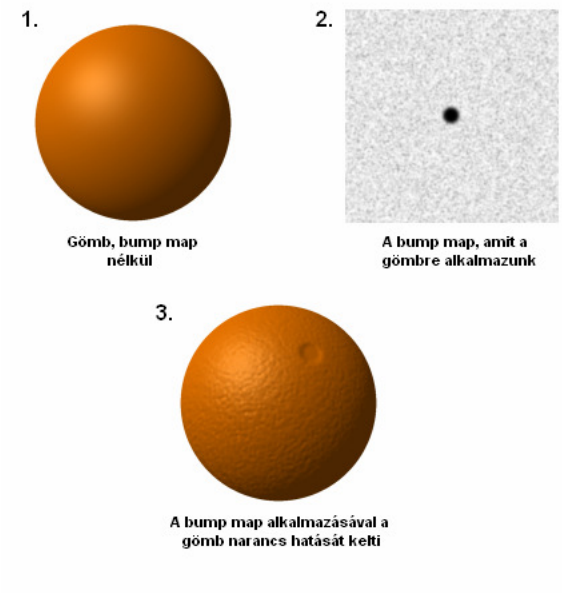
Quincunx: A minták elrendezése a dobókocka ötösével megegyező; a sarkokon lévő mintákat a kártya újra felhasználja a szomszédos pixelekhez. [4]

Bump-mapping: Olyan eljárás, amely a kétdimenziós textúra felületét „érdessé” teszi, sokkal realisztikusabbá téve azokat. A bump mapping-hez alapesetben két textúrára van szükség, de color map nélkül is lehet egyszínű felületet „érdessé” tenni *(42. ábra)*. Az egyik maga a textúra, amit „érdesítünk”, a másik pedig egy ún. magassági térkép (height map), amely az eredeti textúra egyes pixeleinek magassági értékét reprezentálja. A magassági térkép leggyakrabban egy szürkeskálás kép, melynek minden egyes pixeléhez tartozik egy pixel az eredeti textúráról. Így ha a magassági térkép (0,0) pixeléhez egy (127,127,127) RGB érték tartozik, akkor ez azt jelenti, hogy a (0,0) pixel 0.5 egység „magasan” helyezkedik el. A

(0,0,0) RGB érték a „legalacsonyabban” lévő pixelt jelenti, míg a (255,255,255) RGB érték a „legmagasabban” lévőt. Miután megvan a magassági térképünk, felhasználjuk a normal map előállításához. A normal map a textúra egyes pixeleinek a normálvektorát reprezentálja. Az x értéket az r komponensbe, az y értéket a g-be, a z értéket a b-be ágyazzuk. Minden egyes pixel normálvektorára szükség van, hogy mindre a megfelelő mennyiségű megvilágítást alkalmazni tudjuk. A normal map a *texture space*-ben (ami nem más, mint egy a vertexek xyz és uv értékei által meghatározott lokális bázis) van megadva *(41. ábra)*.

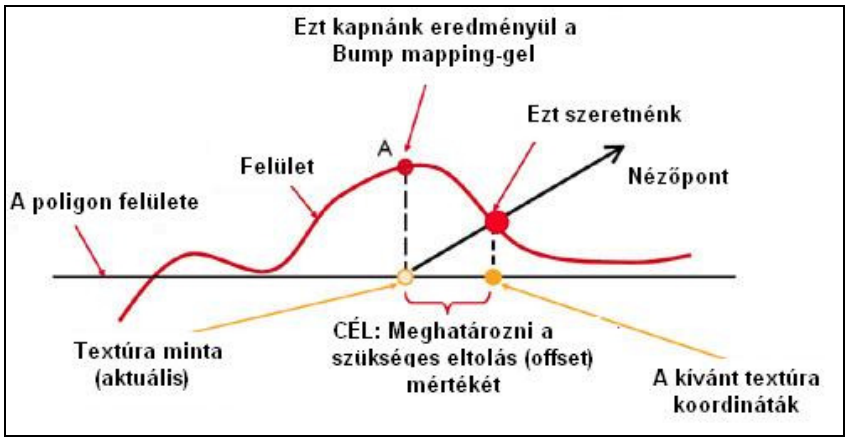


41. ábra Ugyanazon területek texture- , illetve screen space-ben



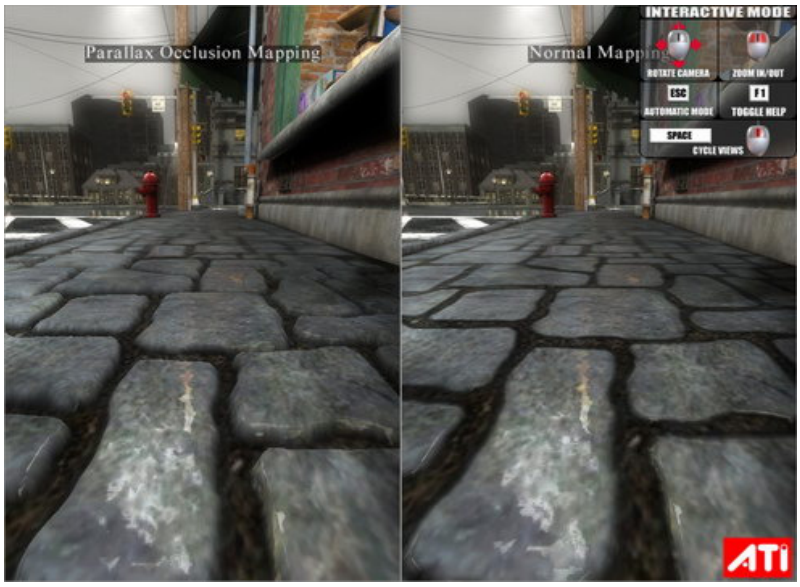
42. ábra Bump mapping

Parallax mapping: Az eljárás a Bump Mapping továbbfejlesztett változata. Elődjéhez képest figyelembe veszi a nézőpont változásából adódó látszólagos eltolódást – parallaxist – így a felület minden irányból másképp fog kinézni, valóságosabb lesz. Magassági térkép itt is van, de itt igazából mélységinek kellene hívni, hiszen a Parallax Mapping a poligon felülete mögött játszódik. A kiválasztott képponthoz először a normal map alapján képezik a megfelelő pontot a poligon felületére, majd a kamera nézési iránya alapján (eye vector) kiszámolják, hogy az adott irányból melyik pont lenne a helyes válasz. A két érték különbsége a szükséges eltolás (offset), mellyel módosítani kell a normal map mintavételezését. Az alábbi ábrán látható, hogy a kamera irányának változásával az eltolás is változik.

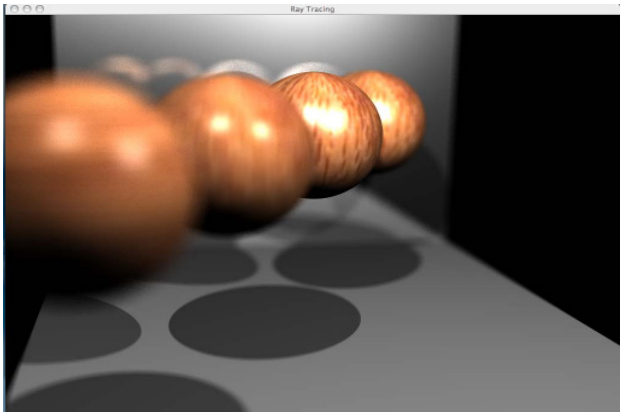


43. ábra A Parallax Mapping elmélete

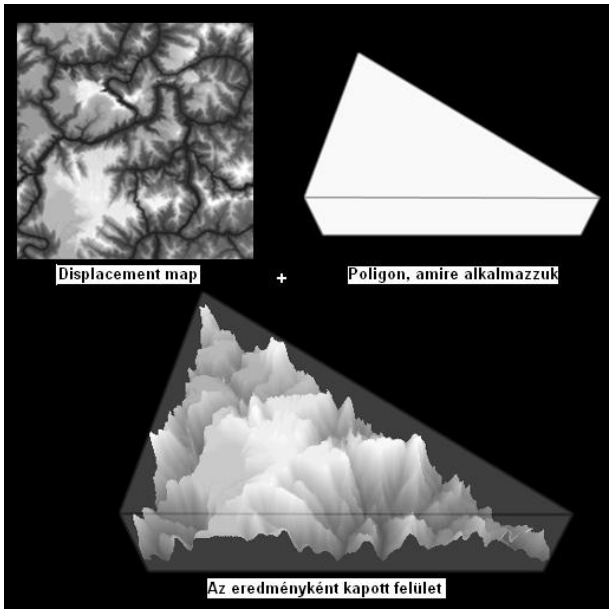
Az eljárást azt ATI továbbfejlesztette. Az általuk kidolgozott Parallax Occlusion Mapping (POM) az egyszerű Parallax Mapping pontatlanságait is kiküszöböli, helyesebben adja vissza a felület egyenetlenségeit (még lapos beesési szögeknél is), pontosabban képezi a felület többi részének árnyékát. Mindezt a pixel shaderben kell megírni, nincs rá hardvertámogatás.



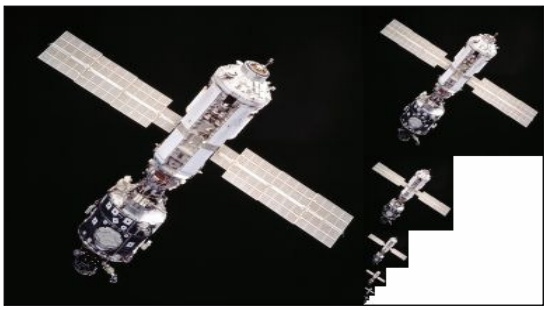
Depth-of-field: (más néven focal anti-aliasing) olyan módszer, melyet a filmipar előszeretettel használ, ám ezidáig a játékipar hanyagolt. Segítségével a játékkészítők rá tudják irányítani a játékos figyelmét bizonyos dolgokra a képernyőn azáltal, hogy a kép egy bizonyos része éles marad, a többi viszont elmosódottá válik.



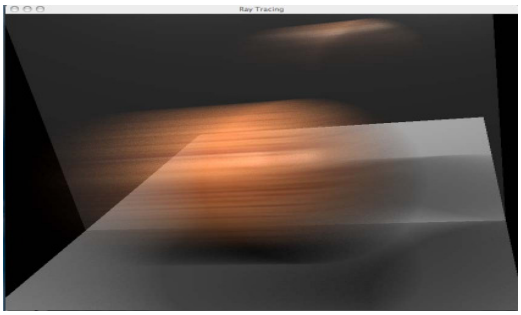
Displacement mapping: A bump- , normal- és parallax mapping mellett egy további megoldás a felületek „érdesítésére”. Gyakorlatilag ez egy olyan eljárás, amely a végeredményre nézve kísértetiesen hasonlít a bump mappingre. Azonban azzal ellentétben itt nem csak a felület normálvektorát módosítják, hanem magát a felületet is. Ez gyakorlatban annyit jelent, hogy a displacement mapping (DM) eljárással előállított testeken a hatás nem csak a felületen, hanem a kontúrvonalakon is látszódni fog. Példaként: bump mapping használatával egy földgömb kontúrvonala kör marad, míg DM használatával a kontúrvonal is megváltozik - a hegyeknél kidudorodik, a völgyeknél beljebb nyomódik -, mivel a DM eljárás magát a test geometriáját változtatja meg *(44. ábra)*.



44. ábra Displacement mapping



Mipmapping: A mipmap-ek textúrák különféle méretű másolatainak gyűjteményei. Az elnevezés is erre utal: „**m**ultum **i**n **p**arvo”, azaz „sok dolog kis helyen”. Minden mipmap negyed akkora mint az őt megelőző, azaz, ha az eredeti textúra 64x64-es, akkor ezen textúra mipmap-jei 16x16, 4x4 és 1x1 méretűek lehetnek. A mipmap-eket kifejezetten a szebb látvány kedvéért alkalmazzák úgy, hogy ha szükséges, lecserélik az adott objektum textúráját ugyanazon textúra különféle verzióira (a mipmap-ekre). Ha a kamera közel van egy objektumhoz, a grafikus kártya a legnagyobb méretű mipmap-et (általában az eredeti textúrát) használja és ahogy egyre távolodunk tőle, az egyre kisebb mipmap-ek kerülnek az objektumra. A mipmapping hatására előfordulhat, hogy egy textúra túl elmosott lesz – ekkor használják az anizotróp szűrést.



Motion blur: (más néven temporal anti-aliasing) az emberi szem által valóságosan leképezhető látványt adja nekünk a számítógépünk monitorán, ha gyorsan mozgó objektumokról van szó (mint ahogy például egy hirtelen elsuhanó járműnek csak az elmosódott körvonalait látjuk).

6.4.1 Textúraszűrési módszerek

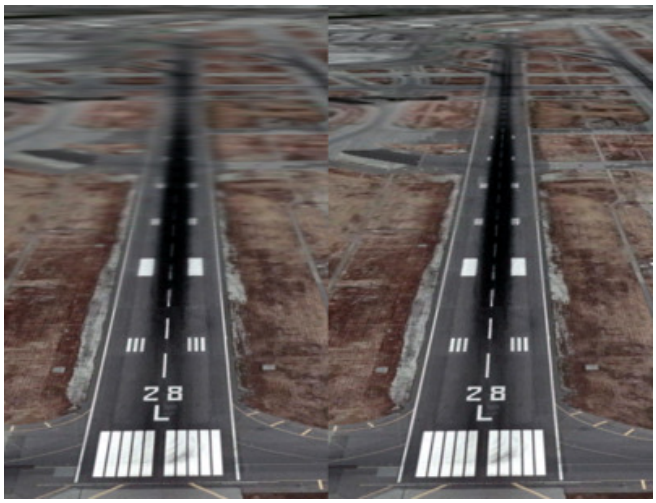
Minden szűrési technika ugyanazt a problémát orvosolja, miszerint túl kevés megjelenített textúraminta esetén újakat hoz létre. A régi, szoftveres renderelést használó játékokban a textúrázás az úgynevezett point sampling eljárást használta, azaz a program a kép minden pixeléhez csak egyetlen ponton vizsgálta a textúra színét. Emiatt a textúrák pixelei közelről nagy, éles körvonalú négyzetekként jelentek meg. A videókártyák fejlődésével lehetővé vált a textúrázás hardveres gyorsítása, miáltal alaposabban meg lehet vizsgálni a textúrát.

Izotróp szűrési metódusok: Legelőször tisztáznunk kell, mi is az az izotrópia. Egy objektumot akkor nevezünk izotrópnak, ha minden vektora a különböző tengelyek mentén azonos értékű: ilyen például egy négyzet, vagy egy kocka. A bilinear és trilinear filtering egyaránt izotróp szűrési metódusok, mivel mindig egy négyzet alakra alkalmazzák őket.

A *bilinear filtering* esetén a kártya minden pixelhez négy ponton kérdezi le a textúra színét és az eredményt átlagolja – ennek eredményeként a textúra pixelei elmosódottan jelennek meg a képen, ami kedvesebb a szemnek. Ehhez viszont 4 texelt kell beolvasni a memóriából az eddigi 1 helyett.

A *trilinear filtering* az előző módszernek egy bővítése, amely már úgynevezett mipmap-eket is használ. Ekkor a hardver a jelenlegi és az eggyel kisebb felbontású mipmapből egyaránt 4-4 mintát vesz, ekkor az eredmény szebb, bár kissé homályosabb is lehet. Azonban ehhez már 8 texel beolvasása szükséges.

Anizotróp szűrés (Anisotropic filtering): Akkor van szükség ilyen szűrési technikára, amikor a szűrési metódusnak különböző értékeket kell figyelembe vennie az x, y és z tengelyek mentén. Tehát ilyenkor nem kocka, hanem általában hasáb alakú formán kerül sor a textúraminta szűrésére. A képernyőn egy képpont több texelinformációt tartalmazhat az y tengely, mint az x tengely irányában. Ilyenkor jön jól az anizotróp szűrés, hogy továbbra is megmaradjon a helyes perspektíva és képélesség. Amennyiben egy olyan képen, amelyen egy távolba tartó utat, vagy akár szöveget látunk, nem így történik a szűrés, akkor teljesen elmosódottan látjuk majd az objektumokat. Ha azonban anizotróp szűrést alkalmazunk, a képen a távolba vesző objektumokon is élesek maradnak a textúrák. Természetesen több fokozata van ennek a funkciónak is – minél több textúramintát alkalmaz a szűrés, annál élesebb lesz a kép (és annál nagyobb erőforrást igényel). Ennek megfelelően beszélünk 2, 4, 8, és (kizárólag ATI kártyáknál) 16-szoros szűrésről.



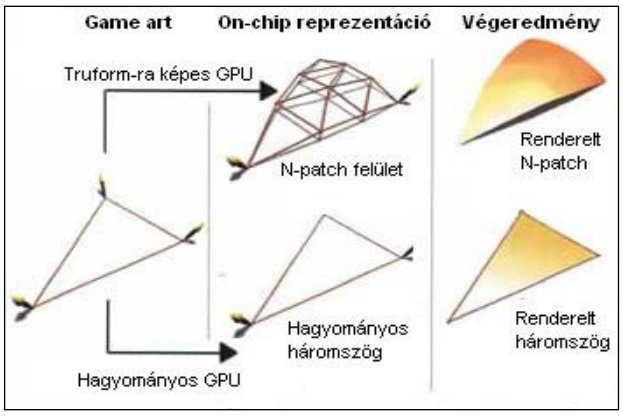
45. ábra Trilinear filtering mipmap-pel, illetve ugyanez, anizotróp szűrést alkalmazva

Vannak más módszerek is a textúra szűrésére (pl. summed area table), de ezekhez nincs hardver. Az anisotropic filteringet azonban számolhatnák non-uniform mipmap-ekből is, amikor a textúra összmérete nem 4/3-szorosa, hanem 4-szerese az eredetinek (azaz vannak 1x2, 1x4, 2x4, 2x1, 4x1, 4x2, stb. méretű mipmap-ek is).

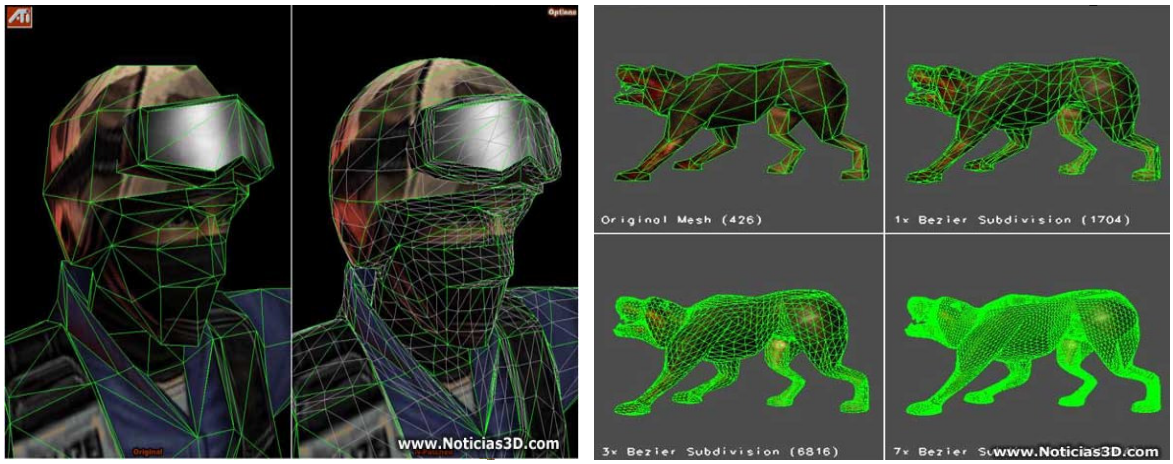
6.4.2 ATI Truform, a technika, ami feledésbe merült

A lényege az, hogy egy elsőrendű felület sík (ilyenek a poligonok), a másod- és harmadrendűek viszont görbületeket is tartalmazhatnak. A gyakorlatban egyébként az ívelt felületeket is sok kis sík felület alkalmazásával közelítik, de nem kell minden egyes ilyen kis felületet (azaz általában poligont) meghatározni, ugyanis azok a magasabb rendű felületet definiáló egyenletekből kiszámíthatók. A 3D-s kártyák esetén a fentiekből látható a görbe alapú felületek használatának egyik fő előnye: nagyságrendekkel kevesebb adat is elegendő a tárolásukhoz. Mivel az alkalmazásukhoz szükséges matematika már évtizedek óta rendelkezésre állt, ezért csak idő kérdése volt a hardveres támogatás megjelenése.

A DirectX 8 kétféle magasrendű felületet (HOS) támogat: ezek közül az N-patch típusú az ATI fejlesztése. Ezen N-patch-ek esetében valójában egy Besier típusú háromszögpatch-ről van szó; ennek megfelelően egy felület definiálásához három sarokpontra és hét további kontrollpontra van szükség. A trükk az, hogy az utóbbi hét kontrollpontot hardveresen generálják a három sarokpontból, az egyes pontok normálvektoraiból kiindulva. Ezek a vektorok a pontban találkozó háromszögek normálvektorainak eredői. Nézzük meg miért is jó ez! Az összesen tíz kontrollpontból kiindulva a magasabb rendű felületet leíró egyenleteknek megvannak a változóik, azaz az N-patch tetszés szerint bontható fel háromszögekre - akár több száz darabra -, amelyek folytonos és ívelt felületet alkotnak. Továbbá az újonnan létrehozott háromszögeket alkotó pontoknak is lesz normálvektora, így például sokkal finomabb, pontosabb bevilágítás hozható létre. Mindehhez pedig a forrásadatokat egy egyszerű poligon szolgáltatja, ráadásul az



összes számítást a 3D-s chip végzi el, tehát a végső képen megjelenő hatalmas poligonszámhoz sokkal kevesebb adatot kell mozgatni. Ez gyakorlatban annyit jelent, hogy bármilyen poligonokból felépített modell N-patch-ekké, azaz ívelt felületekké alakítható.



46. ábra A Truform a gyakorlatban

Az ATI állítása szerint ez a technológia minden meglévő játékban alkalmazható és nem igényel komolyabb változtatásokat a fejlesztők (leginkább a 3D grafikusok és pályatervezők) munkamódszereiben sem. Ez persze színtiszta marketing, hiszen a valóságban a poligon-N-patch lekerekítés során létrejövő modell formája nem mindig hasonlít arra (szinte sosem), amit a modellkészítője elképzelt; tehát egy jól kinéző N-patch modellhez tudatosan kell építeni a technológiára. Emellett az átalakítás minden poligont különállóként kezel, ami nem garantálja azt, hogy a létrejövő N-patch-ek összeérjenek (azaz a modelleken rések jelenhetnek meg). 2001-ben ez a technológia még ígéretesnek bizonyult, de - leginkább a fent említett hiányosságai miatt - nem terjedt el, illetve az évek során ennél sokkal hatékonyabb, látványosabb végeredményt előállító technológiák láttak napvilágot. [6]

Összefoglalás

A számítógépes játékok grafikai fejlődése szinte megállíthatatlan. Napjainkra már annyira felgyorsult, hogy mire ezeket a sorokat írom, már egy új grafikus motor létrehozásán dolgoznak és az ahhoz megfelelő hardver is gyártósoron van. Azonban az újítás a játékiparban nem mindig éri meg a befektetett energiát. Sokkal biztosabb (és könnyebb, ezáltal gyakrabban alkalmazott) módszer egy már jól bevált receptet követni, megvásárolni egy kész engine-t, amihez olyasvalamit hozzátesznek, amitől a sajátjuknak mondhatják. Persze a vásárolt engine sem mindig garancia a sikerre (pl. a Half-Life készítői hatalmas sikert kovácsoltak az általuk módosított Quake-engine segítségével, míg a Heretic 2 hamar eltűnt a süllyesztőben, annak ellenére, hogy a Quake 2 motorját használta). Azt mondhatjuk az eddigi újító szándékú cégek sikerét/kudarcát tekintve, hogy alapjábanvéve hasznára válik a játékiparnak, ha a fejlesztők mernek újat mutatni, hiszen akár jó is származhat belőle, ha pedig nem, akkor azzal is színesítették a játékipar manapság egyre szürkülő palettáját.

Láthattuk, hogy nem volt ez mindig így, a ’80-as években csak néhány játékfejlesztő cég bontogatta a szárnyait, nem volt nehéz újat mutatni és nem is volt akkora a kényszer, hogy minél hamarabb, valami jobbal kell megjelenni a piacon, mint a konkurencia. Napjainkban óriási versenyről van szó, aminek egyetlen igazi nyertesei a játékosok, akik nélkül a játékipar nem tartana ott, ahol ma. Mindenképpen nagy a szerepük mind a grafikai fejlődésben, mind pedig a játékfejlesztést általában véve, hiszen a játékok nekik készülnek, az ő szórakoztatásukra, az igényeiknek megfelelően. Kérdéses, hogy az erőszakra mennyire volt/van igény a számítógépes játékokban, de az tény, hogy a cenzúrázás és a játékok korhatárossá tétele (ESBR szervezet) ellenére, manapság az ilyen stílusú akciójátékok fogynak a legjobban, az FPS/TPS kategória a legsikeresebb. A fejlődés másik mozgatórugója az egyre nagyobb teljesítményű hardverek és kifinomultabb szoftverek megjelenése, melyek számos új (legyen az grafikus, vagy egyéb) technológiát hívnak életre. Azt, hogy mit hoz a jövő, pontosan nem lehet tudni, de a következő nagy lépés talán az lehet, hogy 30 év után a játékosok felállnak a monitor elől és egy sokkal interaktívabb, virtuális környezetben játszanak tovább. Ebben az esetben a grafika az eddiginél is nagyobb szerepet kapna, még jobban kivenné a részét a „nagybetűs” hangulat megteremtéséből. Egy biztos: vagy így, vagy

úgy, de egészen addig fognak számítógépes játékokat készíteni, amíg az emberek játszanak a számítógéppel; és miért is ne játszanának? Hiszen játszani jó.

Irodalomjegyzék

Magyar nyelvű források:

Szirmay-Kalos László - Antal György - Csonka Ferenc: *Háromdimenziós grafika, animáció és játékfejlesztés*, Computerbooks Budapest, 2006

PC ZED évkönyv '98-99: *A számítógépes játékok múltja, jelene és jövője*, CD Pegasus Kft., 1998

[1] PC Games 2001/01: *Legendák nyomában, a számítógépes játékok első 25 éve*, Game Media Lapkiadó Kft., 2001

[2] PC Guru 1999/04: *Direct3D 8.0*, Vogel Publishing Kft., 1999

PC World 2004/05: *3D a kilencediken - DirectX 9-es videókártyák tesztje*, IDG Communications, 2004

[3] PC Guru 2003/12: *XIII*, Vogel Publishing Kft., 2003

[4] PC Guru 2001/12: *Next Generation*, Vogel Publishing Kft., 2001

Gamer 2003/04: *Doom 3, Deus Ex 2 és a normal mapping*, G.F.H. Kft., 2003

PC Guru 1999/09: *Poligonok és voxelek*, Vogel Publishing Kft., 1999

[5] PC Guru 2002/07: *Doom 3: Engine technológia*, Vogel Publishing Kft., 2002

[6] PC Guru 2001/07: *ATI Truform*, Vogel Publishing Kft., 2001

http://hu.wikipedia.org/wiki/3dfx

http://www.inf.u-szeged.hu/oktatas/jegyzetek/KubaAttila/opengl/alapfog.xml

http://www.inf.u-szeged.hu/oktatas/jegyzetek/KubaAttila/opengl/bev.xml

http://prohardver.hu/p.php?mod=21&id=871

http://hu.wikipedia.org/wiki/Vektorgrafika

Angol nyelvű források:

http://www.gamedev.net/reference/articles/article2325.asp

http://www.answers.com/topic/graphics

http://www.answers.com/topic/vector-graphics

http://en.wikipedia.org/wiki/Stencil_buffer

http://en.wikipedia.org/wiki/Z-buffering

http://en.wikipedia.org/wiki/Displacement_mapping

http://en.wikipedia.org/wiki/Cel-shaded_animation

http://www.answers.com/topic/raster-graphics
http://en.wikipedia.org/wiki/Vector_game
http://en.wikipedia.org/wiki/Battlezone
http://www.answers.com/topic/voxel
http://www.answers.com/topic/outcast-game
http://www.answers.com/topic/star-wars-arcade-game
http://en.wikipedia.org/wiki/Arcade_game
http://www.answers.com/ellipsoid
http://www.answers.com/ecstatica
http://www.gamespot.com/pc/action/ecstatica2/review.html?sid=2535803&om_act=convert&om_clk=gsupdates&tag=updates;title;2
http://www.adventureclassicgaming.com/index.php/site/reviews/44/
http://www.answers.com/topic/comanche-series-1
http://en.wikipedia.org/wiki/Onomatopoeia
http://en.wikipedia.org/wiki/XIII_%28game%29
http://en.wikipedia.org/wiki/Hidden_surface_determination
http://www.celshading.com/
http://www.answers.com/topic/half-life-game
http://www.answers.com/topic/mystery-house
http://www.answers.com/topic/king-s-quest-series-1
http://www.answers.com/topic/king-s-quest-i-quest-for-the-crown
http://www.answers.com/Adventure%20Game
http://sierra-fanfare.com/sierra/
http://blog.thingoid.com/category/programming/interactive-fiction/
http://www.answers.com/topic/wolfenstein-3d-engine
http://www.answers.com/topic/wolfenstein-3d
http://www.answers.com/topic/castle-wolfenstein
http://www.answers.com/ray%20casting
http://www.answers.com/topic/doom-engine
http://www.answers.com/topic/doom
http://www.answers.com/topic/doom-ii-hell-on-earth
http://www.answers.com/topic/build-engine

http://www.answers.com/topic/duke-nukem-3d

http://www.answers.com/topic/quake-engine

http://www.answers.com/topic/quake-ii-engine

http://www.answers.com/topic/quake-iii-engine

http://www.answers.com/topic/quake-4

http://www.answers.com/topic/little-big-adventure

http://www.adventureclassicgaming.com/index.php/site/reviews/55/

http://www.answers.com/topic/little-big-adventure-2

http://www.answers.com/topic/the-terminator-future-shock-1

http://www.answers.com/topic/skynet-1

http://www.answers.com/topic/far-cry-3

http://www.answers.com/topic/cryengine

http://www.answers.com/topic/cryengine2

http://www.answers.com/topic/crysis-1

Függelék

47. ábra Will Crowther és Don Woods, az "elsők"

48. ábra John Carmack

49. ábra Ken és Roberta Williams

50. ábra A King's Quest IV (1988), V (1990), VI (1992) és VIII (1998)

51. ábra Egy 2002-es Diablo-klón, a Divine Divinity

52. ábra A Rise of the Triad (1994) módosított Wolfenstein 3D - motort használt

53. ábra A Heretic (1994) és a Hexen (1995) fantasy világban játszódó Doom-klónok

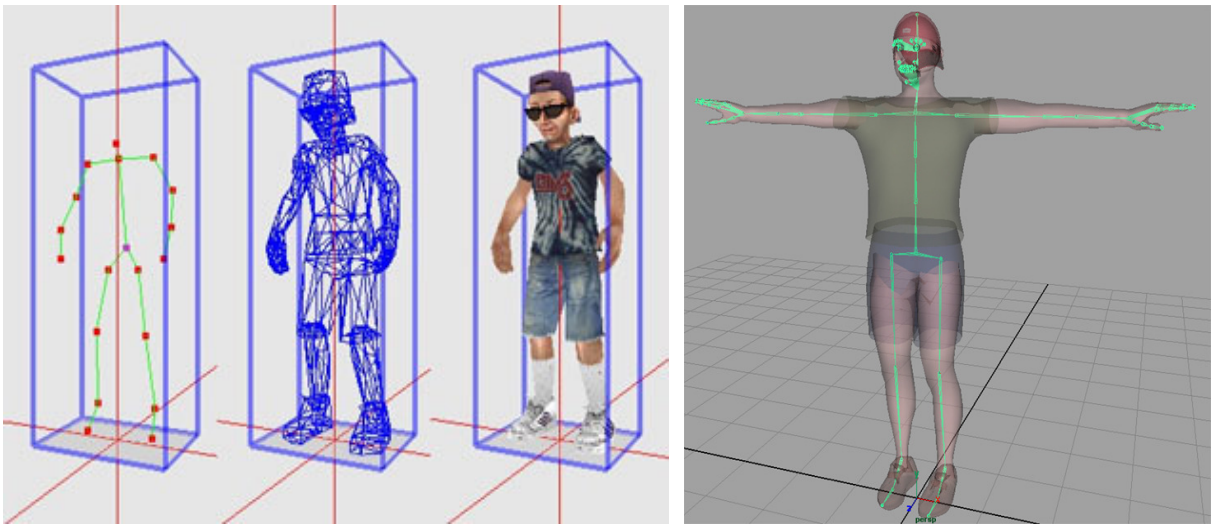
54. ábra Hexen 2 (1997) továbbfejlesztett Quake-engine-nel

55. ábra Az 1998-as Half-life (vetett árnyék módosított Quake motorral!)

56. ábra A Prey (2006) a Doom 3 engine-re épült

57. ábra Unreal 3 engine

58. ábra Csupán a növényzet árulkodik a jobb oldali képeken a CryEngine 2-ről



59. ábra Csontváz-animáció

Köszönetnyilvánítás

Köszönettel tartozom Ledeczky Gábornak és Nagy Róbertnek a dolgozatom elkészítéséhez nyújtott segítségükért.