Focul – Zeul mileniului 3

„Nu ştiu alţii cum sunt dar eu, când mi-aduc aminte de copilăria mea…" Astfel începea un capitol important din opera marelui nostru Ion Creangă, „Amintiri din copilărie". Şi nici că se potrivea mai bine altă introducere pentru acest articol, mai bine zis această succesiune de articole, ce se va întinde pe mai multe numere ale revistei noastre.

De ce am amintit de copilărie? Fiindcă a mea, cel puţin, nu a fost foarte diferită de cea a lui Creangă. Deşi sunt orăşean 100%, mi-au plăcut foarte mult vacanţele petrecute la bunici, într-un colţ uitat de lume. Nici nu vreau să amintesc cum îmi petreceam ziua şi cum mă jucam împreună cu ceilalţi nepoţei, veniţi şi ei la rudele lor mai vârstnice. Aceasta a fost generaţia noastră. Cam pe când a început să ne mijească mustaţa şi ne făceam şi noi curaj să ne uităm după fetişcane, a apărut în viaţa noastră o cu totul altă formă de divertisment. Un aparat ciudat, ce purta numele de Spectrum (localizat sub denumiri gen Cip sau HC), în care, cu ajutorul unui casetofon, încărcam jocuri... jocuri de calculator. Nici nu bănuiam atunci până unde se va ajunge. Revoluţia tehnologică din acest domeniu a trimis repede la reciclare aceste ustensile, pe birourile tinerilor jucători de acum, s-au instalat confortabil Pentium-uri II şi chiar III. Jocul a devenit un fenomen mondial şi una dintre cele mai profitabile industrii. În spatele acelor CD-uri, pe care cu mândrie le introducem în calculatoarele personale, stau ani întregi de muncă a sute de oameni. Împreună, în acest foileton, vom încerca să descifrăm tainele „pistelor argintii", să vedem cum se produce un joc, de la idee până la CD-ul pe care-l cumpărăm din magazine. Cum e şi normal, nu sunt eu tocmai geniul cel mai mare ca să cunosc toate detaliile acestui drum şi, din aceste motive, am cooptat la acest proiect doi dintre producătorii români de jocuri: **FunLabs** şi **Ubisoft**. De ce am apelat tocmai la aceştia? În primul rând pentru că sunt români şi lucrează în România, iar, în cele din urmă, pentru a putea prezenta mai multe metode de lucru. În linii generale şi firmele din Occident se lucrează la fel, dar bineînţeles că ceea ce veţi citi în aceste pagini nu înseamnă că sunt paşii obligatorii ai procesului de producţie al unui joc. **FunLabs** reprezintă micul producător care are curajul să aibă o idee şi să o ducă de la un capăt la celălalt, pe când UbiSoft e genul de concern gigantic care are filiale în nenumărate oraşe ale lumii şi împarte producţia pe compartimente.

Normal, primul pas este conturarea unei idei de joc, idee ce va suferi modificări majore până în stadiul final al produsului. Dar, oricum, se începe de la ceva. După aceasta, începe, poate una dintre cele mai importante etape, organizarea şi planificarea muncii. Pentru aceasta am purtat o discuţie foarte interesantă cu domnul Adrian Filippini, manager la **FunLabs**, visătorul care s-a izbit cu capul de pragul de sus.

Pentru început am vrea să stabilim ceva foarte clar: ideile şi părerile conţinute în acest articol reprezintă punctul personal de vedere al echipei **FunLabs**. Dacă vreţi, aceasta este „viziunea" noastră asupra modului de producere al unui joc, viziune obţinută în urma lucrului efectiv la acest proiect. Având în vedere că acesta este primul nostru joc, şi că, astfel, suntem lipsiţi de experienţă, ne rezervăm dreptul de a ne schimba acest punct de vedere pe măsura trecerii timpului, ajungând la sfârşit să fim de acord cu voi:) Să nu spuneţi că nu aţi fost avertizaţi!

Trecând totuşi la lucruri mai serioase, trebuie să vă spunem în primul rând, că, dacă se doreşte realizarea unui joc, cei care iniţiază proiectul trebuie să privească totul din punctul de vedere al unei afaceri. În consecinţă, primul pas este realizarea unui „Business Plan". Astfel, cam tot ceea ce ţine de joc se va „aşeza" în mod logic în acest plan de afaceri. Vom începe în cele ce urmează să detaliem etapele realizării unui astfel de plan de afaceri şi să explicăm un aspect extrem de important al acestei părţi, şi anume că lungul drum parcurs de la ideea propriu-zisă, şi până la produsul finit (cd-ul din magazin), este presărat cu o multitudine de compromisuri realizate între ceea ce se doreşte a se produce (idee joc, game design, game play, ...) şi ceea ce poate fi produs (depinzând de fondurile disponibile, tehnologia ce se poate implementa, etc.).

Acest business plan a fost gândit având în minte următoarele truisme (valabile începând cu anul 1999, căci, de, lumea s-a mai schimbat), ordinea lor fiind aleatoare :

- degeaba vrei să faci un joc dacă nu eşti şi jucător (dacă nu ştii cu ce se mănâncă nu ai cum să ştii ce ai vrea de la un joc şi nici care sunt limitările tehnologice impuse)
- degeaba vrei să faci un joc, doar de dragul de a-l face (dacă nu „tratezi" subiectul foarte serios, foarte profesionist, va fi de slabă calitate, deci nu îl va juca nimeni)
- degeaba vrei să faci un joc dacă nu ai cu cine (un joc nu mai poate fi făcut de unul singur - adică importanţa echipei)
- degeaba vrei să faci un joc dacă nu are cine să îl joace (dacă nu îl faci având tot timpul în minte pe cel ce îl va juca, nu va place nimănui; notă: nu

poți mulțumi pe toată lumea)

- degeaba vrei să faci un joc dacă nu are cine să ți-l vândă (cum să cumpere cineva un joc, dacă nu are de unde să o facă - vezi publicitatea aferentă și importanța numelui distribuitorului implicat)

- degeaba vrei să faci un joc dacă nu poți scoate profit de pe urma lui (de unde bani de salarii, investiții și chirie, cum să te ia un distribuitor în serios dacă nu are ceva de câștigat de pe urma ta; trebuie să ai în spate un suport financiar serios)

- degeaba vrei să faci un joc dacă nu ai răbdarea și puterea necesară pentru a duce un lucru la bun sfârșit

- degeaba vrei să faci un joc - trebuie să dovedești că și poți.

Rolul acestui „business plan" este de a convinge pe finanțator (de exemplu, o bancă, o firmă distribuitoare de jocuri sau, de ce nu, o firmă producătoare) de fezabilitatea proiectului. Conținutul acestui plan de afaceri, pe scurt, ar fi cam următorul :

- ideea jocului (poate fi total originală ori poate prelua și îmbunătăți idei deja existente);

- piața către care se adresează, care trebuie foarte bine cunoscută și înțeleasă (se vor studia segmentul de piață vizat, profitabilitatea acestuia, se vor colecta date și cifre referitoare la concurență, praguri de intrare pe piață, comportamentul consumatorilor);

- strategii de marketing;

- calitatea echipei manageriale;

- structura organizațională, componența echipei și modul de creare a acesteia;

- planificarea și justificarea cheltuielilor;

- venituri previzionate;

- principali indicatori financiari (cifra de afaceri, profit net, rentabilitate, etc.);

- analiza de sensibilitate (modul în care sunt influențați indicatorii financiari de diverse variații ale condițiilor pieței);

- descrierea detaliată a etapelor proiectului (termenele limită).

Dacă cele de mai sus nu v-au speriat așa cum ne-au speriat pe noi atunci când am realizat că trebuie să lăsăm deoparte plăcerea de a ne imagina jocul așa cum dorim și să dedicăm o maaaare cantitate de timp și energie înțelegerii și creării acestui plan de afaceri, adică a tuturor aspectelor legate de partea financiară și managerială a proiectului, și dacă dorința de a realiza un joc nu s-a diminuat, înseamnă că sunteți pe drumul cel bun.

Primul compromis, și cel mai mare, este legat de suma de bani alocată proiectului. Cu alte cuvinte, în funcție de tipul de joc dorit, fondurile necesare variază destul de semnificativ. Ideea jocului va fi astfel amendată „din start" de realitate. Iar trezirea la realitate nu este chiar așa de plăcută. Este surprinzător cât de mult poate înghiți chiar și un „Jazz the Jackrabbit". Pe lângă asta, alegerea tipului de joc ține foarte mult de piața respectivă și de momentul în care se face alegerea: nu cu mult timp în urmă, toată lumea făcea real time strategy, iar acum toata lumea face role playing games, ca să nu mai vorbim de alegerile oportuniste care explică explozia de jocuri de fotbal cu ocazia Campionatului Mondial.

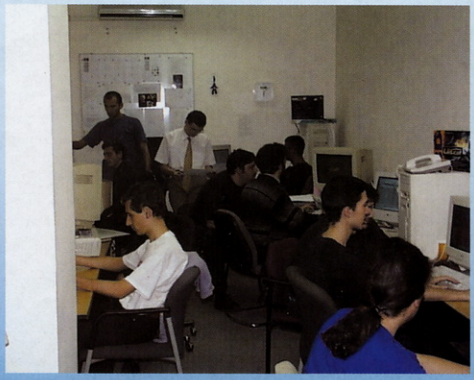
Evident, acestea sunt considerente de ordin general și noi nu dorim să scriem un tratat de genul „Cumpărați-l și veți ști în numai 15 minute ce gen de joc vreți să faceți", dar vă putem spune cum am făcut noi. Fiți atenți! (duruit de tobe, surle, trâmbițe, ce mai - tot tacâmul!). Ne-am hotărât să facem un 3D action/adventure. De ce? Iată motivele: acest tip de joc necesită un buget mai mic decât alte genuri; în al doilea rând, suntem o firmă mică, la început, fără experiență, și un astfel de gen este mai ușor abordabil din acest punct de vedere. În plus, denumirea „action/adventure" și faptul că te „plimbi" printr-o lume virtuală îți permite ție, ca producător, o mai mare libertate din punctul de vedere al creativității. Și, nu în ultimul rând, un engine de acest tip poate fi ușor adaptat și altor genuri: simulator, rpg și, evident, adventure :)

Dar ideea jocului nu constă (evident) numai în tipul de joc ales - aici intră și scenariul care stă la baza jocului, și modul de joc (gameplay-ul), și multe alte detalii, greu de prins în cuvinte care formează atmosfera jocului. Aici pot apare și pericole - cel mai important, credem noi, este lipsa de originalitate.

Nu vom insista asupra cazurilor de plagiat evident (pentru care denumirea de „clonă" ni se pare puțin prea delicată), dar, în anumite cazuri, se poate ajunge la o asemenea situație datorită limitărilor tehnologice. Nu poți lua pur și simplu un lucru care ți-a plăcut la un joc celebru și să-l adaugi jocului tău (pe de altă parte, un 3d action fără strafe n-ar fi un 3d action...); poți, în schimb, să încerci să-ți dai seama ce factor a contat cel mai mult în succesul jocului respectiv și să ții seama de asta în construcția jocului tău. Experiența a arătat că un joc în întregime original nu se bucură de un succes prea mare pentru că necesită din partea jucătorului eforturi mai mari pentru a-și însuși interfața și modul de joc. Soluția este, în opinia noastră, să cauți un echilibru între elementele clasice și cele inovative.

Un alt considerent pentru care ne-am orientat către un joc 3D action adventure este piața. În epoca acceleratoarelor 3D, aceste jocuri ocupau un segment important din vânzări, lucru care suna mai mult decât ademenitor în ochii oricărui investitor (n-ați uitat, aici vorbim despre elementele unui plan de afaceri...)

Să știi cui trebuie să-i vinzi produsul, iarăși nu este de ajuns (știiiim că ne repetăm...). Ceea ce ne aduce la încă o componentă importantă a business plan-ului: marketingul. Marketingul este un element esențial oricărui produs. Se știe de mult că ambalajul și reclama sunt cele care vând un produs (nu sunteți convinși? Ei bine, jocuri care se vând doar pentru că pe copertă pozează o tipă voluptoasă, nu va amintesc de nimic?). Însă, pentru a convinge un investitor, avem



nevoie de mai mult, și anume de un studiu cuprinzător asupra pieței vizate, la care se adaugă o strategie de marketing coerentă. Iată care sunt punctele esențiale ale acestui document:

- segmentul de consumatori pe care se contează;
- motivația de cumpărare a acestora;
- concurenții - puncte slabe și puncte forte (unde îi putem întrece?, putem realiza un produs similar?, putem suporta un atac al acestora?);
- pragul de intrare pe piață, adică suma de bani, timpul, distribuția și alte elemente necesare pentru a realiza un produs competitiv;
- profitabilitatea specifică segmentului de piață vizat.

Odată ideea jocului detaliată și analiza de piață făcută, acestea permit încadrarea jocului nostru într-un segment de piață bine definit. Cu alte cuvinte, știm ce vrem să facem - mai rămâne de văzut cum vom face acest lucru - și, mai ales, cu cine.

Am ajuns deci la organizarea echipei care va lucra la proiect - cum găsim oameni, ce te face să ai încredere în ei sau nu, cum decizi de ce oameni ai nevoie... În construcția echipei trebuie să pleci de la elementele de bază - programatori, game designeri, graficieni, etc. Întrebarea corectă nu este „de câți oameni am nevoie ?" ci, de fapt, „câți oameni îmi permit ?" Asta pentru că trebuie să creezi (pentru început) o echipa cât mai compactă, formată din oameni care să poată îndeplini o varietate mare de sarcini. În fazele inițiale ale proiectului, accentul este pus pe tehnologie, pe dezvoltarea unui engine performant și a uneltelor de lucru. Alți membri vor intra în echipă odată cu trecerea timpului (atunci când ei devin necesari - de exemplu, n-are sens să te zbați să cauți un muzician de clasa, când încă n-ai pus bazele motorului grafic... și cu siguranță va fi nevoie de mai mulți graficieni către sfârșitul proiectului).

Totul sună perfect, până te apuci cu adevărat să strângi oamenii necesari. Principala problema este concepția greșită care există în genere despre o astfel de muncă. Ei bine, nu. Să faci un joc nu e o joacă. În orice cameră ticsită cu gamer-i este îndeajuns să pronunți cuvintele magice: „vrem să facem un joc" și tuturor le sclipesc ochii. Însă dacă încerci să treci la lucruri serioase și le explici că trebuie să muncească între 8 și 12 ore pe zi, fug de parcă ai da cu tămâie. Pe de altă parte, programatorii și graficienii cu experiență privesc un astfel de proiect cu neîncredere, ei bănuind care este de fapt cantitatea de muncă necesară și riscurile implicite.

Cum crearea echipei se bazează foarte mult pe contactele personale, crește și timpul afectat selecției posibililor membri. Și asta nu e tot: ni s-a întâmplat să eliminăm din calcul persoane foarte bine pregătite profesional dar care, pur și simplu, nu s-au integrat în echipă. Insistăm asupra acestui aspect pentru că în acest domeniu, conlucrarea are un rol esențial.

Pe scurt, trebuie să îți formezi o echipa completă, care trebuie să funcționeze ca un tot unitar. Și gândiți-vă un pic ce înseamnă asta : să aduci la un numitor comun pe graficieni, pe programatori, pe scenarist, pe level designer, pe game designer, etc. Iar ideal ar fi să le mai și placă ceea ce fac împreună. Planificarea muncii tuturor trebuie făcută cât mai riguros și cât mai exact, pe compartimente, altfel lucrurile pot scăpa rapid de sub control și totul se va transforma în haos. Unul din cele mai grele lucruri cu care ne-am confruntat a fost găsirea unor algoritmi de lucru în comun, algoritmi ce trebuie îmbunătățiți tot timpul pentru mărirea eficienței.

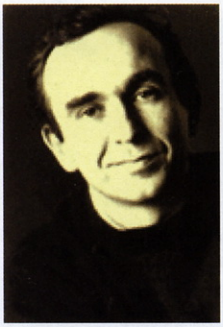
Următorul aspect, ține de fundamentarea cheltuielilor, și este un proces care va da întotdeauna mare bătaie de cap managerilor unei firme. Nu vă cer să vă priviți viitoarea afacere din prisma obsesiei pentru bugetizare a americanilor care iau în calcul până și consumul zilnic de săpun sau hârtie igienică. Însă pentru a atinge scopul propus: obținerea fondurilor, este necesar să dovediți seriozitate. În ceea ce ne privește pe noi, am împărțit cheltuielile în două categorii: investiții (computere - alegerea configurațiilor optime este foarte delicată și vă poate ajuta sau încurca mai târziu -, mobilier de birou, aer condiționat, zugrăvit, alte amenajări interioare) și cheltuieli curente (lunare - chirii, salarii, curent electric, personal administrativ - femeie de serviciu, secretara -, alte cheltuieli de regie).

Cât despre părțile mai plăcute care apar ca urmare a acestui proces am să vă las să descoperiți singuri cât de bine este să rupi din bugetul alocat cut-scene-urilor pentru a putea angaja încă un programator pentru că altfel nu termini partea de collision detection la timp, sau să descoperi că ai estimat greșit cheltuielile cu amenajări interioare și deci trebuie să ceri alți bani, deci alte negocieri, altă distracție...

Proiectul are mult mai multe nivele de planificare, nivele ce includ: termenele limită pentru anumite bucăți din engine-ul grafic, cele pentru grafică, animații, cut-scene-uri, cele pentru pregătirea unui demo, cele ce țin de relația cu distribuitorul, etc. Nerespectarea acestor termene poate avea consecințe din cele mai grave, până la oprirea totală a proiectului. În particular fiind spus, acesta este unul din motivele pentru care absolut orice investitor devine foarte „nervos", deci este un lucru de care mai bine te ferești și cu cum poți: o soluție este să dublezi orice termen la care te-ai gândit mai întâi și să îl folosești pe acesta drept punct de plecare pentru business plan. O soluție, da, dar evident nu cea mai bună. Altă soluție ar fi să îți faci niște planuri foarte exacte, care să acopere cât mai mult din întregul proiect. Aceste planuri trebuie realizate folosind două „axe": timpul și echipa. Adică pe baza unor tabele uriașe să se poată ști în orice moment cine - ce face, unde, și, mai ales, de ce. Un „parametru" mai ciudat va fi: „și altele" - adică lucruri de care ești sigur

Alexei Pajitnov, matematician rus, este cel care a inventat Tetris-ul. Astăzi este unul dintre angajații de seamă ai Microsoft-ului.

Brett Sperry este unul dintre fondatorii Westwood Studios, și cel care a creat seria Eye of the Beholder.

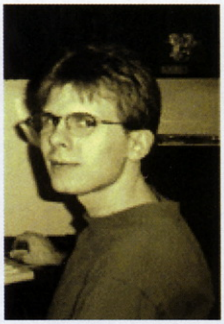


Peter Molineaux, cel care a creeat Populous, primul god-game din istorie și co-fondator al firmei Bullfrog

Richard Garriott, mult mai cunoscut sub pseudonimul Lord British, este creatorul serie Ultima.

Roberta Williams, unul din părinții adventure-urilor, este cea care împreună cu soțul ei, Ken, a înființat Sierra On-Line.

Deși nu este un nume foarte cunoscut, Shigeru Miyamoto este cel a dat viață unor personaje ca Mario, Zelda și Donkey Kong.



Nu putea lipsi din acest tablou al celebrităților, John Carmack. Știți deja: QUAKE!!!

Și am ajuns si în vârful piramidei, acolo unde îl întâlnim pe Sid Meier, cel care a avut cea mai puternică influență în această lume controversată a jocurilor de PC. Ideile sale inovatoare au fost preluate, șlefuite și aplicate de aproape toți producătorii moderni.

că te vei izbi și nu poți aprecia exact consecințele impactului acestora asupra lucrului efectiv. Durata acestor timpi morți, cât și rezolvarea aleasă trebuie date ca soluții alternative. Asta ar vrea să zică ceva de genul : dacă collision detection-ul nu merge, nu este nimic grav, pentru că îl putem ruga pe jucător să nu treacă prin pereți.

O altă problemă sensibilă este distribuitorul. Degeaba faci un joc dacă nu are cine să ți-l vândă. Iară, ca unul din „greii" industriei să accepte să vorbească cu tine, trebuie să ai mai întâi ce-i prezenta. Acest nou element nu face decât să sporească gradul de risc al afacerii și, din această cauză aceasta, problema va fi una îndelung dezbătută cu cei ce vor să investească.

Să creezi un business plan consistent și corect din toate punctele de vedere nu este însă de ajuns. Am descoperit curând că și prezentarea acestui plan de afaceri este o mare problemă. Parțial datorită auditoriului, parțial datorită „prezentatorului". Ceea ce vrem să spunem ține de interesele diferite pe care cele două părți le au: tu vrei să faci un joc și să fii întrebat despre acesta (game play, game design, technologies) și de fapt ți se pun o groază de întrebări despre tot felul de ciudățenii (cote de piață, profitabilitate, concurență, termene limită, mumbo jumbo...). Evident, soluția este să îmbini utilul cu plăcutul și dacă reușești să faci o impresie bună și să destinzi atmosfera s-ar putea să ai noroc.

Oare? Cu toții (membrii echipei FUN labs) suntem de acord că nu îți poți da seama ce implică un asemenea proiect și cât este de greu decât odată cu lucrul efectiv. Vă garantăm că va fi mult mai greu decât v-ați imaginat inițial. Se muncește non-stop (a se citi exact cum s-a scris). Trebuie să renunți la viața particulară (cel puțin în cazul nostru așa a fost). Concluzia: stres maxim.

Pe lângă asta, invariabil veți avea parte de porția voastră de critici. În cazul nostru, forumul ne-a oferit atât multe încurajări cât și destule critici - mai puține decât laudele, zicem noi. Când l-am conceput, l-am văzut ca pe un ajutor atât pentru noi cât și pentru cei pasionați de domeniu. Deși au fost mulți cei care ne-au încurajat și ne-au dat sugestii utile - trebuie să le mulțumim și pe această cale - ne-a rămas totuși un gust amar datorită superficialității și modului de gândire simplist al altora. Spunem asta pentru că acele critici proveneau din examinarea superficială și unilaterală a forumului - pe care noi l-am vrut ca o sursă de inspirație, și unde în locul discuțiilor despre ce ne place și ce nu ne place la un joc 3D action-adventure, am descoperit că aproape toți cei ce ne-au criticat doreau doar să devină testeri (yeah, ce slujbă bună... să iei bani fiindcă te joci...) sau să critice un joc deja făcut și nicidecum să participe la CREAREA unui joc.

Însă medalia are și un revers. Provocarea intelectuală este foarte mare și dacă suntem capabili să „ținem ritmul", la sfârșit vom putea spune că ne-am îndeplinit un vis: „Facem jocuri."

Dacă a-ți avut răbdare să ajungeți să citiți aceste rânduri înseamnă că sunteți pregătiți pentru numerele viitoare în care vom încerca să vă prezentăm, în linii mari, procesul de producție al fiecărui segment în parte al jocului; grafica, sunetul, engin-ul, AI-ul, ș.a.m.d. Inițial intenția mea a fost aceea de a culege cât mai multe date posibile și, din cele asimilate, să încerc să încropesc această idee mai veche de-a mea. Domnul Adrian Filippini a fost totuși atât de amabil și a avut o răbdare de fier cu mine, astfel că efectiv mi-a fost milă să mă introduc și eu în rândurile de mai sus, preferând să vă redau exact cuvintele interlocutorului meu. Până data viitoare, când voi reveni cu noi detalii, baftă și JUMP TO THE NEXT LEVEL!!!

Claude

Focul - Zeul Mileniului Trei

După cum v-am promis, ne întoarcem cu noi amănunte legate de procesul de creare a unui joc.

Numărul trecut am început un articol, un proiect uriaș ce se va întinde pe mai multe numere ale revistei. V-am spus și atunci și o repet încă odată, toată această muncă o depun în speranța că veți realiza ce efort se depune pentru producerea unui joc. De multe ori, în controversele mele cu prietenii, chiar și unii dintre cei mai înverșunați gameri, am observat cât de puține lucruri știu referitoare la acest subiect. Am prieteni care știu nivelele de la *Quake* pe de rost, le vi-seză și noaptea, am alții care-ți pot enumera toate trupele din *Panzer General* cu slăbiciuni și avantaje cu tot. Dar nici unul dintre ei nu știa câtă muncă s-a depus pentru realizarea acelor nivele de *Quake* sau pentru adunarea documentației necesare unui joc precum *Panzer General*. De aceea am ținut eu neapărat să tratăm și acest subiect, pe de o parte pentru satisface curiozitatea multora legată de acest proces de producție, pe de altă parte pentru a scoate în evidență cantitatea imensă de muncă depusă de acești oameni doar pentru ca noi să ne putem distra mai bine cu calculatorul. Bineînțeles că aceste aprecieri ale noastre, adică ale utilizatorilor, constituie doar o parte din satisfacția personală a producătorului, cealaltă părticică reprezentând-o acele bancnote verzi ce trec din buzunarele noastre în ale lor.

Buuuuuun... Data trecută am vorbit de fazele preliminare ale producției, constituirea unui plan de producție. Dacă nu v-am plictisit atunci și mai sunteți încă alături de noi, vă vom prezenta, în această a doua parte, cum se face organizarea activității și a echipei (echipelor, după caz). Din nou am apelat la cunoștințele și experiența domnului **Adrian Filippini**, Project Manager **FUN Labs**.

«Suntem convinși că majoritatea celor care au destulă răbdare cu noi și citesc acest articol au visat să facă un joc sau plănuiesc să înceapă lucrul la unul. Așa că ne simțim datori să vă explicăm ceva: neexistând nici un fel de documentație veritabilă cu privire la ce înseamnă crearea unui joc, tot ceea ce vă povestim aici este venit din propria noastră experiență. Acestea sunt concluziile noastre, învățate „the hard way", prin încercări și greșeli succesive. Nu există „instrucțiuni" care să-ți arate cum să faci un joc, cum să definitivezi project management-ul, sau business plan-ul, sau cum să construiești grafica, sau cum să creezi un engine cu posibilitățile *Quake*-ului. Tot ceea ce poți spera este ca, după ce îți pierzi nopțile cercetând articole, cărți sau documente, să ciupești de icicolo câte o bucățică și punându-le cap la cap să încerci să obții „the big picture". Practic, ceea ce facem noi pe tot parcursul acestui proiect, și ceea ce ar face oricine altcineva în situația noastră, care s-ar apuca de așa ceva, este mai apropiat de activitatea de cercetare și de cea de creație decât de cea „standard" de scriere de soft.

Cu speranța că aceste informații vă vor fi de folos și că ele vă vor ajuta să vă faceți o imagine de ansamblu asupra a ceea ce vă așteaptă, și, mai ales, că veți putea folosi aceste date pentru a pleca cu un avantaj pe drumul ce duce la împlinirea visurilor voastre (acest lucru însemnând și că această serie de articole și-a atins scopul), vom continua în acest număr prin a vă prezenta modul efectiv în care lucrăm noi.

Ei bine, în numărul trecut am prezentat modul în care puteți scrie un business plan care să vă ajute să obțineți o sumă cu care să începeți. Să zicem că am obținut banii... Acum ce facem cu ei? Dacă ar fi după ce am vrea noi, ne-am lua vreo douăzeci de computere meseriașe, și am juca *Quake* și *Half-Life* într-o veselie. Numai că lucrurile nu stau chiar așa... Totuși, de început cam așa am început,

chiar și numai alegerea configurațiilor hardware fiind extraordinar de sâcâitoare și mare consumatoare de timp și nervi. Acum însă trebuia să respectăm ceea ce promisesem în acel Business Plan cu care v-am tot bătut la cap. Așa că ne-am pus pe treabă și am început să strângem echipa. Un joc nu se poate face altfel decât în echipă, așa că practic, acesta este începutul: „harvesting people".

Primul pas ar fi să știi ce cauți – adică trebuie să ai o organigramă pe baza căreia să construiești fișa fiecărui post. De aici încolo vă vom prezenta lucrurile mult mai la obiect, mai mult descriind ceea ce s-a întâmplat cu/la noi.

Am fi vrut să vă prezentăm echipa în această ordine pentru a sublinia mai bine și evoluția „închegării" acesteia. S-a început cu un nucleu de bază, care a fost completat pe măsură ce acesta a început să funcționeze, conform necesităților ce au apărut. Datorită faptului că în evoluția unui joc există mai întâi o parte ce se numește „Technology Implementing" ne-am concentrat mai întâi pe acest domeniu, ce poate fi descris mai pe scurt ca engine-ul jocului, lucru observat și în ponderea mai mare a programatorilor pe care am avut-o la început. Cu această ocazie a trebuit să stabilim foarte clar dacă urma să implementăm un engine mai general – care ar fi avut avantajele sale, cel mai important părând a fi cel al comercializării acestuia, adică o posibilă sursă de venit suplimentară – sau să particularizăm acest prim engine la nevoile actuale, reducându-i acestuia atât gradul de portare cât și complexitatea. Datorită faptului că ne aflam la început, am preferat să alegem cea de-a doua variantă, mult mai sigur de realizat, considerând că, de exemplu, pentru a ajunge să scrii un engine general cum ar fi să zicem NetImmerse-ul sau unul de succes cum ar fi cel al lui **ID Software** trebuie să ai ceva experiență în spate. Ca să nu mai vorbim de dificultatea portării unui joc pe alte mașini sau sisteme de operare cum ar fi Mac, Linux, Playstation sau Nintendo. Așa că decizia o dată luată (ținând cont evident de multe alte lucruri – vezi articolul precedent – joc 3D first-person pentru platformă Windows cu suport hardware DirectX și OpenGL) am purces la întocmirea, în paralel, atât a documentației de design (adică un document de uz intern în care îți stabilești ce se vrea a se face în și cu acel joc) cât și a organigramei firmei (de câți oameni ai nevoie și ce anume trebuie să știe să facă aceștia).

Fiecare dintre noi face cel puțin două job-uri având însă unul prioritar (mai multe detalii puteți vedea vizitând pagina noastră web). Pe lângă asta, așa cum am spus mai sus, pe măsură ce am început să lucrăm efectiv la acest proiect am constatat ce lipsuri și ce nevoi avem și am încercat să rezolvăm aceste probleme. Astfel, dacă la început am avut o structură organizatorică bine definită (Project Manager+Project Administrator; Lead Programmer+Technologies Leader; Storyteller) aproape imediat am căutat (și am găsit) un Level Designer + Lead 3D Modeller, urmați imediat de un număr de programatori, artiști și modelatori. „Piramida" a fost astfel completată de la vârf în jos, de fapt așa fiind și logic. Un alt lucru interesant pe care l-am constatat în această primă perioadă a ținut atât de structurarea în etape a proiectului cât și de ordonarea/organizarea acestuia. Astfel voi înșirui mai jos câteva din lucrurile pe care le-am învățat până acum noi, cei de la **FUN Labs**:

- nu vă așteptați să rezolvați problema echipei într-un timp bine stabilit; nu se poate - este un proces ce durează și durează chiar al naibii de mult. Cât a durat la noi? Șase luni până am strâns echipa plus încă trei necesare pentru ca membrii echipei să învețe să lucreze împreună. Aici am inclus și timpul necesar acomodării acestora cu uneltele pe care le folosim (programatorii au avut de luptat cu înțelegerea API-urilor impli-

cate, DirectX și OpenGL, iar graficienii și modelatorii au avut de furcă atât cu uneltele folosite – cum ar fi editorul nostru propriu de texturare – cât și cu unele animale mai năbădioase cum ar fi mouse-ul, tăbliță grafică sau driverele unor plăci video);

- cum și de unde îți alegi oamenii ? Păi, să începem mai întâi cu „de unde". Aici trebuie să aveți o mică-mare discuție cu zeița Fortuna. Căci un răspuns corect ar fi: Dumnezeu știe ! Cert este că dacă vrei, poți. Sursele noastre s-au numit: Facultatea de Automatică și Calculatoare - UPB, Institutul de Arhitectură Ion Mincu + Colegiul de Design, Facultatea de Cibernetică - ASE, cenaclul SF Planetar, forumul FUN labs (vezi www.funlabs.com secțiunea forum), precum și străzile Bucureștiului... just kidding, of course. Uite așa am înțeles noi de ce este atât de mare lista aceea de „special thanks to ..." la sfârșitul unui joc. Revenind la „cum": este destul de greu să găsești personal calificat pentru ceea ce avem noi nevoie. Gândiți-vă că la noi în țară nu există o experiență anterioară în domeniu. Iar ideea acestui proiect este să facem ceva autohton, „importul" unor specialiști de afară fiind exclus, din cauze, zic eu, evidente. Deci, iată încă o mare problemă care mărește atât timpul de dezvoltare a proiectului cât și dificultatea acestuia. Din această cauză, această problemă a echipei ne-a luat atât de mult timp. Cunoștințele de bază pe care acești oameni trebuie să le aibă sunt doar un punct de pornire, în primă fază aceștia muncind foarte mult pentru a se integra într-un colectiv și pentru a deveni eficienți. Colac peste pupăză, majoritatea nici nu au mai lucrat într-un colectiv până atunci, așa că să te ții discuții ... (exemple: obișnuiește programatorul să nu se apuce să facă el un obiect de care are nevoie, ci să îl ceară modelatorului, acesta să vorbească cu artistul care să se înțeleagă repede cu scenaristul, cu level și game designer-ul). Pe deasupra, cineva trebuie să coordoneze, fără să se contrazică, acest ... haos, pardon, stil de lucru. Evident am exagerat un pic doar pentru a vă ajuta să vă faceți o idee, adică tocmai scopul acestui articol, nu ?

- voi face și o precizare specială ce privește forumul: este unul din lucrurile care m-au intrigat cel mai mult. Unii oameni au fost entuziaști, alții numai cu critica, ceva intermediar aproape că nu există. Dar una peste alta este foarte bine, căci ne-a adus și multe lucruri bune: acolo am întâlnit câțiva oameni care acum lucrează cu noi, acolo ne ducem să citim o laudă atunci când vrem să ne simțim bine și tot acolo ne ducem când vrem să vedem ce îi deranjează pe unii. În concluzie, a fost o experiență utilă, deși cam traumatizantă pentru mine.

- echipa ar trebui să fie cât mai stabilă; datorită faptului că, practic, așa ceva este imposibil ca urmare a condițiilor actuale din România, trebuie menținut măcar un nucleu stabil al acesteia, urmând ca pe parcurs să încercăm să cooptăm noi membri sau să-i schimbăm pe cei existenți, atunci când această schimbare se impune. Aici vreau să fac două comentarii importante:

1. principiul de bază al economiei de piață (concurența naște progres – uite că am zis-o !) trebuie aplicat și aici, dar cu măsură, pe cât posibil fără exagerări, competența și eficiența celor ce lucrează împreună fiind atât două dintre criteriile de bază prin care se face selecția acestora la început, cât și criteriile prin care se stabilește aportul acestora la echipă, respectiv menținerea, avansarea sau schimbarea activității lor;

2. un al doilea lucru foarte important ține de specificul activității noastre: echipa! Am avut cazuri de oameni foarte buni care nu au putut stabili niște relații tocmai cordiale cu ceilalți membri ai echipei (ca să spun numai atât) fapt ce a dus la câteva situații delicate. Alt caz întâlnit este cel al unor persoane capabile care pur și simplu nu au făcut față ritmului de lucru impus sau, așa cum am mai zis, cantității de stres „generate" în urma procesului de producție. Concluzia ? Este destul de greu să iei anumite decizii, dar, forțat de împrejurări, ești obligat să faci ce trebuie, nu ce vrei, și astfel, trebuie să alegi. Se pare că cea mai bună decizie este totuși să păstrezi întregul ... dacă pot spune așa, dar asta nu vom ști cu siguranță decât la sfârșit ... indiferent care va fi acela. Pe scurt, deși este destul de dur să spui așa ceva, o echipă strâns unită, în care membrii acesteia se înțeleg foarte bine și lucrează eficient împreună este mai valoroasă decât un cumul de genii care pierd jumătate de zi cu certuri.

- câțiva dintre membrii echipei sunt studenți (evident nu cei din nucleu, pur și simplu nu ar fi posibil ...), lucru ce influențează direct orarul și modul de lucru al întregii echipe, devenind evidente câteva cerințe/constrângeri. Astfel, un lucru inițial impus din exterior, (în cazul acesta orarul facultății) că doi graficieni ce lucrează part-time pe același computer, a devenit mai apoi o necesitate și prezintă chiar avantaje considerabile (exemplificare: o varietate mai mare a schițelor, o acoperire mai bună a zilei de lucru).

- datorită motivelor prezentate mai sus se ajunge la următoarea realitate crudă: acomodarea unui „rookie" durează cel puțin o lună până ce acesta începe efectiv să lucreze cu noi, sau, dacă preferați, să „producă" efectiv. Evident asta îndeamnă la o mare prudență atât în alegerea oamenilor cât și în luarea deciziilor de înlocuire sau abandonării cuiva; credeți-mă mai durează cel puțin o lună să găsești pe altcineva și încă una până se integrează. Personal cred că datorită complexității crescânde a proiectului, plecarea unui om din echipă poate întârzia considerabil proiectul, mai ales dacă se petrece la mijlocul acestuia sau spre final.

- care este atunci concluzia firească a lucrurilor prezentate mai sus ? Rezultă că pentru optimizarea activității e imperios necesar un grad de organizare ridicat al proiectului. Și uite așa cădem în birocrație. Structura arată cam așa – mai întâi centrul: documentația de design, care stabilește ce se vrea. Urmează apoi, în paralel :

- implementarea tehnologiei: grafica 3D, partiționarea spațiului, collision detection, fizica lumii, AI, sunet, muzică,

networking, tools-uri pentru modelatori și graficieni. La programare se ordonează engine-ul pe module apoi pe funcții și se construiește toată documentația aferentă. Prezența documentației este obligatorie pentru eliminarea bugurilor și pentru testare. Ce te faci peste o lună dacă vrei să citești ce făcea sau cum făcea o funcție ? Ce te faci când un membru al echipei vrea să se pună la punct cu ce au făcut ceilalți ? Și exemplele pot continua. În plus, astfel poți repartiza mai ușor oamenilor anumite task-uri precise: tu și tu faceți aia, tu depanezi, tu testez etc. Iar dacă faci asta poți avea un control asupra a cât se lucrează și în cât timp. Astfel poți estima durata proiectului, poți observa rapid ce probleme apar, poți interveni rapid etc.

- scenariul, cu toate elementele acestuia: plot – adică gradarea treptată a poveștii și modul în care îi este dezvăluită aceasta jucătorului, unde, cât și când, existând elemente ce preced jocul și țin de publicitate (vezi trailer, demo, beta testing, forum, chat, reviste de specialitate etc.), precum și elemente ce țin de cursul jocului (când îi spunem asta, când îi dezvăluim asta ... uite acu' vine cut-scene-ul, etc.) – apoi dialoguri, voci, sunete, muzică, ambianță, atmosferă, elemente ce țin de mediul lumilor respective (asta mai ales dacă vrei să ai o grijă pentru detaliu și o poveste adevărată, așa cum vrem noi): iconografia – semnele și simbolurile, designul vestimentar, habitatul, religiile, legenda, rasele, oponenții, arhitectura, până și culoarea cerului sau momentul din zi când se petrece acțiunea. Noi susținem că absolut toate aceste lucruri contează și cu siguranță îmbunătățesc atmosfera jocului. Astfel, chiar dacă nu vom reuși să avem cine știe ce grafică, FPS, AI sau game-play pentru că facem asta pentru prima oară, măcar să fim siguri pe ceea ce putem face: atmosferă și poveste.

- strâns legate urmează animațiile, modelarea, designul de nivel, efectele speciale, multitudinea de obiecte dintr-un joc, texturile, personajele, interfața grafică, cut-scene-uri. Iar pentru toate acestea se face de asemenea câte o fișă !!! Avem astfel: traseul jucătorului prin nivel, traseele oponenților, câteva perspective, fișe pentru fiecare nivel (conține: scop final, trasee, plot, mediu, clădiri, personajele, arme, item-uri), fișe pentru clădiri mai importante, o fișă generală de mediu, fișe pentru fiecare pesonaj (conține: rol, arme folosite, mobilitate, inteligență, rezistență, nr. de poligoane, nr. de skinuri, calitatea texturii, nr. de animații, sunet). Evident toate aceste fișe au și câte o rubrică de observații sau alte detalii.

- partea negativă ? Se pierde totuși foarte mult timp cu asta la început: orar pentru fiecare om pentru toată săptămâna următoare + cine, ce face, când + urmărirea rezultatelor = stres total ! Dar reamintim avantajele: poți controla munca, o poți cuantifica, poți interveni rapid dacă ceva merge prost, designul de joc merge mult mai repede când ai fișele complete în mână, iar poate cel mai important – poate absolut oricine să înțeleagă ce s-a făcut acolo, adică simplitate în comunicare. Dar mai este ceva. Bomboana de pe colivă. La toate astea poți sta o groază de timp întrebându-te: o fi bine, o fi rău ?

Cam așa lucrăm noi. Am făcut un calcul. Lucrăm în medie 70 de ore pe săptămână. Deci este greu. Foarte greu. Mai greu decât ne-am imaginat la început. Dar asta nu puteți afla cu exactitate decât într-un singur fel: apucați-vă și voi de ceva asemănător și veți vedea. Nu vrem să vă speriem. Vrem să vă spunem pur și simplu ceea ce am învățat noi lucrând la asta. Dar nouă ne place și vrem să ducem asta până la capăt. Rămâne să vedem dacă o să și putem să facem asta.»

Uh! Și noi care credeam că e atât de simplu, o mână de oameni ce se strâng în jurul unui calculator, își dau cu părerea și apoi încep să încropească ceva. Și mai stăm și ne uităm la marile companii care angajează sute de oameni doar pentru un singur proiect. Fiindcă sunt încă enorm de multe alte domenii neacoperite în cele prezentate de domnul **Filippini** și cărora în cadrul acestor mari producători li se acordă o atenție deosebită, cum ar fi promotion-ul produsului. Sunt oameni angajați doar pentru a găsi o lozincă potrivită acelui joc. Din fericire, noi românii, suntem multilateral dezvoltați și, așa cum și domnul **Filippini** amintea la un moment dat, fiecare membru al echipei are cam 2-3 atribuții în cadrul firmei. Ce fac ei, cum lucrează, cu ce lucrează sunt întrebări ale căror răspunsuri le veți găsi în numerele viitoare. Așa că fiți pregătiți pentru un: Jump to the Next LEVEL!

Claude

Jocul - Zeul Mileniului Trei

Lumea atât de fascinantă a jocurilor pe calculator vi se dezvăluie cu un mic ajutor din partea noastră

În articolele precedente am arătat care sunt etapele preliminare ale creării unui joc, apoi cum se poate încerca obținerea fondurilor și cum se face constituirea echipei; acum a venit momentul să trecem la treabă...

În cele ce urmează vom începe prin a vă prezenta modul în care se alege și se construiește „baza" oricărui joc 3D, și anume motorul grafic, engine-ul 3D, cel ce permite reprezentarea lumii jocului. Ne vom concentra în acest articol asupra uneia din primele componente esențiale ale unui engine grafic 3D - algoritmul de partiționare a spațiului, componentă mai puțin cunoscută și înțeleasă de cei ce nu lucrează efectiv la crearea unui joc. Pe măsură ce veți citi acest articol sunt sigur că veți observa importanța covârșitoare a acestui algoritm, atât pentru engine-ul jocului cât și pentru joc (evident, există, așa cum am mai arătat, o strânsă legătură între tipul de engine și tipul de joc).

Pentru a avea o imagine de ansamblu a acestui subiect și pentru a arăta mai clar unde se situează algoritmul ales de dezvoltatori în cadrul acestuia, vom începe cu o scurtă recapitulare.

1. Noțiuni generale

Engine-urile constituie practic „sufletul" oricărui joc 3D și al uneltelor specifice level designerilor și artiștilor, unelte cu ajutorul cărora se creează nivelele. Cu el se asigură funcțiile de rendering, împărțire a spațiului, collision detection, inteligență artificială, animații, efecte speciale etc. Din punct de vedere conceptual dar și al destinației lor engine-urile se împart în două categorii:

1.1. Engine-uri bazate pe poligoane

Aceste engine-uri folosesc accelerarea 3D a plăcii grafice, constituie marea majoritate și se împart la rândul lor în două subtipuri:

1.1.a. Engine-urile optimizate pentru exterior

Acestea se preocupă mai mult de crearea unor suprafețe mari, mai puțin detaliate, iar pentru a reprezenta cât mai realist terenul, au nevoie de un număr foarte mare de triunghiuri, cu mult peste capacitatea unui sistem actual. Ca urmare aceste engine-uri folosesc o serie de tehnici pentru a reduce numărul de poligoane afișate la un moment dat.

Una dintre aceste tehnici este ceața, care limitează vizibilitatea la distanță și astfel se randează numai zona din apropierea jucătorului, cu un grad mai mare de detaliu. Altă metodă constă în stabilirea dinamică a nivelului de detaliu pentru teren (LOD – Level Of Detail). Triunghiurile scenei se generează în timp real, astfel încât în apropierea jucătorului terenul are un număr mare de poligoane (suprafețe netede etc.), iar pe măsură ce distanța față de jucător crește, suprafața e aproximată prin mai puține triunghiuri.

Nu ne credeți? Vreți exemple? Ei bine, din titlurile dezvoltate cu un astfel de engine se distinge cu pregnanță un nume: *Mech Warrior.*

1.1.b. Engine-uri optimizate pentru interior:

Se bazează pe structura închisă a universului 3D (chiar dacă uneori reușesc să dea senzația de spațiu mare, vezi Unreal), unde cea mai mare parte a geometriei este obturată de pereți și nu se afișează. Astfel pot fi folosite multe poligoane pentru reprezentarea detaliilor. Astfel de motoare grafice au fost folosite pentru jocuri de mare succes ca *Quake, Half-Life* sau *Unreal.*

Motivele pentru care s-a decis folosirea unui astfel de engine constau în bogăția de amănunte pe care dorim să o avem în joc, controlul pe care îl putem avea în special asupra game-play-ului și nu în ultimul rând ușurința în implementarea ceva mai puțin dificilă.

După cum veți vedea în continuare, algoritmul de partiționare a spațiului este un element specific și esențial al acestui tip de engine.

Mai rămâne de precizat că unele jocuri folosesc două engine-uri, unul de interior și altul de exterior, care se comută în funcție de necesitate, evident acest lucru făcându-se în general, cu o penalitate dată de timpul mare de loading al unui nivel.

1.2. Engine-uri bazate pe voxeli

Termenul „voxel" provine de la „volume element" (element ce aproximează un volum, așa cum un pixel aproximează o suprafață; altfel spus, un voxel este unitatea de bază de reprezentare a volumelor sau dacă preferați un pixel volumetric).

Datorită caracteristicilor specifice, aceste engine-uri pot fi utilizate optim pentru simularea suprafețelor exterioare, acest lucru fiind posibil datorită existenței unor restricții foarte mici în simularea reliefului neregulat. Astfel, nefiind folosite poligoane, importanța folosirii unui LOD dinamic sau a unui algoritm pentru determinarea poligoanelor „ascunse" este mai mică. Poligoanele ascunse sunt acele poligoane invizibile din punctul în care se află jucătorul pe direcția privirii acestuia, iar algoritmul de

„ascundere de feţe" este unul ce dă mare bătaie de cap celor ce creează un engine de interior. În plus, gradul de detaliere al lumii jocului, mai exact al elementelor situate la mare depărtare de jucător, este foarte mare, astfel engine-ul permiţând crearea unor peisaje foarte frumoase (vizibilitate la foarte mare distanţă) şi având încorporată şi o funcţie de zoom cu adevărat spectaculoasă.

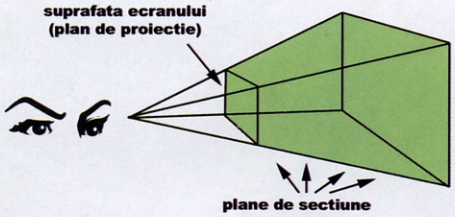
Aceste engine-uri se folosesc numai de procesor, ignorând facilităţile acceleratoarelor grafice 3D, şi sunt mult mai puţin întâlnite, poate şi datorită dificultăţii sporite a implementării (aparatul matematic utilizat este cu mult mai complicat). Exemplul clasic al unui astfel de engine este jocul *Delta Force*.

Engine-urile de exterior bazate pe voxeli folosesc „hărţi de înălţimi". O hartă de înălţime e reprezentată ca o matrice (n x n) în care fiecare element are un atribut de înălţime şi de culoare. În funcţie de valorile acestor elemente se generează voxelii. Algoritmii de vizualizare sunt mai puţin cunoscuţi (documentaţi) şi se bazează în general pe tehnici de ray casting (se trasează raze de vizualizare pornind din punctul de observaţie al jucătorului şi se testează ce obiecte au fost intersectate).

2. Procesul de rendering

Nivelele sunt reprezentate în memoria calculatorului ca o colecţie de triunghiuri într-un spaţiu tridimensional. Pentru a le afişa pe ecran, asupra acestora se aplică o serie de operaţii care transformă reprezentarea tridimensională într-o proiecţie plană, 2D (aşa cum se face de exemplu proiecţia unui cub pe trei plane, date de cele trei axe „standard" x,y,z la desen tehnic). În final, pe triunghiuri se aplică texturile corespunzătoare, ţinând cont de distanţa şi orientarea faţă de privitor a triunghiurilor. Sunt texturate şi afişate pe ecran numai poligoanele care se încadrează în „câmpul vizual" al jucătorului. Acest câmp vizual cuprinde numai o parte din întregul nivel, şi anume o „piramidă" al cărei vârf se află undeva în faţa ecranului (unde se află user-ul, dacă vreţi) şi ale cărei laturi sunt delimitate de marginile ecranului. Calculul proiecţiilor şi afişarea poligoanelor pe ecran implică deci o decupare a triunghiurilor care intersectează planele celor patru laturi ale acestei piramide. Întregul proces descris mai sus poartă numele de rendering.

Procesul de rendering aplicat unei scene foarte complexe solicită mult procesorul şi placa grafică, limitând foarte mult frame-rate-ul (numărul de cadre afişate pe secundă). A apărut astfel necesitatea de a reduce numărul de calcule, astfel încât să nu se mai proceseze întregul nivel, ci numai poligoanele potenţial vizibile din poziţia curentă a jucătorului. Astfel au apărut algoritmii de partiţionare a spaţiului.



3. Partiţionarea spaţiului

Nivelele unui joc sunt formate din poligoane şi texturi. Afişarea acestora pe ecran presupune o serie de calcule prin care se transformă structurile 3D din memorie în imagini 2D (ştiu că mă repet, dar e un lucru foarte important). Acest proces este denumit rendering şi este mare consumator de resurse. Cum publicul cere lumi din ce în ce mai vaste, cu cât mai multe amănunte, dar şi o mişcare fluentă în joc, este imposibil să se efectueze calculele pentru un întreg nivel în acelaşi timp.

Pe scurt, un algoritm de partiţionare a spaţiului nu face altceva decât să ofere o structură de organizare a datelor prin care să fie prelucrate (randate) numai zonele potenţial vizibile. Acest algoritm se aplică în faza de preprocesare a nivelului (nivelele se includ în joc abia după ce sunt astfel prelucrate pentru a se adăuga informaţiile suplimentare legate de organizarea datelor).

Pentru ca totul să fie mai clar, iată un exemplu. Dacă jucătorul se află într-o cameră a unui nivel, cu ajutorul algoritmului de partiţionare a spaţiului programul va şti să proceseze numai acea cameră (cu toate elementele din ea) şi camerele potenţial vizibile din ea (cele care sunt accesibile printr-o uşă sau o fereastră). Pentru a nu exista scăderi ale frame-rate-ului de la o zonă la alta, designerului de nivel îi revine sarcina de a menţine constant numărul de poligoane existente într-o scenă.

Există doi algoritmi utilizaţi în mod curent pentru partiţionarea spaţiului. Primul dintre aceştia este algoritmul bazat pe arbori BSP (de la Binary Space Partitioning).

3.1 Metoda arborilor BSP

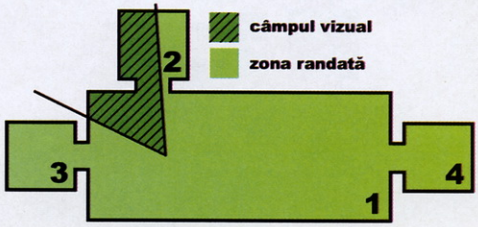
Reprezintă o metodă prin care dintr-o colecţie de poligoane se obţine un arbore binar cu ajutorul căruia se poate stabili o ordonare din spate în faţă a poligoanelor scenei, din orice poziţie a privitorului. Pe scurt, matematică la greu fraţilor, dar acest lucru cred că este deja clar, nu ? Absolut orice abordare alegi algoritmii de bază implică nişte cunoştinţe temeinice şi foarte serioase de matematică, cunoştinţe concretizate sub forma unor cărţi foaaaarte groase L Cine spune că mărimea nu contează ? Folosind această ordonare se poate stabili faptul că unele poligoane sunt obturate de altele (deci nu trebuie afişate).

Algoritmul de construire a arborelui presupune că pentru planul fiecărei feţe a scenei, toate celelalte poligoane sunt fie în faţa fie în spatele acelui plan. Acest lucru implică „spargerea" poligoanelor care nu se supun acestei reguli, deci a celor care sunt intersectate de plan. Această structură oferă în sine doar o ordonare a poligoanelor, şi capătă utilitate în cadrul algoritmului de partiţionare doar prin ataşarea la arborele BSP a unor zone potenţial vizibile (PVS = Potentially Visible Set). Aceasta înseamnă că la o parcurgere a arborelui nu se desenează poligoane în mod independent, ci zone convexe formate din poligoane potenţial vizibile din acea poziţie. În exemplul de mai jos celulele 1,2,3 şi 4 formează un PVS pentru cazul în care jucătorul se află în camera 1. După cum puteţi vedea, celulele sunt randate chiar dacă jucătorul se află cu spatele la ele :

3.2. Metoda portalilor

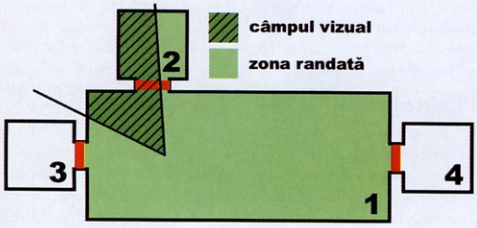
Producătorii au început să aleagă această metodă datorită avantajelor pe care aceasta le oferă faţă de clasica tehnică a BSP-ului. Trebuie menţionat totuşi că acest algoritm este cam ultima găselniţă a matematicienilor, lucru dovedit atât de puţina documentaţie disponibilă pentru acesta cât şi de existenţa unui singur joc care îl foloseşte : Requiem – The Avenging Angel. Astfel, decizia alegerii acestui algoritm nu a fost chiar una uşoară, aş spune chiar riscantă.

Să vedem acum cum funcţionează şi de ce este atât de bun. Pornind de la arhitectura nivelului se realizează o partiţionare a acestuia într-un sistem de celule şi portali. Celula este echivalentul unei camere dintr-o clădire, iar portalul echivalentul spaţiului gol al uşii. Celulele sunt conectate între ele prin portali, formând o structură de tip graf, unde în noduri se află celule iar arcele reprezintă portali. Ideea de vizualizare a acestei structuri este următoarea: pornind din celula în care se află jucătorul, se determină portalii vizibili din celula curentă şi toţi portalii vizibili prin ei (implicit celulele conectate prin portali). Tehnica aceasta presupune



calcule de intersecție în timp real, ceea ce poate determina apariția unor limitări în construcția nivelelor, pentru a elimina situații de exploatare intensivă a resurselor, ca atunci când sunt vizibili recursiv mai mulți portali (imaginați-vă că priviți de pe un hol o fereastră prin care se vede clădirea vecină unde o altă fereastră dă spre un alt hol ș.a.m.d.... :). Unul din primele avantaje evidente al acestei tehnogii a portalilor este că practic se randează numai celulele pe care jucătorul le vede efectiv.

Evident, pentru a îl optimiza, și acest algoritm are nevoie de o structurare suplimentară de ordonare a camerelor/celulelor la un nivel macro al nivelului, gen PVS. Combinația dintre algoritm și felul în care



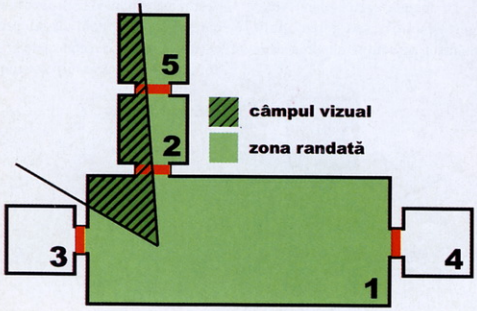
se face level design-ul trebuie pe de-o parte să evite cazul de mai jos de folosire excesivă a resurselor mașinii (cu penalitate imediată in ceea ce privește fps-ul „Frame Per Second"), adică să evite randarea a mai multor celule, iar pe de altă parte să permită artiștilor echipei să se „dezlănțuie" în interiorul celulei, evident în ceea ce privește grafica și arhitectura acesteia. Astfel acestora le este lăsată o mult mai mare libertate în privința formelor și a decorurilor, singura lor grijă fiind fps-ul, sau dacă vreți „conturul" și poziționarea celulelor. Acest fapt permite o bogăție de detalii mai mare, un univers mai spectaculos, contribuind fără îndoială la creșterea imersiunii jucătorului în mediu și la creșterea plăcerii acestuia de a juca, îmbogățind atmosfera jocului.

De asemenea, nivelul, fără loading, poate deveni mai mare, limita constituind-o practic memoria computer-ului respectiv și timpul necesar încărcării (nu vrem ca acesta să devină prea mare, nu ?).

Tot cu ajutorul acestui algoritm se pot implementa mai ușor anumite efecte speciale, cum ar fi oglinzile, se poate ușura managementul de texturi și, nu în ultimul rând, se pot distruge anumite obiecte din interiorul unei celule, sau chiar pereți interiori ai unei celule.

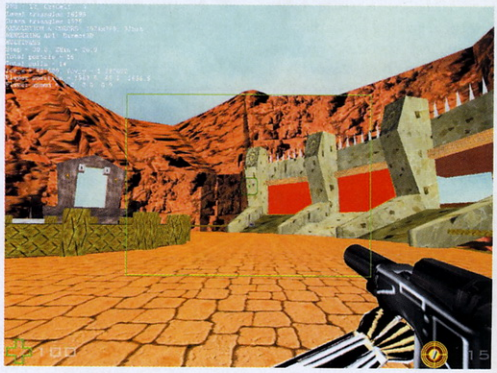
Pentru a exemplifica mai bine iată aici desenați cu roșu, folosind tehnologia FUN labs creată pentru jocul Broken Balance, dar ușor modificată în scopul prezentării, portalii unor celule:

Rolul micului dreptunghi verde din centru ecranului este de a observa ce se întâmplă cu celulele dintr-o scenă (practic debugging pentru algoritmul de portali). Astfel, trimitem la rendering toată scena dar verificăm celulele care „se văd" doar în interiorul acelui dreptunghi. Iată

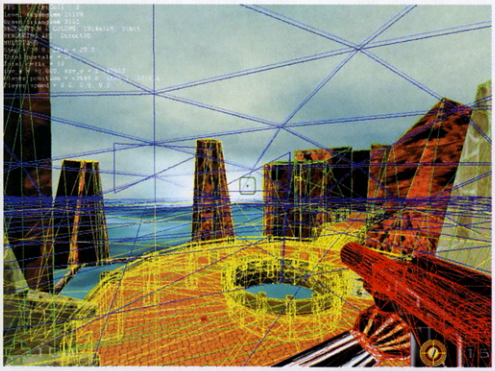


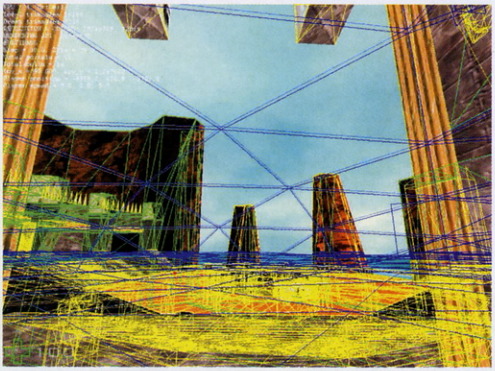
în poza următoare cum celula din stânga și două celule din dreapta pur și simplu „dispar" (fără ca cea conectată de cele din centru – planul intermediar al imaginii – să dispară):

Dacă mai rotim un pic camera spre stânga acestea vor dispărea complet și din dreapta :



Pentru a arăta și numărul mare de poligoane pe care îl permite o astfel de tehnologie iată mai jos câteva screenshot-uri făcute în modul wireframe (observați numărul de poligoane pe caracter, armă, decor etc.):





Portali

Avantaje	Dezavantaje
Permite nivele de mare întindere, anume cu multe, multe triunghiuri	Algoritm de partitionare a spațiului mai greu de implementat
Libertate mai mare de creare a nivelelor – camerele pot avea forme neregulate, spre deosebire de algoritmi pe baza de BSP care sunt limitați la anumite forme și unghiuri între suprafețe	Algoritmul bazat pe portali presupune calcule în timp real legate de determinarea vizibilității celulelor
Ușurința de implementare a părții care ține de vizualizare	Celulele sunt neapărat convexe
Overdraw relativ mic (Overdraw = efect nedorit de suprascriere a unui pixel în memoria plăcii video. Cel mai nefericit caz apare în momentul afișării din spate în față, când practic tot ce se desenează suprascrie informația deja existentă)	Este o metodă puțin folosită până acum, ceea ce implică un grad mare de inovație
Permite modificarea dinamică a structurii nivelului (pot fi distruse mai multe lucruri)	Crearea nivelelor este puțin mai complicată, deci și crearea unui editor de nivele pentru multiplayer este mai dificilă; dar reamintim că oferă și un grad mai mare de libertate, adică, pe scurt, cam orice tip de obiect și de arhitectură
Ușurința de implementare a unor efecte speciale (oglinzi)	Aparat matematic mai complex
Poate ușura management-ul de texturi	

BSP

Avantaje	Dezavantaje
Este extrem de rapid, deoarece implică puține calcule în timp real	Dimensiunea scenei este limitată de mărimea arborelui BSP (deci nivele cu puține triunghiuri)
Arborele BSP poate fi folosit și la alte prelucrări (AI, collision detection etc.)	Limitări la construirea nivelelor la unghiuri fixe sau de mărimi cuantificate.
Este un algoritm folosit pe scară largă. În consecință, este disponibilă o documentație mult mai amănunțită decât în cazul portalilor	Obiectele incluse în arbore nu se pot modifica în timpul jocului
Crearea de nivele se face mai ușor (vezi mai sus, editoare mai simple de multiplayer), dar există foarte multe reguli de respectat - iată de ce sunt așa cubice arhitecturile respectivelor jocuri;)	Overdraw mare, deci dacă nu îți construiești atent nivelul, deși ai poligoane puține, ai frame-ul mic
	Ca o consecință a algoritmului, crește numărul de poligoane ale nivelului (deci se accentuează overdraw-ul)
	Pentru a fi funcțional, are nevoie de informații suplimentare (PVS)

4. O comparație a metodelor de partiționare a spațiului – Portali vs. arbori BSP

Cam atât despre unul din algoritmii esențiali creării unui joc, partiționarea spațiului. Sperăm că ați găsit subiectul captivant și vă asigurăm că vom continua (după cum am promis) să vă oferim cât mai multe astfel de detalii, care să vă introducă în universul creării unui joc.

Claude

Jocul - Zeul Mileniului Trei

De această dată vom discuta despre o altă componentă esențială a unui engine 3D, și anume collision detection **- algoritmul de detectare a coliziunilor.**

La ce e bun un astfel de algoritm ? Mai întâi, el este cel care nu vă lasă să treceți prin pereți, de exemplu, sau prin inamici, sau prin podele. Principala utilitate a unui astfel de algoritm este deci asigurarea unei deplasări corecte a jucătorului prin nivelele jocului – iată de ce aplicarea collision detection asupra jucătorului este obligatorie.

Pentru computer, nivelele unui joc 3D nu sunt altceva decât o sumedenie de triunghiuri strânse laolaltă. Determinarea coliziunii între jucător (care și el este tot o mulțime ordonată de triunghiuri) și elementele care alcătuiesc lumea jocului (pereți, obiecte, adversari) se face deci prin verificarea existenței unor intersecții între aceste două mari mulțimi de triunghiuri. Dar această verificare nu se poate face „grosolan", prin determinarea coliziunii între fiecare triunghi din mulțimea constituită de mesh-ul jucătorului și oricare triunghi al nivelului. O astfel de abordare ar încetini foarte mult jocul, pentru că în acest caz numărul de comparații necesar ar fi uriaș.

Iată de ce se folosește reducerea jucătorului și a lumii jocului la un subset de triunghiuri numit **primitive**, elemente geometrice de bază, ceea ce reduce foarte mult numărul comparațiilor necesare.

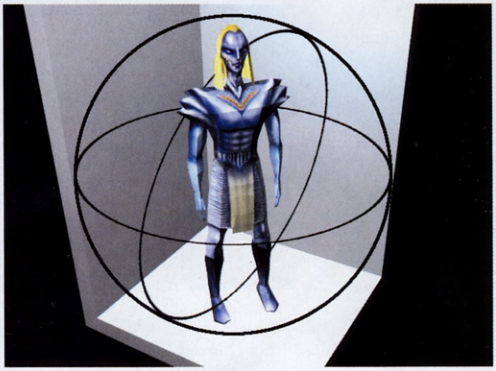
Dar algoritmul de collision detection nu se aplică numai jucătorului. El se poate aplica și celorlalte personaje din joc, pentru a evita situații în care inamicii apar din pereți sau trec prin stâlpi. Cum, în general, algoritmul de collision detection este mare consumator de resurse, soluția aplicării sale tuturor personajelor dintr-o scenă nu e rentabilă; pentru evitarea unor situații precum cele descrise mai sus se folosesc alte metode, cum ar fi definirea unor trasee obligatorii pentru deplasarea personajelor în nivel.

O altă situație în care utilizarea unui algoritm de collision detection este obligatorie este în cazul armelor. Atunci când jucătorul își folosește arma, proiectilul tras va lovi unul dintre obiectele scenei (fie el un perete, un butoi, un inamic sau cerul). Pentru a determina efectul acțiunii jucătorului, deci, trebuie determinat cu o precizie cât mai mare locul impactului, ceea ce implică folosirea unui algoritm de collision detection, și aceasta pentru fiecare proiectil tras.

Algoritmii de collision detection se implementează prin calcule de intersecție între **primitive** (primitivele sunt elementele grafice 3D „de bază": sfere, prisme, piramide etc.) sau între primitive și triunghiurile care compun scena.

1. Calculele de intersecție între primitive

Calculele de intersecție între primitive se fac în general pentru determinarea coliziunii între jucător și personajele și obiectele din scenă (excluzând pereții). În acest caz, atât jucătorul cât și obiectele se încadrează în **primitive de aproximare**, numite astfel deoarece aproximează forma obiectelor sau a jucătorului la un moment dat.



Dezavantajul evident al acestei metode este că sfera nu aproximează foarte bine forma obiectelor, putând apărea situații de erori grosolane chiar pentru forme relativ simple.

Aceste primitive de aproximare pot fi:

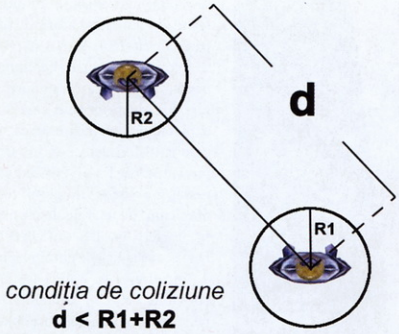
- **sfere;**
- paralelipipede aliniate la axele de coordonate (ale căror fețe sunt paralele cu planele determinate de axele de coordonate), numite **AABB** (Axis Aligned Bounding Box);
- paralelipipede aliniate la obiect (de volum minim, care aproximează cel mai bine volumul obiectului respectiv), numite **OBB** (Oriented Bounding Box).

Sferele au marele avantaj de a avea teste de intersecție foarte simple și rapide.

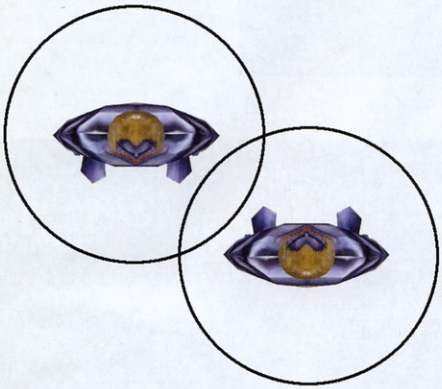
Astfel, pentru a testa intersecția dintre două sfere, se calculează distanța dintre centrele sferelor, care se compară cu suma razelor.

Paralelipipedele AABB au două avantaje față de sfere: mai întâi, ele aproximează în majoritatea cazurilor forma obiectelor mai bine decât sferele, iar în al doilea rând testul de intersecție între două AABB este foarte rapid (se rezumă la testarea intersecțiilor dintre proiecțiile fețelor pe planele axelor; cum AABB sunt aliniate la axe, aceasta testare poate fi optimizată foarte ușor).

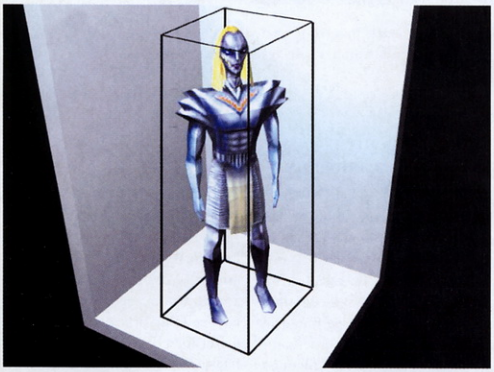
Dezavantajul acestei metode apare în momentul în care obiectul astfel aproximat se rotește; atunci, AABB-ul nu va mai corespunde noii poziții a obiectului (nu-l mai încadrează) ceea ce obligă la calculul dinamic al AABB.



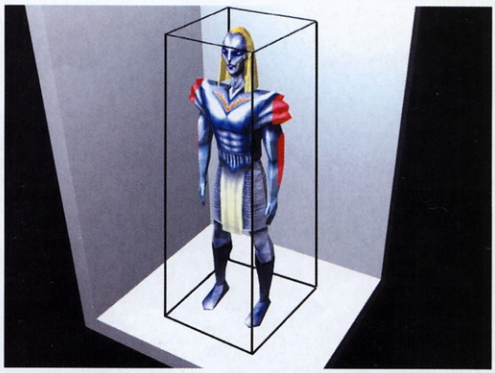
Dacă distanța dintre centrele sferelor este mai mare decât suma razelor acestora, cele două sfere nu se intersectează, deci se consideră că personajele înscrise în sfere nu se ciocnesc.



Iată un caz în care deşi sferele se intersectează, personajele încadrate de sfere nu se ating.



Observaţi caracteristica de bază a AABB – alinierea la axe.

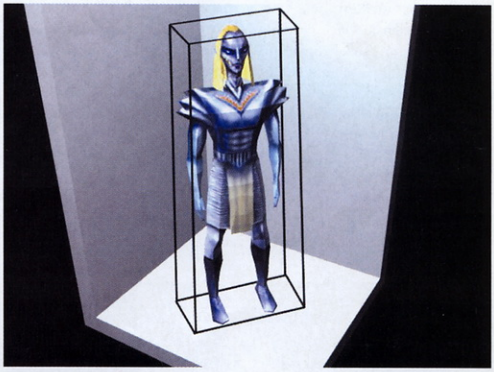


După rotire, personajul nu mai este încadrat de AABB (umerii şi o parte din braţe, marcate cu roşu, ies din paralelipipedul iniţial).

O soluţie pentru evitarea calculului dinamic al AABB este aproximarea volumului prin AABB-ul corespunzător cazului cel mai nefavorabil (de volum maxim), dar acest lucru, evident, nu mai oferă încadrarea optimă şi măreşte probabilitatea apariţiei unor erori la determinarea coliziunilor.

Paralelipipedele OBB aproximează cel mai bine forma unui obiect (chiar din definiţia lor apare această proprietate, ele fiind paralelipipedele de volum minim care încadrează obiectul respectiv), dar calculele de intersecţie sunt mai complicate şi implicit mai lente.

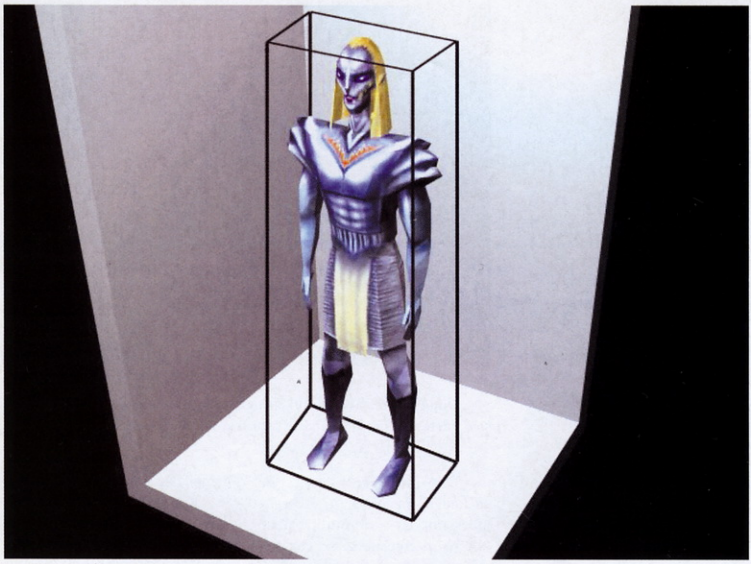
Un alt dezavantaj al OBB apare la rotirea obiectului; pentru a păstra încadrarea obiectului în paralelipiped, e necesară rotirea simultană a OBB şi a obiectului, ceea ce presupune calcule suplimentare.



Spre deosebire de AABB, OBB nu este aliniat la axele de coordonate. OBB este paralelipipedul de volum minim care încadrează în întregime personajul.

2. Calculele de intersecţie între primitive şi triunghiurile care compun scena

Elementele fixe ale nivelului (pereţii, scările, tavanul, podeaua) nu pot fi aproximate prin primitive. Pentru determinarea coliziu-



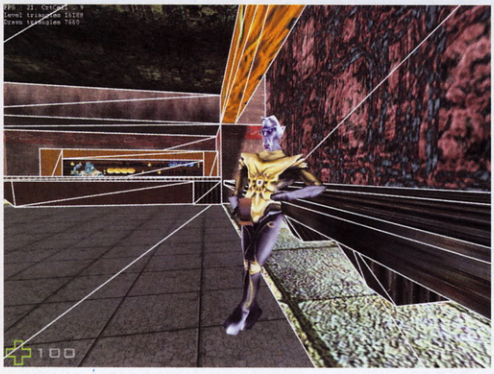
La rotirea personajului, rotirea concomitentă a OBB duce la păstrarea încadrării.

nilor între jucător şi acestea se folosesc teste de intersecţie între primitiva care încadrează jucătorul şi:

a. triunghiurile scenei. Acest test este ineficient în zonele cu multe triunghiuri, din cauza numărului mare de teste, şi nu oferă un collision de bună calitate.

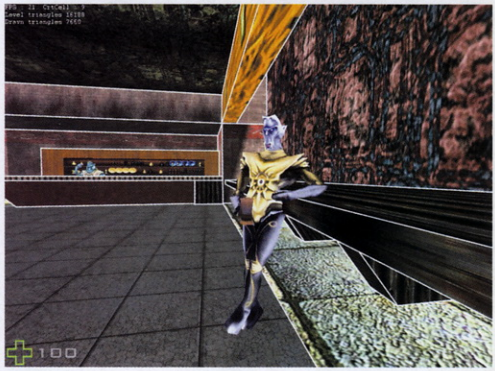
b. poligoanele scenei. Acest test se bazează pe gruparea triunghiurilor care formează scena, în momentul construcţiei nivelului, într-un set de poligoane convexe. Testele de coliziune se vor efectua între primitiva care aproximează jucătorul şi acest set redus de poligoane. Această metodă depinde indirect de numărul de triunghiuri ale scenei.

c. plane ce aproximează geometria. Acest test se bazează pe decuparea nivelului în plane de coliziune, care aproximează geometria nivelului. Testele de coliziune se fac între primitivă şi planele cele mai apropiate. Această metodă este independentă de numărul de triunghiuri ale nivelului. Ea s-a folosit în general la engine-uri bazate pe arbori BSP, cu rezultate foarte bune.



Observaţi numărul mare de triunghiuri din care este format acest interior simplu.

Iată, mai exact, cum are loc testul de coliziune : primitiva de bază (fie ea sferă, AABB sau OBB) se foloseşte doar ca test preliminar, pentru determinarea ramurii din ierarhia de primitive care va fi explorată în continuare. În cazul în care collision detection-ul cu primitiva de bază s-a verificat, se va testa mai departe pe nivelele interioare (pe ierarhia de primitive, fi ele sfere, AABB sau OBB-uri). După ce se ajunge la primitiva finală de încadrare, se aplică testul de coliziune cu acele triunghiuri ale personajului care sunt încadrate de această primitivă, pentru a putea aplica corect texturile de „simulare" (impact, deteriorarea costumului, sânge etc.) şi pentru a putea calcula efectul impactului (de exemplu, în funcţie de locul în care a avut loc impactul, personajul poate muri, îşi poate pierde un membru – aţi ghicit, în acest caz se elimină din ierarhia de primitive acea ramură corespunzătoare mâinii sau piciorului „dispărut").



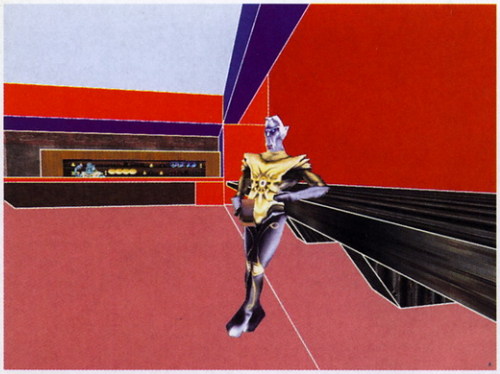
Aceeaşi scenă, în care o parte din triunghiuri au fost grupate în poligoane. Acum numărul de teste necesare este mai mic.

3. Calculele de intersecţie pentru determinarea impactului proiectilelor

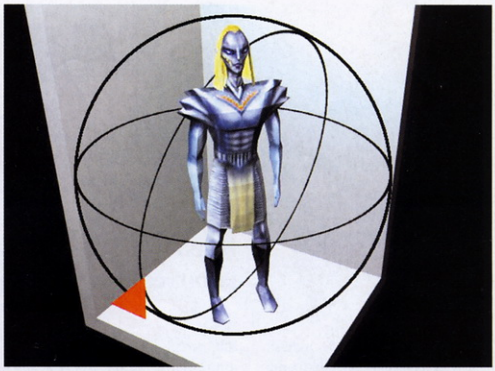
Pentru a determina locul exact al ciocnirii dintre un proiectil şi un personaj nu se poate folosi doar o singură primitivă de încadrare a personajului. Se foloseşte o ierarhie de primitive, grupate pe ramurile unui arbore binar, care va fi descompus în cazul testului de coliziune până la detectarea celei mai mici primitive care verifică testul.

Un aspect foarte spectaculos al calculelor de intersecţie îl reprezintă aplicarea texturilor de simulare (a decals-urilor), care sunt răspunzătoare pentru realismul grotesc al jocurilor. Petele de sânge, găurile din armuri şi celelalte urme lăsate de armele voastre pe corpurile inamicilor se aplică prin texturi suprapuse texturilor de bază; locul unde urmează să fie aplicate decals-urile se stabileşte prin calcule de intersecţie.

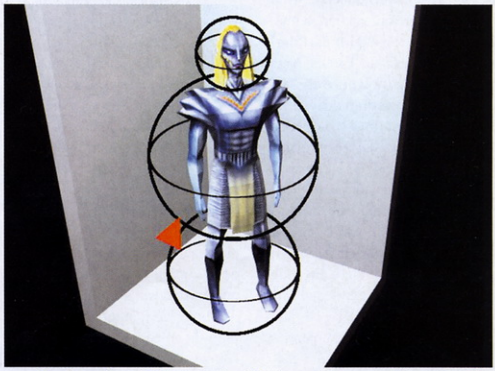
Unele jocuri „înşeală" jucătorii



Suprafeţele de aceeaşi culoare aparţin aceluiaşi plan. Observaţi reducerea semnificativă a numărului de teste necesare.



Proiectilul atinge prima sferă de încadrare.



Conform ierarhiei, se testează intersecţia cu următoarele sfere din ierarhie.

prin aplicarea unor decals în apropierea zonei de impact (uneori, factorul de aproximare este foarte mare...) sau prin aplicarea la întâmplare a texturilor. Pentru un realism complet al jocului, însă, aplicarea decals-urilor prin calcule de intersecţie este obligatorie.

Un alt aspect de care este răspunzător acest test de impact al proiectilelor este precizia ţintirii. Astfel, fără determinarea cât mai exactă a locului de impact, nu ar fi posibil să ratezi ţinta trăgându-i printre picioare, de exemplu.

Cam atât deocamdată. După ce aţi parcurs această prezentare a principalilor algoritmi de collision detection, probabil că vă întrebaţi care dintre aceste metode este cea mai bună.

Ei bine, de obicei, se recurge la un „amestec" de metode, fiecare dintre ele fiind folosită în condiţiile pentru care dă randament maxim.

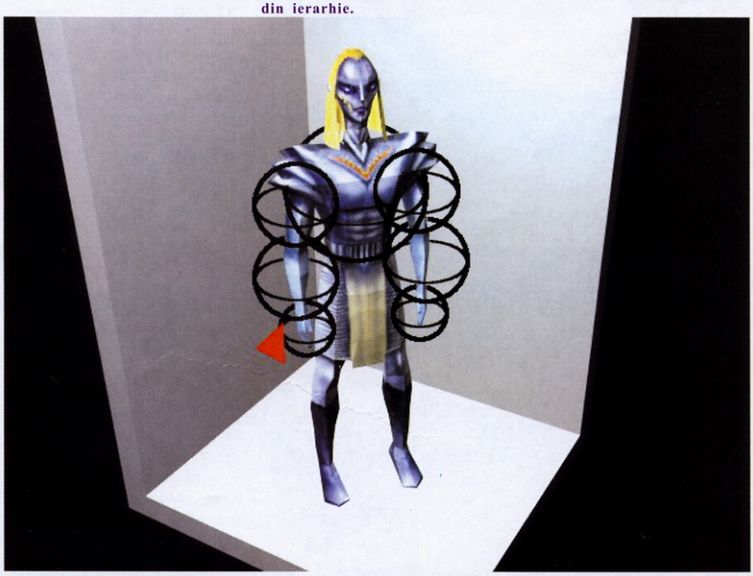
Astfel, pentru deplasarea jucătorului în nivel, se poate folosi decuparea nivelului în plane care aproximează geometria. Acest lucru permite efectuarea unui număr relativ mic de teste, ceea ce duce la o mişcare mai fluidă a jucătorului în nivel şi reduce „ocuparea" procesorului.

Pentru testarea contactului între primitive, ca şi pentru determinarea impactului proiectilelor, se va folosi o ierarhie complexă alcătuită din sfere (la nivel primar, pentru testele preliminare) şi din OBB (pentru o aproximare cât mai bună a locului impactului şi segmentarea mesh-ului personajelor în trunchi, cap şi membre).

Acestea fiind spuse, sperăm că explicaţiile oferite nu au fost prea „seci" şi că aţi înţeles ceea ce vrem, de fapt, să vă spunem prin această serie de articole : **să faci un joc nu e deloc o joacă...**

Interesul pentru această serie de articole a depăşit cu mult aşteptările mele, fapt ce nu poate decât să mă bucure. Vom reveni ca de obicei în numărul viitor al revistei cu noi detalii interesante despre - crearea unui joc de calculator.

Claude



Şi aşa mai departe, până la depistarea celei mai mici primitive cu care testul de coliziune se verifică.

Jocul - Zeul Mileniului Trei

După ce în articolele anterioare am fost destul de tehnici, povestindu-vă câte ceva despre componentele esențiale ale unui motor grafic 3D, acum vă propunem o dezbatere asupra unui alt punct important în construcția unui joc: scalabilitatea.

Cu toții am vrea să putem juca cele mai noi și mai reușite jocuri. Pentru cei mai mulți, însă, cerințele de sistem necesare rulării ultimelor jocuri apărute depășesc cu mult capacitățile computerelor la care avem acces. Cu siguranță, și voi ați trecut prin situația în care ați fost nevoiți să înghițiți în sec, văzând screenshot-urile unui joc nou-nouț și gândindu-vă că nu-l veți putea juca pe amărâtul vostru de calculator. Această problemă nu poate fi rezolvată decât în două moduri: un upgrade *(adică un efort din partea jucătorilor pentru a-și aduce hardware-ul la nivelul cerut de joc)* sau un engine scalabil *(adică un efort din partea producătorilor pentru a face ca jocul lor să poată fi rulat pe configurații cât mai diferite)*. Scalabilitatea reprezintă, deci, proprietatea unui joc de a putea fi „adaptat" pentru a fi rulat fără probleme pe configurații de putere sensibil diferită. Astfel, folosind un engine scalabil, același joc ar putea fi jucat și pe un K7 cu 128 M RAM și Nvidia TNT2, cât și pe un Pentium 100 cu 16 MB RAM.

Motivul pentru care decizia de a construi un engine scalabil este foarte importantă în construcția unui joc este clar: permite o creștere sensibilă a audienței jocului. Este greu, însă, să faci în așa fel încât jocul tău să poată fi jucat și pe sistemele „low-end" *(acest termen se referă la configurațiile cele mai ieftine pe care le puteți găsi în magazine, și chiar și la cele care sunt atât de depășite încât nu le mai puteți găsi în magazine... :))*, dar să și ofere grafica și efectele speciale pe care se așteaptă să le întâlnească fericiții posesori ai sistemelor de top. Producătorii, însă, nu se prea înghesuie să încerce așa ceva. Motivul imediat este faptul că dezvoltarea unui engine scalabil mărește timpul de dezvoltare a jocului, deoarece impune implementarea unor metode și tehnici suplimentare. Echipa **Fun Labs**, însă, crede că, datorită exploziei hardware și a „prăpăstiei" din ce în ce mai accentuate între sistemele de top și cele obișnuite, un engine scalabil este în acest moment absolut necesar. În același timp companii, precum **UbiSoft**, nu au folosit și nici nu intenționează să folosească această metodă.

Să vedem care ar fi motivele pentru care scalabilitatea a căpătat o importanță sporită în ultimul timp. În primul rând, este vorba de creșterea explozivă a puterii de calcul și a frecvenței de lucru a microprocesoarelor. Dacă în 1998 un procesor „de top" avea 400 MHz, astăzi, la începutul anului 2000, întâlnim procesoare cu frecvențe de 800 MHz. În al doilea rând, au apărut seturile de instrucțiuni specializate pentru creșterea performanței în prelucrările grafice 3D (vă spune ceva acronimul MMX?). În cele din urmă, revoluția acceleratoarelor grafice 3D este poate cel mai important factor care a dus la acutizarea problemei scalabilității – diferența de performanță dintre un sistem care posedă o placă accelatoare și unul lipsit de aceasta este enormă.

Vă întrebați cu siguranță cum, în aceste condiții, putem face ca un joc să poată fi jucat la fel și pe un computer mai vechi și pe unul nou-nouț. Dacă nu ați observat până acum, trebuie să vă spun că am vorbit numai despre modul în care jocul va fi jucat, nu despre modul în care va arăta. Cu alte cuvinte, e imposibil să încerci să recreezi puzderia de efecte speciale din *Unreal* pe un 486, la fel cum luminile, umbrele și exploziile din *Incoming* nu pot fi recreate pe un computer care nu are o placă grafică acceleratoare. Din punct de vedere exclusiv al graficii, deci, scalabilitatea nu oferă o soluție universală. Ea permite însă ca, renunțând la unele „artificii" destinate exclusiv sistemelor performante, jucătorii mai puțin „înzestrați" să se poată bucura și ei de joc.

1. Dificultățile de abordare

Există două impedimente mari în rezolvarea problemei scalabilității.

Mai întâi, trebuie să stabilim cum anume putem scala elementele jocului (de exemplu, cum putem reduce numărul de poligoane folosit pentru reprezentarea unui personaj) – altfel spus, ce metodă sau ce algoritm folosim pentru scalarea conținutului jocului.

A doua problemă este determinarea „amplitudinii" scalării. În funcție de capacitățile sistemului pe care va rula jocul, trebuie să determinăm cât de multe din elementele jocului care vor fi scalate sau chiar eliminate și în ce măsură.

Vom începe prin a aborda primul aspect – determinarea metodelor de scalare a elementelor jocului.

2. Un exemplu de scalare a elementelor jocului

Să presupunem că jocul nostru este un 1st person shooter. Jucătorul se află într-un oraș cu clădiri înalte și străzi intens circulate. Deodată, în momentul în care jucătorul intră pe o alee înghesuită între două clădiri, doi șmecherași îl atacă.

Din punct de vedere al jocului, viteza cu care atacă cei doi adversari, strategia lor de luptă și reacțiile lor la acțiunile jucătorului trebuie să fie aceleași și pe un sistem slab, și pe unul performant. Ceea ce poate să difere, însă, este bogăția detaliilor. Astfel, un sistem performant poate face în același timp calculele necesare randării scenei, stabilirii traseului adversarilor, calculului coliziunilor, ca și introducerii unei mulțimi de elemente suplimentare cum ar fi trecătorii și vehiculele care se văd în capătul aleii, păsările care zboară în văzduh, o fereastră deschisă care se mișcă datorită vântului, gunoaiele de pe jos... Toate aceste elemente pot avea umbre, se poate auzi zgomotul circulației sau al ferestrei lovite de bataci, armele folosite pot avea efecte vizuale spectaculoase șamd. Aceste elemente suplimentare au un impact foarte mare asupra atmosferei jocului, a imersiunii jucătorului în lumea fictivă care i se propune. Ele nu sunt esențiale, însă; introducerea sau absența lor nu schimbă cu nimic faptul că jucătorul trebuie să se lupte, folosindu-și armele, împotriva a doi adversari. Aceasta este esența jocului.

De aceea, este evident că pe un sistem slab elementele suplimentare vor dispărea, păstrându-se doar acele elemente considerate esențiale pentru joc – inamicii, armele, efectele sonore și vizuale legate de folosirea armelor și de lupta propriu-zisă. Se poate stabili o ierarhie a

importanței elementelor jocului, în funcție de care aceste elemente pot apărea sau nu în joc, conform puterii de calcul a sistemului pe care se joacă.

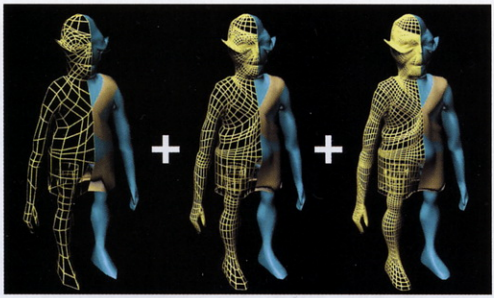
Până acum am vorbit despre modul în care se poate face scalarea elementelor jocului. Aplicarea practică a scalării ridică însă probleme legate de scalarea geometriei jocului, de metodele sau algoritmii folosiți pentru reducerea numărului de calcule necesare pentru reprezentarea lumii jocului.

3. Scalarea geometriei jocului

Pe scurt, scalarea geometriei reprezintă modul în care lumea jocului va putea fi reprezentată folosind mai multe sau mai puține detalii. Pentru un engine 3D, cantitatea de detalii oferită poate fi echivalată cu numărul de poligoane folosite pentru reprezentarea elementelor jocului – a clădirilor, a personajelor, a armelor etc. Scalarea geometriei jocului presupune, în consecință, creșterea sau micșorarea numărului de poligoane randate pentru a păstra constant frame rate-ul. Este clar, deci, că sistemele mai performante pot afișa imagini construite din mai multe poligoane, anume imagini mai clare și mai realiste.

Păstrarea constantă a numărului de cadre pe secundă *(frame rate)* se poate realiza numai prin menținerea sub o limită stabilită a numărului de poligoane pe care computerul trebuie să le prelucreze și să le afișeze la un moment dat. Dacă scalarea presupune modificarea numărului de poligoane folosite pentru a reprezenta lumea jocului, astfel încât și computerele mai slabe să poată păstra un fps (număr de cadre/secundă) constant, o tehnică asemănătoare este folosită pentru optimizarea calculelor. Această metodă poartă numele de LOD (Level Of Detail – nivel de detaliu). Pe scurt, ea permite modificarea în timp real a numărului de poligoane folosit pentru reprezentarea unor modele 3D, în funcție de distanța dintre aceste modele și jucător. Dar această tehnică, deși înrudită cu tehnicile folosite pentru implementarea scalabilității, nu este subiectul principal al acestui articol. Să ne întoarcem la oile noastre (3D, evident)...

În cele ce urmează vom prezenta cele patru metode principale de scalare folosite în jocurile 3D.



3.1 Metoda modelelor multiple (Multiple LOD Models)

O metodă des întâlnită de scalare a geometriei presupune existența mai multor seturi de modele 3D, de calități diferite (construite din mai multe sau mai puține poligoane, deci la un nivel de detaliu - LOD - diferit), care să aproximeze lumea jocului. În funcție de puterea sistemului, va fi folosit unul sau altul din aceste modele.

Această metodă are mai multe dezavantaje.

În primul rând, cantitatea de muncă a modelatorilor 3D va crește considerabil; în loc să creeze doar un singur model, ei vor trebui să genereze mai multe modele pentru fiecare element al jocului.

În al doilea rând, existența mai multor seturi de modele implică spațiu suplimentar ocupat pentru stocarea versiunilor „reduse" ale lumii jocului.

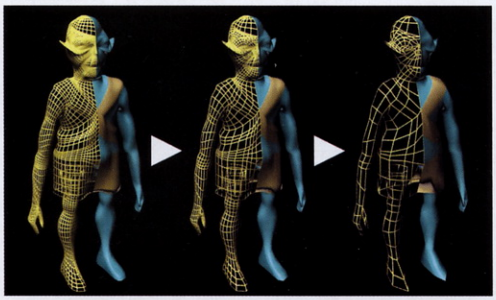
În sfârșit, trecerea de la un model mai complex la unul mai puțin bogat în poligoane poate produce o impresie vizuală neplăcută. Închipuiți-vă ce se întâmplă atunci când jucătorul se îndepărtează destul de mult de un element al jocului (să zicem, un alt personaj). Din moment ce de la o anumită distanță detaliile care „înghit" poligoane nu se mai pot desluși, ce sens are să obligăm computerul să le calculeze în continuare? O metodă uzuală este înlocuirea modelului complex cu unul mai simplu, caz în care poate apărea problema descrisă mai sus.

3.2. Metoda modelelor progresive (Progressive Meshes)

Se pleacă de la ideea înlăturării unora din poligoanele care formează un model 3D complex, pentru a realiza astfel un model mai simplu, din mai puține poligoane. Se pleacă deci de la un model 3D complex, creat pentru a fi reprezentat integral pe mașinile cele mai performante ale momentului, care va fi modificat conform unui algoritm pentru a se obține modele apropiate ca formă de modelul inițial, dar reprezentate printr-un număr redus de poligoane. Metoda modelelor progresive permite deci reprezentarea unui model 3D cu un număr arbitrar de poligoane, numărul maxim de poligoane prin care poate fi reprezentat fiind determinat de numărul de poligoane al modelului complex inițial.

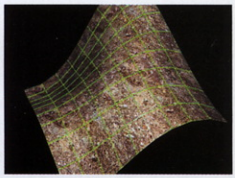
Un avantaj clar al acestei metode ține de micșorarea cantității de muncă a modelatorilor, ca și de reducerea spațiului ocupat pentru stocarea modelelor. Se vor stoca numai modelul 3D inițial și informațiile legate de algoritmul de „divizare" a modelului (care spun care este ordinea de eliminare a poligoanelor).

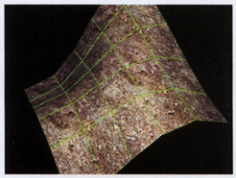
Dezavantajul acestei metode este ca algoritmul folosit trebuie să fie foarte performant pentru a păstra o asemănare cât mai mare cu modelul inițial; cu cât algoritmul de diviziune este mai bun, cu atât puterea de calcul necesară este mai mare.

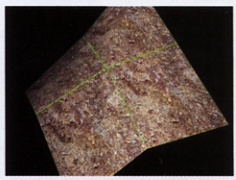


3.3. Metoda suprafețelor parametrice (Parametric Surfaces)

O altă metodă, cu rezultate mai bune, este folosirea suprafețelor parametrice. Ce înseamnă asta? Un element al jocului poate fi descris folosind formule matematice care







generează suprafețele care îl constituie (de exemplu, brațul poate fi definit de o funcție 3D, fiecare din elementele feței – ochi, buze, obraji etc. - de altele etc.). Aceste formule matematice pot fi apoi folosite pentru a genera în timp real un număr mai mare sau mai mic de triunghiuri care vor fi randate de motorul grafic al jocului.

Este clar că folosirea suprafețelor parametrice permite o scalare foarte ușoară a jocului; în funcție de performanțele sistemului, numărul de triunghiuri generat în timp real va fi modificat fără nici o problemă.

Dezavantajul acestei metode este, în principal, legat de implementare. Mai exact, de punctele de intersecție ale funcțiilor. Să zicem că o funcție va aproxima foarte exact forma unui deget, în timp ce o altă funcție poate aproxima podul palmei. În punctul în care se „leagă" degetul de podul palmei, trebuie realizată o interpolare a celor două funcții, pentru ca trecerea de la deget la palmă să fie suficient de lină. Problema este că în cazul existenței unui număr mare de funcții (pentru modelele 3D complexe) calculul interpolărilor poate depăși ca resurse necesare randarea propriu-zisă a modelului și calculele de scalare, luate împreună.

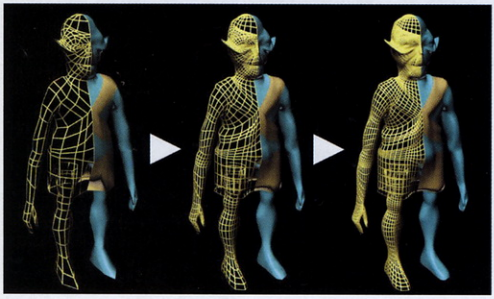
4. Alte posibilități de aplicare a scalabilității

Nu numai numărul de poligoane poate fi modificat pentru a ajuta la obținerea performanțelor dorite (viteza, fps) pe un calculator mai slab. Gândiți-vă și la efectele speciale sau la modul în care jocul implementează lumina și umbrele. În sfârșit, dacă putem scala modelele (mesh-urile) clădirilor și personajelor, de ce să nu facem același lucru pentru animații?

Să parcurgem pe rând aceste elemente.

4.1. Scalarea luminii

Lumina este unul dintre factorii cei mai importanți pentru crearea



3.4. Metoda suprafețelor de subdiviziune (Subdivision Surfaces)

Ideea de bază a acestei metode este cumva inversa celei folosite în metoda modelelor progresive. Dacă acolo se pleca de la un model complex pentru a-l reduce în modele mai simple, aici se pleacă de la un model simplu care va fi modificat pentru a-l face din ce în ce mai complex.

Dacă avem, de exemplu, un model 3D, compus dintr-un număr redus de poligoane, algoritmul de bază al acestei metode presupune ca fiecare suprafață a modelului (fiecare poligon, în cele din urmă) să fie împărțit în mai multe poligoane, care să aproximeze din ce în ce mai bine forma complexă a modelului. Acest procedeu va duce în cele din urmă la crearea unor suprafețe din ce în ce mai netede și care să aproximeze mai bine suprafețele curbe ale modelului.

Dezavantajul acestei metode este legat de complexitatea relativă a calculelor de interpolare și a metodelor de calcul al punctelor optime de „spargere" a suprafețelor.

unei lumi realiste. Puteți să vă convingeți de asta dacă folosiți editorul de nivele al *Unreal* pentru a vedea cum arată nivelele cu sau fără luminile de atmosferă. Practic, lipsită de lumini, lumea din *Unreal* este la fel de plată ca și cea din *Wolfenstein*.... :) Bine, recunosc, am exagerat, texturile din *Unreal* sunt totuși mai bune... :)

De aceea, trebuie să existe o metodă prin care lumina și umbrele să poată fi scalate pentru a apărea într-un mod asemănător atât pe sistemele slabe cat și pe cele performante.

În primul rând, deoarece o serie de metode care sunt accesibile pentru sistemele performante (iluminarea hardware, de exemplu) pot să fie imposibil de aplicat pe sistemele low-end, este clar că trebuie să existe metode diferite de iluminare pentru cele două situații. Iată de ce am folosit termenul „asemănător" în fraza anterioară. Ideal ar fi ca metodele alese să fie diferite ca implementare, dar rezultatul lor să fie cât mai apropiat; acest lucru trebuie să se facă fără ca elementele esențiale ale gameplay-ului să aibă de suferit (e poate inutil să subliniez din nou acest lucru, dar trebuie să vă amintesc că, de fapt, principalul scop al scalabilității este să păstreze intact gameplay-ul și să ajute la reprezentarea unei lumi a jocului cât mai coerentă atât pe sistemele „de top" cât și pe cele mai slabe).

Să presupunem că, pentru un sistem dotat cu accelerare 3D, luminile jocului vor fi dinamice și se vor calcula umbrele în timp real ale tuturor obiectelor și personajelor. Pentru un sistem mai slab, luminile nu vor mai fi dinamice, iar umbrele vor fi implementate prin lightmaps. Evident că diferența grafică dintre cele două sisteme va fi mare, dar jocul va putea fi jucat pe ambele computere.

Iată o scurtă trecere în revistă a câtorva metode care pot fi folosite pentru implementarea luminilor, în funcție de puterea de calcul a sistemului de care dispunem: iluminarea hardware, care folosește capacitățile plăcilor grafice acceleratoare 3D; simularea luminii prin aplicarea unor texturi a căror poziție și intensitate sunt calculate în timp real; simularea luminii prin texturi precalculate și texturi ale căror coordonate se schimbă în timp real.

Pentru reprezentarea umbrelor există de asemenea o serie de metode, mai mult sau mai puțin realiste. Cea mai simplă este folosirea unor texturi de dimensiune și formă fixă, care vor fi afișate întotdeauna sub obiectul care ar trebui sa aibă umbră. Alte metode, ceva mai realiste, ar fi folosirea unor texturi dinamice (seturi de texturi care să aproximeze umbra unui obiect în mișcare), folosirea proiecției obiectului (care este „aplatizat" pe sol, poligoanele astfel obținute fiind texturate cu o textură întunecată) sau folosirea posibilităților plăcilor acceleratoare 3D (este versiunea cea mai realista, prin care se generează volume de umbra – corpuri calculate în funcție de poziția obiectului umbrit și a sursei de lumină – a căror intersecție cu celelalte obiecte din lumea jocului determină poziția și intensitatea umbrei).

4.2. Scalarea efectelor speciale

Efectele speciale sunt unele din cele mai atrăgătoare elemente ale jocurilor. Există jocuri care au câștigat enorm datorită efectelor speciale (*Incoming*, de exemplu), acest capitol ajutând jucătorii să treacă cu vederea unele lipsuri ale jocului.

Cea mai simplă metodă de scalare a efectelor speciale este pur și simplu desființarea acestora. Dacă jucătorul are o mașină slabă, ei bine, poate să tragă cu bazooka în clădiri cât vrea – tot n-o să vadă nici o explozie.

Evident, această metodă e puțin prea „radicală". Și pentru efecte, ca și pentru elementele jocului, se poate construi o ierarhie a importanței lor. Astfel, pe un sistem puternic, impactul rachetei lansate de jucător va declanșa o explozie care va arunca schije de jur împrejur; după impact, solul sau peretele lovit va păstra urma exploziei și va fumega. În funcție de puterea de calcul a sistemului, se pot elimina unele din efecte (fumul, schijele) sau calitatea lor grafică (se vor folosi texturi mai mici sau un număr mai mic de poligoane pentru a simula exploziile).

4.3. Scalarea animațiilor

Dacă pentru personaje am descris deja câteva metode de scalare a

modelelor 3D, animațiile ridică o serie de probleme suplimentare, legate de modul de implementare a animației în joc.

Cea mai simplă metodă de animație 3D este stocarea unei serii de modele care să reprezinte personajul în timp ce efectuează o anumită acțiune (de exemplu, când merge).

Este clar că această metodă are o mulțime de dezavantaje.

În primul rând, cantitatea de muncă necesară. Fiecare acțiune a jucătorului trebuie prevăzută și pentru fiecare trebuie creată animația respectivă. În plus, existența a două animații distincte (de exemplu, o animație pentru mers și alta pentru săritură) nu permite construcția automată a acțiunii compuse (trebuie să creăm o animație suplimentară pentru săritură din alergare).

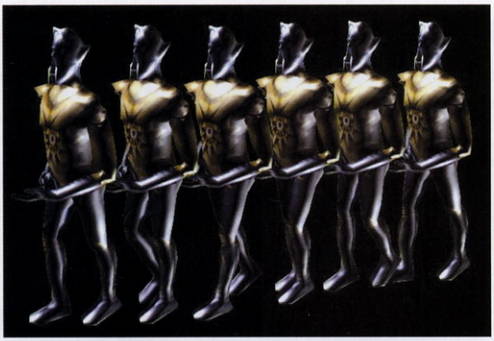
În al doilea rând, animațiile astfel stocate sunt gândite pentru a fi afișate la un anumit număr de cadre pe secundă. Să zicem că un personaj are o animație de mișcare care se întinde pe zece cadre (un pas durează zece cadre). Dacă jocul va fi rulat pe o mașină puternică, în stare să calculeze 70 de cadre pe secundă, animația personajului îl va aminti pe Charlot cel din filmele mute; în schimb, pe o mașină slabă, capabilă de numai 15 cadre pe secundă, personajul se va deplasa cu încetinitorul.

O altă metodă de animație este cea bazată pe „oase". Oasele sunt reprezentări simplificate ale elementelor de bază ale unui model; animațiile modelului sunt stocate ca animații ale oaselor, care influențează pe baza unui algoritm deformările modelului.

Scalarea unei animații se bazează pe o tehnica asemănătoare, indiferent de tipul de animație folosit. Această tehnică poarta numele de interpolare a cadrelor.

În funcție de numărul de cadre/secundă care poate fi obținut, animația se va modifica real time, astfel încât durata totala a acesteia să fie aceeași. De exemplu, dacă animația a fost creată pentru 24 de cadre/secundă și computerul respectiv poate randa 70 de cadre/secundă, se vor insera prin interpolare noi poziții în cadrul animației. Dacă sistemul respectiv nu poate randa decât 16 cadre/secundă, se vor elimina unele din cadrele animației.

Putem vedea acum că un avantaj important al metodei bazate pe oase este că permite o interpolare mai ușoară a diferitelor cadre ale



animației (calculele se efectuează asupra unui număr mult redus de puncte).

5. Determinarea amplitudinii scalării

Este foarte important să alegem corect metoda prin care se va determina amplitudinea scalarii, adică numărul de elemente ale jocului care vor fi scalate sau la care se va renunța pentru a permite rularea jocului la parametrii maximi pe care îi permit caracteristicile computerului respectiv. Există patru posibilități:

5.1. Configurarea scalării - de către utilizator

În acest caz, jucătorul este cel care poate modifica variabilele de scalare, încercând diverse configurații, în funcție de preferințele sale. Este cea mai ușor de implementat. Pe de altă parte, deși îi permite jucătorului să intervină în acest proces și să aibă controlul total asupra aspectului jocului, mulțimea de parametri asupra căruia va trebui să intervină poate fi uneori prea mare pentru un jucător începător și poate ajunge să plictisească sau să zăpăcească și pe jucătorii cu experiență.

5.2. Configurarea scalării - în momentul instalării jocului

În această situație, imediat după instalarea jocului, programul de instalare va efectua o serie de teste asupra configurației computerului, comparând rezultatele cu o grilă de configurații sau cu o serie de limite individuale ale componentelor. În funcție de rezultate se va stabili amplitudinea scalării și numărul de elemente ale jocului la care se va renunța. Principalul dezavantaj al acestei metode este că după instalare pot exista modificări ale configurației care să invalideze rezultatele obținute.

5.3. Configurarea scalării - înainte de începerea jocului propriu-zis

De această dată, testele de performanță vor fi realizate înainte de lansarea jocului propriu-zis. Sărind peste problema nivelului de relevanță al testelor, adevărata problemă ridicată de acest tip de configurare este mărirea importantă și nejustificată a timpului de așteptare până la începerea jocului.

5.4. Configurarea scalării - în timpul jocului

Această metodă este poate cea mai bună, deoarece permite introducerea în joc a unor elemente suplimentare în funcție de caracteristicile speciale ale nivelelor jocului. De exemplu, dacă implementarea gunoaielor zburate de vânt și a ferestrei care se zbate din exemplul nostru inițial nu se poate face într-o zonă intens populată, în schimb, atunci când jucătorul străbate o zonă pustie asemenea elemente pot fi introduse. Cu alte cuvinte, scalarea real time a jocului presupune reducerea nivelului și numărului detaliilor în zonele CPU intensive (cum ar fi zonele de luptă, unde jucătorul oricum n-are timp să studieze cerul sau să admire modul în care se rostogolește o frunză pe jos) și creșterea lor în zonele mai lipsite de acțiune.

Cea mai bună soluție o constituie un hibrid: trebuie lăsată jucătorului posibilitatea să-și configureze jocul, dar se pot adăuga în mod automat elemente care să îmbogățească atmosfera jocului acolo unde acest lucru este posibil.

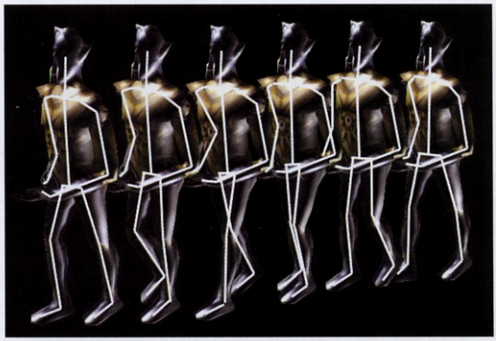
6. Punctul pe i

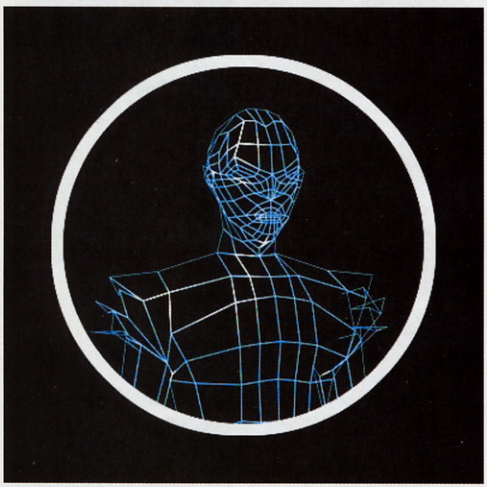
Una peste cealaltă, scalabilitatea ridică unele întrebări la care numai voi, jucătorii, puteți răspunde. Este oare mai important să te poți juca un joc, chiar dacă nu te poți bucura de toate efectele speciale și de toate elementele de atmosfera suplimentare pe care acesta le include?

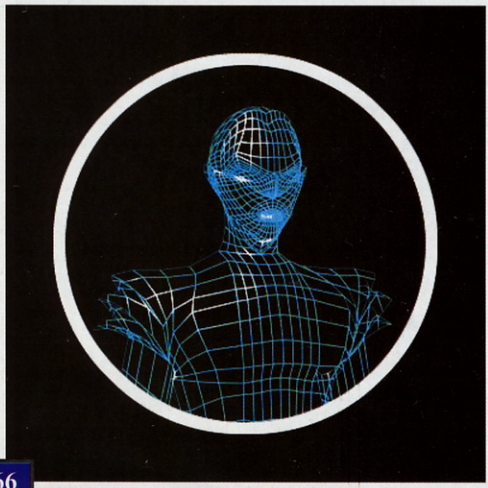
Cu alte cuvinte, ați prefera să puteți juca un joc și pe computerul vostru, cu putere de calcul medie, chiar dacă nu veți vedea unele dintre exploziile superbe sau jocurile de lumina pe care le pot admira cei care au sisteme mai puternice? Sau ați strânge din dinți și v-ați apuca să economisiți pentru a va putea rula jocul la parametrii maximi pe care acesta îi oferă? Ce este, deci, mai important: jocul în sine, gameplay-ul, sau efectele speciale și grafica trăznet? (Eterna întrebare a unui producător.)

După cum se vede, a face un joc nu este atât de simplu și chiar dacă se pornește în urma unei pasiuni sau impuls de moment, cu timpul ne vom lovi capul de pragul de sus.

Claude







Jocul - Zeul Mileniului 3

LOD - Level Of Detail

Cu siguranţă aţi auzit de termenul FPS (Frames Per Second). Pentru cei care abia acum descoperă seria aceasta de articole, iată definiţia: FPS sunt numărul de cadre pe secunda, deci numărul de imagini succesive pe care engine-ul unui joc le poate afişa într-o secundă.

Importanţa deosebită a FPS este evidentă – cu cât valoarea acestuia este mai mare, cu atât animaţia şi deplasarea jucătorului in joc este mai fluidă, şi invers, cu cât valoarea sa este mai mică, cu atât deplasarea jucătorului va fi mai sacadată. Practic, dacă valoarea FPS scade sub 24, această „sacadare" devine vizibilă, iar un FPS sub 12 face jocul... de nejucat.

În afară de asigurarea unui FPS mare, o altă grijă a programatorilor unui engine este păstrarea aproximativ constantă a valorii FPS. Acest lucru nu reuşeşte întotdeauna; de câte ori nu vi s-a întâmplat ca jocul să se „împotmolească" tocmai când eraţi atacat din mai multe părţi deodată?

Principalii vinovaţi pentru „împotmolirea" jocurilor (inclusiv în multiplayer) sunt tocmai... adversarii. Apariţia lor în câmpul vizual al jucătorului (pe ecran, deci) este cea care determină de cele mai multe ori acest „lag", această întârziere neplăcută care duce de cele mai multe ori la pierderi de frag-uri şi nervi descărcaţi asupra tastaturii nevinovate.

În articolul trecut am pomenit în trecere despre o metodă prin care se poate realiza menţinerea la un nivel constant a FPS. Spuneam atunci că numărul de cadre pe secundă nu poate fi păstrat la un nivel predefinit decât în condiţiile în care numărul de poligoane pe care computerul trebuie să le prelucreze şi să le afişeze la un moment dat este, la rândul său, constant (sau, cel puţin, nu depăşeşte o anumită limită).

Despre această metodă vom vorbi în acest articol. Ea poartă numele de LOD (Level Of Detail – nivel de detaliu). Pe scurt, ea permite modificarea în timp real a numărului de poligoane folosit pentru reprezentarea unor modele 3D, în funcţie de distanţa dintre aceste modele şi jucător.

Definiţie

Aşa cum spune şi numele, LOD se bazează pe modificare în timp real a detaliilor unui model. Dacă modelul 3D se află departe de cameră (de punctul din care jucătorul observa acţiunea jocului), atunci multe detalii ale modelului nu vor putea fi percepute de acesta. Dacă aceste detalii nu pot fi văzute oricum, atunci de ce să mă mai străduiesc să le afişez?

Să luăm un exemplu concret. Un personaj alcătuit din 1600 de poligoane se va vedea foarte bine de aproape. Dacă ne îndepărtăm de el, vom vedea însă că multe din detaliile interesante pe care le puteam admira din prim plan dispar. Nu mai vedem exact câte degete are, nici nu ne mai putem da seama că platoşa sa e formata din carapace separate, legate între ele. La un moment dat, din războinicul falnic va mai rămâne o pată colorată de zece pixeli înălţime.

La această distanţă, computerul se chinuie în zadar să afişeze platoşa, casca sau ochii personajului – ei sunt acolo, dar noi nu-i mai vedem.

Putem spune astfel că, după ce ne îndepărtăm la o anumita distanta de personaj, acesta ar putea fi alcătuit la fel de bine şi din

Iată o serie de detalii care să vă edifice mai bine asupra diferenţei calitative dintre cele trei personaje din imaginile anterioare. Se poate observa cum se reduce numărul de poligoane astfel încât din patru poligoane alăturate se alipesc, generând unul singur. Personajul cel mai îndepărtat va avea deci de patru ori mai puţine poligoane decât cel din prim plan.

Diferența dintre cele trei personaje nu este vizibilă atunci când mesh-ul este acoperit cu o textură.

500 sau 100 de poligoane, diferența de calitate nemaifiind vizibilă.

Aceasta este ideea pe care se bazează LOD - reducerea (sau creșterea) numărului de poligoane care alcătuiesc un personaj (sau orice alt element al unei lumi 3D) în funcție de distanța dintre personaj și cameră.

Avantajele LOD

Mai puține poligoane înseamnă, deci, mai puține informații de procesat, și eliberarea unei părți din puterea de calcul a computerului pentru alte activități. Și, credeți-mă, puterea de calcul nu va fi niciodată suficientă – mereu va fi nevoie de altceva, de mai mult – un AI mai bun, efecte speciale mai spectaculoase, sau, pur și simplu, mai mulți inamici sau o lume a jocului mai detaliată și mai „plină".

Această problemă se rezumă, deci, la economisirea numărului de poligoane și, prin aceasta, de vertex-i. (Tot pentru cei care nu au parcurs întreaga serie de articole, vă amintim că vertex-ii sunt vârfurile poligoanelor, punctele care le definesc.)

Dar economisirea numărului de vertex-i aduce după sine și alte avantaje. Animația personajelor, de exemplu, ține seama de numărul de vertex-i. Reducerea acestora va însemna reducerea numărului de transformări necesare pentru realizarea animației și, deci, economisirea unei alte bucăți din puterea de calcul... (ne învârtim în cerc în jurul acestei puteri de calcul). LOD este avantajos și în cazul obiectelor statice, nu numai în cazul animațiilor.

Astfel, dacă pe un stâlp de marmură aplicăm environment mapping (un efect care permite ca mediul înconjurător să se reflecte pe o suprafață), acest efect special cere calcule suplimentare pentru fiecare cadru, pe fiecare dintre poligoanele care alcătuiesc stâlpul. Economia de poligoane duce, din nou, la o economie de calcule.

Să examinăm acum aplicarea LOD în cazul jocului multiplayer. Să presupunem că o scenă (o arenă sau un nivel multiplayer) este construită în așa fel încât numărul maxim de jucători este prestabilit, să zicem, la 8. Ce se întâmplă dacă apare un al 9-lea jucător? Îi interzicem accesul sau acceptăm ca FPS-ul să scadă dramatic? Nici una, nici alta. Trebuie doar să aplicăm LOD pentru a ajusta numărul total de poligoane ale personajelor; astfel, cu o scădere minoră a calității modelelor, putem accepta un număr mai mare de jucători. Acest lucru ajută și la scalarea jocului; prin modificarea numărului de poligoane, putem face ca jocul să poată fi rulat (la un FPS onorabil) și pe sisteme mai slabe. Dacă placa grafică sau procesorul nu fac față, nici o problemă, numărul de poligoane va fi ajustat conform unor reguli prestabilite astfel încât utilizatorul să nu ajungă să citească mesaje de genul „Sorry, but your system should be studied by historians" (decât în cazul în care, într-adevăr, computerul respectiv merită un loc în muzeu – în vremurile crude pe care le trăim, cam orice sistem sub Pentium 150 și fără placă accelleratoare se înscrie în acest grup nefericit).

Modalități de implementare

O metodă de implementare a LOD este tehnica suprafețelor Bezier. Nu vă speriați, aceia dintre voi care au citit articolul precedent știu deja despre ce este vorba, căci suprafețele Bezier sunt doar un alt nume pentru suprafețele parametrice.

La ce servesc suprafețele Bezier? Ele aproximează, printr-o colecție de funcții matematice, suprafețele unui model 3D. Aceste formule matematice sunt folosite pentru a genera în timp real un număr mai mare sau mai mic de poligoane, construind astfel modelele 3D de care avem nevoie.

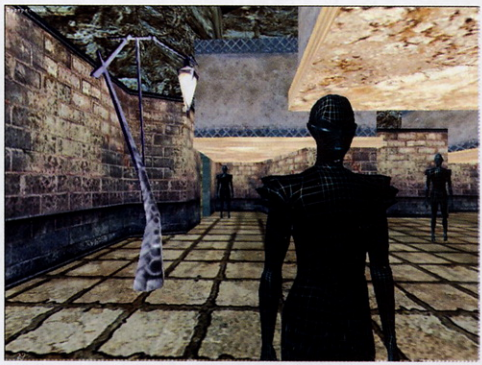
Avantajele folosirii suprafețelor Bezier sunt următoarele :

- generalitatea. Folosind un singur set de funcții se pot genera un număr practic infinit de modele.

- scalabilitatea. Numărul de poligoane generate prin modificarea parametrilor funcțiilor poate fi precis determinat, astfel încât să satisfacă nevoile LOD.

- permit reprezentarea precisă a suprafețelor curbe. Acest din urmă punct este foarte important, deoarece reprezintă un pas înainte în jocurile 3D – până acum, numai câteva jocuri (Quake 3, Messiah) au implementat un asemenea algoritm.

Această metodă de implementare a suprafețelor parametrice se bazează pe stocarea punctelor de control care generează curbele Bezier. Modelul 3D este reprezentat ca fiind alcătuit dintr-o mulțime de patches (petice) Bezier; fiecare astfel de petec se definește printr-o serie de puncte de control care îl definesc și pe baza cărora se generează funcțiile de aproximare. Cu ajutorul acestor funcții și a punctelor de control se poate împărți un petic în mai multe petice noi, făcând ca suprafața modelului 3D să fie din ce în ce mai netedă.



În wireframe, însă, putem vedea că există o diferență clară între personajul din prim plan, alcătuit dintr-un număr mai mare de poligoane și cele din planul a doilea.

Așteptăm comentariile voastre, pentru a aborda în articolele următoare și alte aspecte ale efortului de creație a unui joc; rămâne să ne scrieți și să ne spuneți care dintre aceste aspecte vă interesează în mod deosebit.

Claude

Jocul - Zeul Mileniului Trei

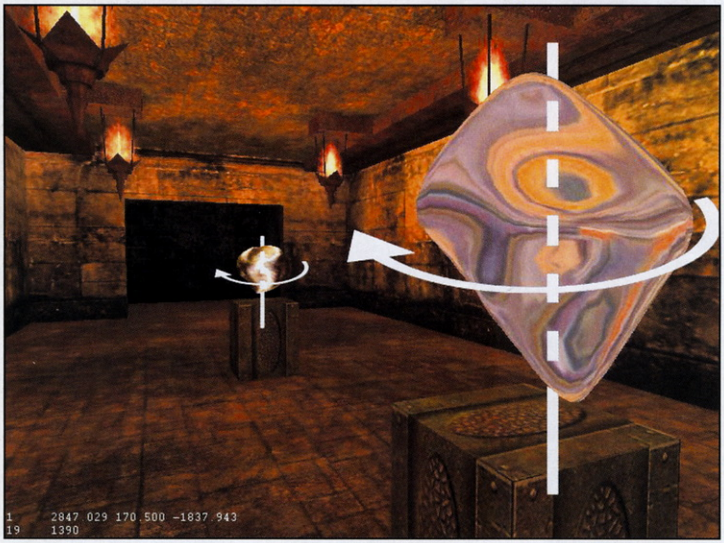
Unul din aspectele cele mai importante pentru „impresia artistică" pe care o face un joc asupra celor care îl privesc este animația. Aduceți-vă aminte de „*Prince of Persia*". Acest joc arcade tipic, în care jucătorul, în rolul unui tânăr viteaz, străbătea zeci de kilometri de temnițe pentru a o salva pe aleasa inimii sale, a impresionat în primul rând prin calitatea animațiilor personajului. Dacă faceți o comparație între acest joc și altele asemănătoare din acea vreme („*Jill of the Jungle*" sau „*Dangerous Dave*", de exemplu) vă puteți da seama cât de important a fost sistemul de animație pentru succesul jocului.

Acest exemplu se referă la un joc 2D, dar animația este la fel de importantă și într-un joc 3D. Să comparăm animațiile fluente și realiste ale unor jocuri ca *Unreal Tournament* sau *Half-Life* cu animațiile ciudate ale dinozaurilor din *Trespasser*. Dacă în *Unreal* animațiile sunt cumva puse în umbră de calitatea texturilor, ele sunt totuși suficient de bune ca să nu contrasteze. În *Half-Life*, animațiile au un rol foarte important în construirea atmosferei jocului și, chiar dacă mai au unele mici scăpări, sunt atât de bine concepute încât nu ne dăm seama de aceasta. În Trespasser, însă, defectele animațiilor sunt atât de mari încât nu pot fi trecute cu vederea (eu n-am reușit să ucid nici măcar un dinozaur din cauza balansului anormal al brațului care ținea arma - pur și simplu, mă amețea...).

Dar poate că importanța animațiilor se vede cel mai bine în alte tipuri de jocuri 3D, nu în first-person shooters. Comparați *NHL 98* cu *Actua Hockey* sau *Fifa 98* cu *World League Soccer* (am pus alături jocuri apărute cam în aceeași perioadă) și veți înțelege ce vreau să spun...

Dacă în jocurile 2D animațiile personajelor și ale obiectelor se bazau exclusiv pe sprites, secvențe de imagini succesive al căror principiu de funcționare ar trebui să va fie familiar, fiind același ca în cazul desenelor animate, pentru jocurile 3D animațiile se realizează într-un mod puțin diferit.

Să vedem mai întâi ce anume se poate anima într-un joc.



Animația obiectelor

Obiectele sunt acele elemente ale jocului asupra căruia jucătorul poate interveni (butoane, lăzi, mobilier, uși, lifturi etc.). În general, un obiect este unitar (format dintr-un singur element), astfel încât animația sa se aplică asupra obiectului în ansamblu.

Astfel, o ușă se deschide fie rotindu-se în jurul unei axe corespunzând poziției balamalelor, fie alunecând în lateral. În primul caz, se aplică o rotație în jurul axei fixe, predefinite; în al doilea caz, ușa suferă o translație pe o direcție fixă.

În jocuri nu vom întâlni prea des obiecte care să fie formate din mai multe părți. De aceea, animațiile obiectelor se reduc la rotații și translații, o animație putând fi astfel perfect definită printr-o formulă matematică.

Animația personajelor

Personajele sunt elemente ale jocului cu mult mai complexe (din punct de vedere al animatorului) decât obiectele. Explicația acestei complexități constă în alcătuirea personajelor - în general, ele sunt formate din mai multe elemente între care există conexiuni și reguli de mișcare. Un personaj umanoid, de exemplu, are un trunchi de care sunt legate picioarele și mâinile

prin articulații. Apoi, fiecare mână este alcătuită din mai multe elemente (degetele, palma, antebrațul, brațul, umărul) care sunt legate între ele și care au anumite libertăți de mișcare (nu se pot deplasa decât într-un anumit fel, în funcție de poziția celorlalte elemente).

Există mai multe metode de realizare a animațiilor care permit realizarea acestui tip de legături complexe. Ele tratează elementele care alcătuiesc personajele ca o ierarhie, stabilind cu exactitate ordinea de tratare a elementelor și influența pe care fiecare dintre ele o are asupra celorlalte.

Forward Kinematics

Forward Kinematics (termenul comun folosit este FK) este cea mai veche metodă de animație a unei ierarhii de obiecte. Folosind FK, fiecare element al unei ierarhii este tratat individual și în ordine - pentru ca un anumit element situat în partea finală a ierarhiei să fie pus într-o anumită poziție, toate celelalte elemente care îl preced trebuie să fie deplasate manual în poziția corectă. Practic, mișcarea elementelor înlănțuite se transmite numai de la elementul superior la cel inferior, de la „părinte" (elementul primar) la „copil" (elementul derivat).

Majoritatea programelor vechi de animație nu permiteau decât folosirea FK; principalul dezavantaj al FK devenea evident în momentul în care încercai să păstrezi un personaj cu piciorul lipit de pământ, să zicem, în timp ce își deplasa celălalt picior. Datorită structurii ordonate a FK, erai nevoit să miști piciorul, să îndoi genunchii, să flexezi articulația călcâiului și abia apoi să te asiguri că laba piciorului și-a păstrat poziția.

Același lucru se petrecea și în cazul menținerii fixe a mâinilor - pentru determinarea poziției palmei, trebuiau întâi deplasate elementele superioare ierarhic : brațul și antebrațul.

3.2. Inverse Kinematics

Inverse Kinematics (termenul comun folosit este IK) este cea mai folosită metodă de animație a unei ierarhii de obiecte. După cum spune și numele, ea este opusă ca mod de tratare metodei Forward Kinematics; astfel, în IK, fiecare element al ierarhiei este deplasat în așa fel încât un element inferior ierarhic să capete o anumită poziție sau orientare. Practic, în IK, mișcarea se transmite în ambele sensuri - în sus, de la copil la părinte, conform regulilor de IK stabilite, și în jos, de la părinte la copil, ca moștenire clasică FK, cu restricțiile care decurg din regulile de IK.

Folosind IK, atunci când un animator vrea să deplaseze laba piciorului unui personaj, în mod automat, piciorul se destinde sau se întinde conform regulilor de mișcare, astfel încât să se ajungă la efectul dorit (poziționarea corectă a labei piciorului). Atunci când corpul personajului se mișcă, elementele intermediare (șoldul, genunchiul) se deplasează astfel încât să mențină laba piciorului în poziția dorită. IK este deci mult mai comod de utilizat pentru animatori.

Un dezavantaj al IK îl constituie necesitatea existenței unui set de reguli foarte precise care stabilesc modul de deplasare al elementelor ierarhiei și limitele acestor deplasări. Astfel, pentru a realiza o animație realistă, limitele de destindere/întindere ale diferitelor părți care alcătuiesc un personaj de formă umană trebuie să fie cele care există și în realitate (capul nu poate fi răsucit la spate, de exemplu). Este foarte dificil să definești cu exactitate aceste reguli, iar implementarea unui sistem performant care să respecte cu strictețe aceste reguli este încă și mai dificilă.

Motion Capture

Motion capture (mocap, pe scurt) nu este practic o tehnică de animație, ci o tehnică prin care se pot înregistra informații reale asupra poziției și vitezei unui corp. Folosind motion capture, se pot înregistra mișcările unui actor, iar aceste informații pot fi folosite pentru a crea animații.

Evident, motion capture nu este o soluție decât pentru jocurile

realiste, în care personajele sunt ființe umane sau cel mult umanoide, aceasta deoarece există tentația de a importa direct informațiile înregistrate, fără a le mai prelucra, pentru a crește productivitatea. Dacă pentru anumite jocuri (cum ar fi simulatoarele sportive - vezi seria *FIFA*) motion capture este într-adevăr unealta potrivită, îmi imaginez cu greu cum ar putea fi ea folosită pentru a crea animația unui dragon sau a unui extraterestru miriapod.

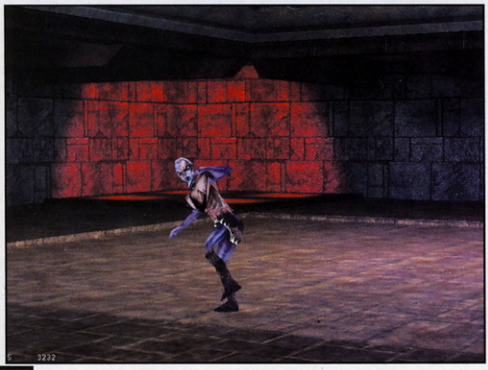
4. Stocarea animațiilor

După realizarea animațiilor, acestea pot fi stocate pentru a fi redate mai târziu, în timpul jocului. Există două metode „clasice" de stocare a animațiilor.

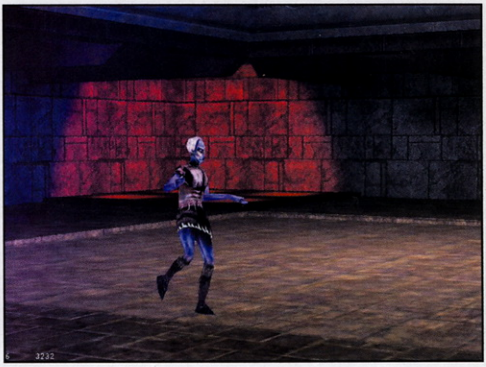
Prima metodă presupune stocarea tuturor cadrelor care compun animația prin înregistrarea poziției tuturor vertex-ilor în fiecare din cadre. Să presupunem că am realizat o animație în care un soldat aleargă. Tempoul animației (timpul în care ea trebuie să fie redată) este fix și se exprimă de obicei în cadre/secundă. Să zicem că animația noastră trebuie să dureze doua secunde și este formata din 30 de cadre. Pentru a stoca animația vom avea deci nevoie de spațiu de depozitare pentru de 30 de ori numărul de vertex-i din care este format soldatul.

Avantajul acestei metode este ușurința de redare a animației - computerul trebuie doar să citească informațiile stocate și să construiască modelul. Dezavantajul ei constă în dimensiunile uneori foarte mari ale fișierelor în care sunt stocate animațiile.

A doua metodă se bazează pe „oase". Oasele (organizate, evident, într-un schelet) sunt un sistem de tip ierarhic, asemănător sistemului folosit pentru realizarea animației, în care fiecărui element din ierarhie îi corespunde o anumită submulțime de poligoane. În funcție de poziția fiecărui os se poate calcula poziția poligoanelor asociate. Astfel, în loc să înregistrăm poziția tuturor vertex-ilor în fiecare cadru, vom înregistra numai poziția oaselor, la redare calculând dinamic, în funcție de această poziție, valorile vertex-ilor.

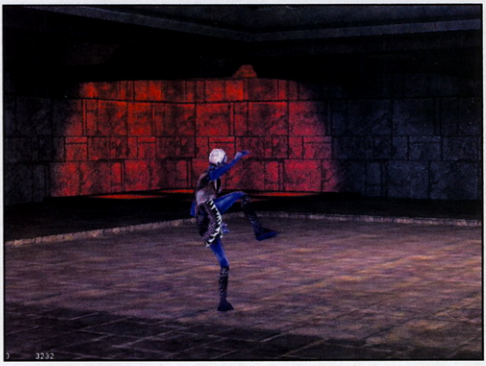


Avantajul acestei metode este deci că economisește foarte mult spațiu. Dezavantajul ei este că are nevoie de mai multe resurse la redare (deoarece valorile vertex-ilor trebuie calculate).



(imaginați-vă ce efect tip Charlie Chaplin ar apărea :).

Dacă vreți să vedeți o astfel de problemă în realitate, căutați demoul jocului „*Nightmare Creatures*"; eu n-am mai reușit să-l joc după ce mi-am cumpărat o placă Voodoo !!!



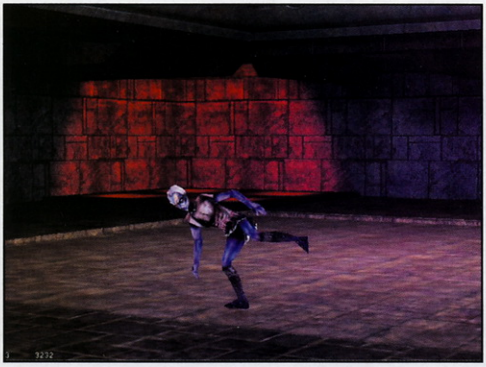
Implementarea efectivă a animațiilor

În funcție de modul de stocare ales, redarea se va face fie prin citirea efectivă a valorilor stocate ale vertex-ilor și construcția modelului pe baza acestora, fie prin citirea datelor și calculul vertex-ilor.

Indiferent însă de metoda de stocare, redarea animațiilor trebuie să țină seama de tempo-ul acestora.

Să presupunem că animația este redată pe două sisteme de calcul diferite, unul reușind să randeze 30 de cadre pe secundă, iar celălalt 60 de cadre pe secundă. Dacă nu ar exista un sistem de control al animației, o animație de 30 de cadre, gândită să dureze 2 secunde, ar dura doar o secundă pe primul computer și o jumătate de secundă pe al doilea Motivul : toate animațiile erau mult prea rapide, depășindu-mi reflexele...

Iată de ce este nevoie de un sistem de interpolare a animației. Interpolarea presupune crearea unor noi cadre de animație pornind de la valorile deja existente. Pentru ca animația să fie la fel de rapidă indiferent de viteza computerului, prin interpolare se realizează o adaptare dinamică a numărului de cadre al animației la numărul de cadre pe secundă realizat pe computerul respectiv. Practic, cu cât FPS-ul este mai mare, cu atât se generează un număr mai mare de cadre pentru a păstra constantă durata animației. Invers, pentru un FPS scăzut, se mai elimină din cadrele animației.



Interpolarea poate fi folosită şi pentru asigurarea unei tranziţii line între două animaţii. Atunci când se trece de la o animaţie la alta, pentru asigurarea unei treceri corecte de la una la alta nu exista decât trei soluţii :

- să se creeze animaţiile astfel încât să se poată imbrica perfect;
- să se creeze o animaţie intermediară;
- să se interpoleze porţiunile extreme ale celor două animaţii;

Evident, această interpolare dă rezultate bune numai dacă animaţiile sunt apropiate ca poziţie şi orientare a mişcării.

O altă problemă care apare la implementarea animaţiilor este legată de gestiunea „legăturilor" dintre oase. Atunci când stocăm animaţia ca un schelet, ca un sistem ierarhic de oase, vor exista întotdeauna zone ale skin-ului personajului care vor fi influenţate simultan de mişcarea mai multor oase. Aceste zone necesită o tratare specială pentru ca în skin să nu apară fisuri. O metodă care permite gestionarea acestor zone este metoda numita vertex blending.

Prin vertex blending se înţelege „amestecarea" vertex-ilor de graniţă ai regiunilor care se unesc în zona de influenţă despre care am vorbit mai sus. Să spunem că avem o animaţie în care un personaj îşi flexează braţul. În acest caz, zona cotului este o zona în care asupra skin-ului acţionează două oase. Pentru ca la flexarea completă în cot să nu apară o fisură, trebuie ca vertex-ii celor două părţi ale skin-ului (antebraţul şi braţul) să se îmbine.

Soluţia moderne

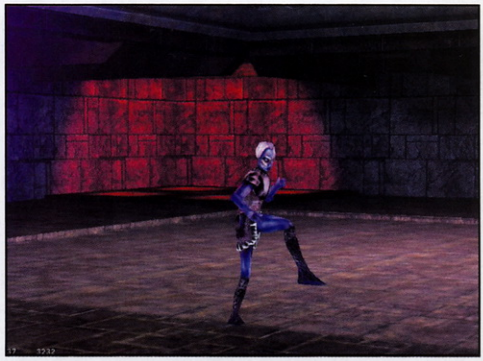
Pentru stocarea animaţiei se alege metoda ierarhiei de oase, în primul rând datorită necesităţilor de stocare reduse implicite, care ne permit să stocăm un număr mare de cadre ocupând în acelaşi timp un spaţiu mic de memorie.

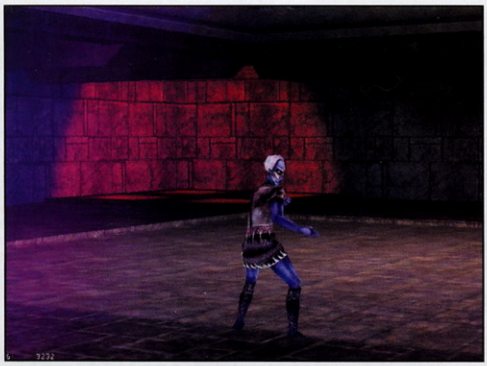
De asemenea, sistemul ierarhiei de oase permite dezvoltarea unei metode de interpolare rapidă între cadrele animaţiei, investiţie utilă pentru a asigura un comportament identic al jocului pe maşini diferite. În ceea ce priveşte implementarea animaţiilor, aceasta s-a făcut în integrare perfectă cu programele folosite pentru crearea animaţiilor, precum şi cu modul de stocare a personajelor.

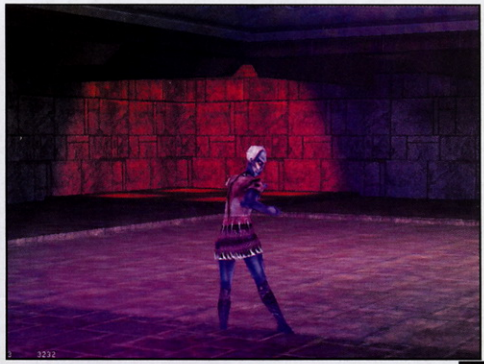
Personajele jocului sunt realizate folosind „petice" Bezier, o tehnică ce permite o variaţie semnificativă a numărului de poligoane al personajelor. Astfel, se stochează un număr relativ mic de vertex-i pentru fiecare personaj, urmând ca în funcţie de puterea de calcul a sistemului folosit să se genereze un număr mai mare sau mai mic de vertex-i.

În cele din urmă, se foloseşte vertex blending pentru ameliorarea aspectului animaţiilor şi eliminarea efectelor neplăcute („ruperi" ale skin-urilor, fisuri etc.).

Claude







Jocul - Zeul Mileniului Trei

Grafica unui joc este primul lucru care ne impresionează, iar efectele speciale au devenit deja o prezență obligatorie pentru orice joc ce se respectă

Efecte speciale - efecte artistice adăugate unei producții video pentru o creștere a calității acesteia prin crearea suspansului, ameliorarea atmosferei sau creșterea impactului pe care îl are scenariul asupra privitorului. Efectele speciale pot varia de la adăugarea unor motive vizuale sau sonore limitate sau mixarea mai multor imagini video până la efecte digitale sofisticate cum ar fi deformarea imaginilor, morphing-ul sau efecte 3D.

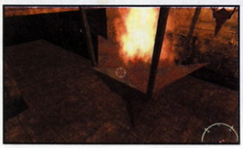
Citind această definiție (preluată dintr-unul din site-urile de referință în cultura digitală a zilelor noastre, www.visualfx.com) se observă accentul pus pe rolul efectelor speciale. Prezența lor în filme și jocuri se justifică, astfel, prin nevoia de a adăuga lumii prezentate spectatorului acele elemente care, chiar dacă nu sunt esențiale, ajută la o mai bună imersiune și transformă extraordinar modul de percepție al spectatorului. Într-o exprimare mai puțin pompoasă, putem spune că efectele speciale sunt o componentă importantă a ceea ce formează „sarea și piperul" jocurilor. Putem trăi și fără ele, dar totul ni s-ar părea atunci fără gust...

Iată deci care este subiectul articolului din aceasta lună. După prezentarea elementelor esențiale ale unui motor grafic 3D (partiționarea spațiului, collision detection) și după abordarea unor aspecte mai tehnice (scalabilitate, level of detail), iată-ne ajunși la aspectele mai „frivole" ale muncii de creație a unui joc (n-am vrut să spun că realizarea FX ar fi mai ușoară, ci doar – parcă – mai distractivă... Voi încerca să prezint în continuare principalele tipuri de efecte speciale întâlnite în jocuri, însoțite de precizări privind modalitățile de implementare specifice.

Nu-i așa că arată realist?

Flăcări, fum, explozii

Am grupat toate aceste efecte deoarece sunt asemănătoare din punct de vedere al implementării. Cea mai utilizată tehnică de simulare a flăcărilor este folosirea unor texturi care, afișate unele după celelalte, să dea senzația unui foc care arde. Aceste imagini sunt aplicate pe un plan (un dreptunghi) care este orientat tot timpul către cameră. Practic, reușita simulării depinde în acest caz de realismul texturilor afișate. Datorită ușurinței implementării și a numărului mic de resurse procesor necesare, această tehnică este întâlnită în majoritatea jocurilor 3D.



O altă tehnică de simulare a flăcărilor este folosirea unor particule suprapuse unele peste celelalte a căror mișcare este calculată după formule care simulează mișcarea flăcărilor. Astfel, fiecare din particulele (flăcările) care formează focul este afișată peste celelalte particule, iar poziția particulelor este calculată la fiecare cadru al animației. Efectul este cu atât mai convingător cu cât se afișează mai multe particule (mai multe texturi). Dezavantajul major al acestei tehnici constă în calculele necesare, care trebuie reluate pentru fiecare cadru al animației. Avantajul ei, însă, este eliminarea repetiției care apare în animațiile obișnuite. Această tehnică poate fi ameliorată prin simplificarea formulelor de calcul. Cel mai reușit exemplu de implementare a acestei tehnici o constituie Unreal. În acest joc, exploziile și flăcările sunt generate în timp real folosind o tehnică asemănătoare.

În unele cazuri, cele două tehnici sunt combinate. Astfel,

pentru a mări realismul exploziilor, în *Half Life* se folosesc două planuri perpendiculare pe care sunt afișate texturile de explozie, în timp ce tehnica particulelor este folosită pentru „distrugerea" obiectelor. Evident, în acest caz, particulele nu mai sunt bitmap-uri, ci obiecte 3D de mici dimensiuni.

Pentru simularea fumului se folosesc texturi afișate pe planuri distincte, situate la distanțe relativ mici unele de celelalte. Varianta cea mai simplă presupune repetarea aceleiași texturi, scalarea și modificarea transparenței texturii dând efectul de „pierdere în depărtare" a fumului. Pentru a crea o simulare mai apropiată de realitate se pot folosi mai multe texturi și se pot complica „formulele de legătură" între planurile care formează dâra de fum.

Efectele „optice": reflexia și refracția

Această clasă de efecte speciale se bazează pe folosirea legilor fizicii pentru a crea efecte optice asupra unor elemente ale lumii jocului.

Reflexia constă în desenarea pe suprafața unui obiect a unei imagini care să dea impresia că obiectul reflectă mediul înconjurător.

Cea mai simplă metodă de implementare a reflexiei poartă numele de environment mapping. Aceasta constă în aplicarea a două texturi peste obiectul reflectant : textura de bază, care stabilește materialul din care este alcătuit obiectul și textura de mediu. Textura de mediu este o textură care simulează reflecția; ea este predefinită și se obține cel mai ușor prin randarea celor șase „puncte de vedere" principale ale obiectului, rezultatul randarilor (cele șase imagini) fiind compus ulterior într-o singură textură.

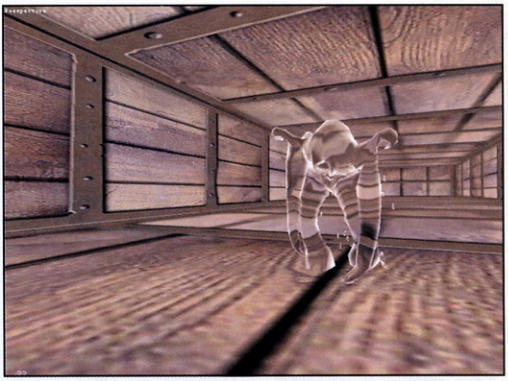
Dezavantajul principal al acestei metode constă în faptul că textura nu se modifică în funcție de mediu. Astfel, dacă jucătorul se află în fața unei sfere în care vede reflecția camerei în care se află, el nu se va putea vedea pe el însuși (folosirea environment mapping nu satisface cerințele necesare unei oglinzi). Avantajul principal al metodei constă în ușurința implementării ei și în consumul mic de resurse (textura fiind predefinită, „efortul" computerului constă doar în afișarea texturii); în plus, efectul este destul de convingător...

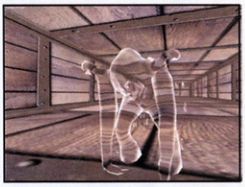
O metodă mult mai apropiată de realitate este folosirea ca textură de mediu, în locul unei imagini predefinite, a imaginii pe care o vede obiectul când se uită spre jucător (spre cameră). Folosind această metodă, textura de mediu este generată în timp real, prin randarea scenei în care se află jucătorul din punctul de observație situat în centrul obiectului. Astfel, textura de mediu este generată din nou pentru fiecare deplasare a jucătorului, constituind un calcul suplimentar.

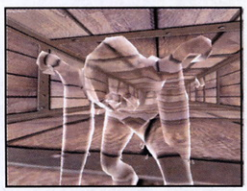
Acesta este de fapt principalul dezavantaj al metodei, ea presupunând o randare suplimentară a scenei pentru fiecare obiect reflectant (ca să nu mai vorbim de calculele complicate de creare a reflexiilor multiple – o sferă care se reflectă în altă sferă care se reflectă în alta sferă care se reflectă... Evident, realismul acestei metode este mult mai mare decât al environment mapping, efectul vizual fiind cu mult mai spectaculos.

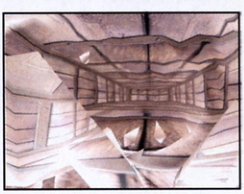
O aplicație particulară a reflecției o constituie oglinzile. Ele sunt un caz particular de obiect reflectant, fiind în general plane. Implementarea lor este o simplificare a metodei propuse anterior, în sensul că pentru texturarea oglinzii se folosește o singură textură – textura de mediu.

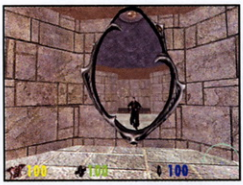
Refracția constă în deformarea imaginii obiectelor ca urmare a fenomenului optic

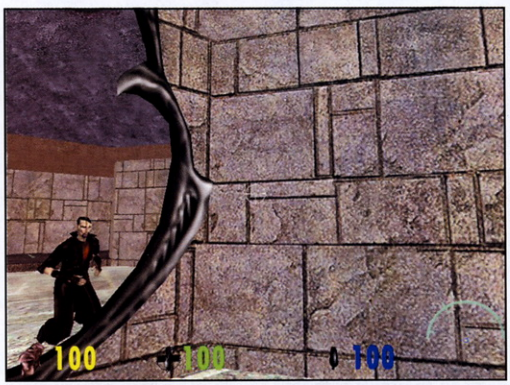












Oglinda privită din mai multe unghiuri

O mică secvență gen Predator realizabilă cu ajutorul tehnicii de refracție

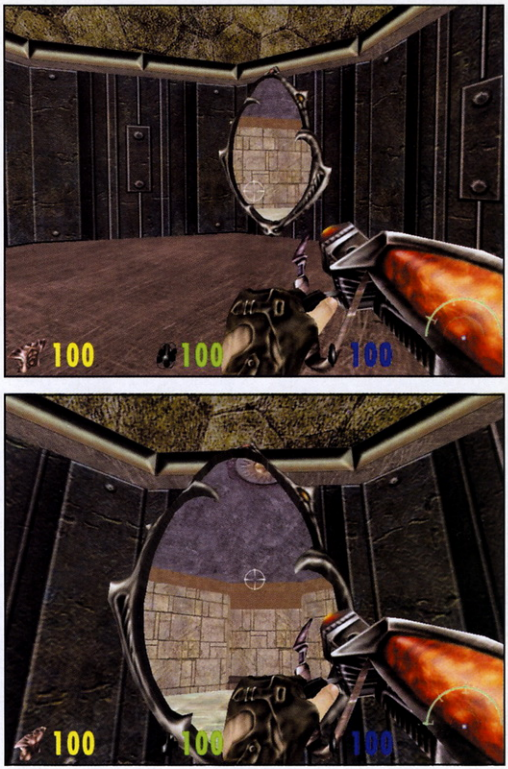
cunoscut (în general în apă sau în fronturi de aer).

Implementarea refracției se face prin generarea unei texturi de mediu aplicată obiectului pe care se produce refracția. Textura de mediu se obține prin randarea scenei din punctul de observație al jucătorului, dar, atenție, fără a include și obiectul. Partea din imaginea rezultată care s-ar fi suprapus peste obiectul „refractant" se folosește ca textură, calculând coordonatele de mapare după formulele de refracție din orice carte de fizică.

Un efect special „rezultat" din implementarea tehnicilor de reflecție și refracție este simularea apei. O apă foarte realistă se poate obține folosind efectele de reflexie și refracție pe o suprafață care se deformează ca suprafața unei ape. „Valurile" necesare pot fi obținute folosind funcții trigonometrice; prin compunerea funcțiilor se pot obține efectele de ciocnire a valurilor.

Un alt efect optic interesant este folosirea portalilor (a teleportoarelor). În acest caz, un portal este (nu uitați că vorbim despre efecte speciale !) o suprafață pe care se vede imaginea unei alte zone a jocului; la trecerea jucătorului prin portal, el va ajunge în acea zonă.

Metoda de implementare este foarte simplă: suprafața - portal va fi texturată cu o textură obținută prin randarea zonei-țintă. Altfel spus, mai întâi se randeaza zona-țintă, apoi se texturează poligoanele care formează portalul. Această operație se reia la fiecare frame; din acest motiv efectul vizual al portalilor este mare consumator de resurse.



Efectul este asemănător unor oglinzi, dar...

Se poate observa foarte ușor efectul pe care îl produce mișcarea valurilor.

Efectele de deformare: morphing și metaballs

Evident, efectele de deformare se bazează pe schimbarea formei obiectelor, deci pe modificarea numărului vertex-ilor sau a poziției acestora în spațiu.

Morphing-ul este tranziția unui obiect tridimensional de la o forma inițială la o forma finală, tranziție care se face uniform și continuu, asemenea unei metamorfoze.

Cea mai uzuală metodă de implementare a morphing-ului presupune că forma inițială și cea finală a obiectului au același număr de vertex-i. Astfel, pentru trecerea de la forma inițială la cea finală se schimbă coordonatele vertex-ilor – se interpolează, de fapt, în timp, poziția vertex-ilor. Să presupunem că la momentul inițial, t0, toți cei n vertex-i au pozițiile xi0, yi0 și zi0 (i=1..n). La momentul final, t1, vertex-ii vor avea pozițiile xi1,yi1 și zi1. Fiecare vertex se deplasează cu viteza constantă între pozițiile xi0, yi0, zi0 și xi1,yi1,zi1, astfel încât transformarea este uniformă și continuă.

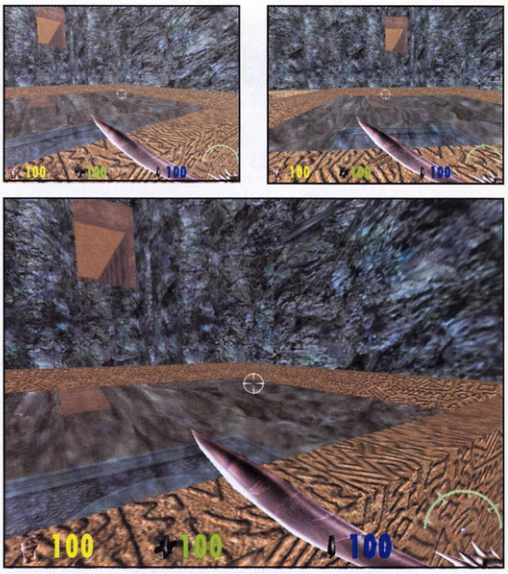
Dezavantajele acestei metode sunt evidente. Mai întâi, legătura între poziția inițială și cea finală a aceluiași vertex trebuie să fie atent făcută, pentru a evita transformările haotice (în cursul cărora obiectul ar ajunge să semene cu o pernă de ace). Apoi, metoda trebuie puțin modificată pentru a permite transformări substanțiale de formă, prin introducerea unor puncte intermediare. Dacă morphing-ul de la un cub la o sferă nu ridică

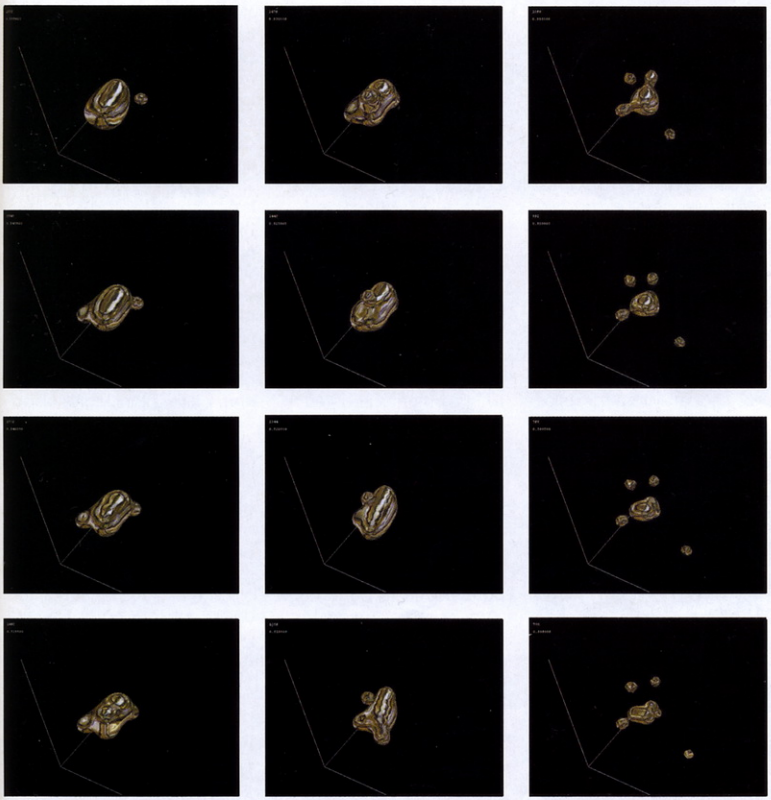
probleme, fără existența unor forme intermediare nu se poate face un morphing „estetic" între un personaj umanoid și o insectă, de exemplu.

Un alt exemplu elocvent al limitărilor morphing-ului îl dă un joc celebru – *Quake*. Aici, animația personajelor se bazează pe morphing. În multiplayer se poate observa foarte bine cum, atunci când ceilalți jucători se uită în sus sau în jos, se rotește tot corpul personajului, și nu numai capul și/sau arma. Acest lucru se întâmplă fiindcă morphing-ul nu permite animații separate pentru porțiuni ale unui obiect, el tratând obiectul în întregime.

Tehnica metaballs simulează efectul forțelor de atracție ce apar la suprafața unor obiecte 3D. La o distanță suficient de mică, obiectele apropiate se unesc într-un singur corp. Efectul vizual seamănă cu mișcarea și contopirea unor picături de mercur.

Implementarea metaballs se bazează pe calculul unei funcții în fiecare punct al spațiului tridimensional în care se deplasează obiectele-metaballs. Această funcție (în general distanța dintre centrele metaball-urilor) este folosită pentru a genera o suprafață în spațiul 3D care să aproximeze mulțimea punctelor care satisfac aceeași condiție. Această suprafață este corpul complex rezultat din „unirea" metaball-urilor. Generarea propriu-zisă a suprafeței se face folosind un set de mesh-uri predefinite numite marching-cubes. Aceste mesh-uri aproximează fragmente mici





o secvență foarte sugestivă de metaballs

de suprafață precalculate pentru anumite seturi de valori extreme ale funcției. Metoda seamănă cu rezolvarea unui puzzle - avem piesele (marching cubes) și trebuie să construim cu ele imaginea tridimensională a unei funcții.

Avantajele acestei implementări a tehnicii metaballs sunt următoarele : generează un mesh independent la fiecare cadru al animației, nu necesită obiect inițial și obiect final, nu necesită un număr constant de vertex-i sau de poligoane.

Efectele speciale sunt un subiect care a atras întotdeauna foarte mult atenția oamenilor. Sperăm că prin aceste material am eliminat câteva din semnele de întrebare ce planau asupra modului de realizare a acestora. Această rubrică și-a propus să vă prezinte câteva din tainele producerii unui joc, iar dacă acest obiectiv a fost atins, voi, cititorii, decideți. Așteptăm deci, părerile voastre precum și propunerile pentru viitoare subiecte ale rubricii pe adresa redacției - level@chip.ro.

Claude

Jocul - Zeul Mileniului III

În perioada în care activam încă la revista LEVEL am reușit să scriu, cu ajutorul lui Adrian Filippini (pe atunci director general al FunLabs), o serie de articole ce purtau titlul de mai sus. Acele articole au prezentat, pe parcursul a mai multe numere ale revistei, câțiva dintre pașii importanți și cunoștințele de bază pe care trebuiau să le cunoască orice tânăr sau grup ce dorea să-și încerce norocul în producția de jocuri. Din motive obiective, la un moment dat, din păcate mult prea devreme, acea serie de articole a fost întreruptă. Cu această ocazie doresc să redeschid acesta serie de articole fiind convins că voi, cititorii acestei publicații, încă nu v-ați pierdut speranța experienței unui joc sută la sută românesc.

Din nefericire producția autohtonă de jocuri este la pământ și nu pentru că nu am avea programatori și artiști buni ci pentru că nimeni nu are încă curajul să investească într-un asemenea proiect. Idei au fost și încă sunt, dar fără un plan coerent și fără o investiție șansele de reușită ale unui astfel de proiect sunt extrem de scăzute.

În acest caz ce ne rămâne de făcut? În nici un caz să abandonăm. Trebuie doar să ținem seama de anumiți factori importanți și să regândim strategia.

În continuare am să vă prezint pe scurt câteva puncte de reper pe care să le aveți în vedere atunci când entuziasmul unei idei de joc vă va cuprinde, asta în eventualitatea în care vă și gândiți serios să o transformați în realitate.

1. Notează tot.

Oricât de trăznită, banală sau ciudată pare această afirmație am pus-o prima deoarece este una din acțiunile omise de tinerii întreprinzători. Oriunde te-ai afla, în orice situație să ai la tine o agendă și un pix pentru a nota orice idee legată de joc ce poate apărea. Odată ajunși acasă filtrați-vă ideile și concentrați-le într-un design doc. Asupra acestui aspect voi reveni mai pe larg în viitor.

2. Concepe un plan de dezvoltare

Odată ce ideile de joc apar este absolut necesar să începi încropirea unui plan de dezvoltare sau producție a jocului. Acesta trebuie să conțină câteva elemente importante:

- Necesarul tehnic (echipamente, rețele, etc.);
- Necesarul uman (ce oameni și ce specializări);
- Necesarul financiar (este evident la ce mă refer);
- Documentația necesară (ce lucrări aveți nevoie și de unde le procurați);
- Planul de lucru în timp (în care se regăsesc termenele de finalizare ale fiecărui proces în parte).

3. Concepe un plan de afaceri

Al treilea pas firesc este conceperea planului de afaceri pe baza căruia să încercați atragerea unei investiții (de preferat un om de afaceri deschis noilor oportunități). La rândul său acest plan de afaceri trebuie să conțină câteva elemente precum: planul de dezvoltare (prezentat anterior), o descriere nu foarte detaliată a jocului sau proiectului, o analizÎ a pieței internaționale de jocuri PC și o estimare financiară profitabilă (reală).

4. Coeziune

Mulți și-au închipuit că pot produce un joc înjghebând o echipă de entuziaști de prin toate colțurile țării. Nimic nu poate fi mai departe de adevăr. Echipa de lucru trebuie strânsă într-un singur loc. Astfel se creează un climat de colaborare între membrii echipei, ideile noi sunt discutate mai aprins și mai eficient, iar apoi vor fi și aplicate cu un grad de succes mult mai ridicat.

De asemeni în ochii unui posibil investitor o echipă unită, serioasă și disciplinată constituie un argument în plus în favoarea alocării unor sume proiectului prezentat.

5. Ambiție și determinare

Producția unui joc nu este un lucru ușor de realizat și nici nu este la îndemâna oricui. Entuziasmul inițial se stinge repede dacă nu se întrevăd rezultate palpabile într-un timp relativ scurt. Iar aceasta este o problemă a românilor. Ne așteptăm mereu să dăm marea lovitură din prima încercare, într-un timp foarte scurt și cu minim de efort. Pe parcurs vă veți lovi de nenumărate dificultăți, unele din vina voastră, altele datorită lipsei de experiență, altele pur și simplu datorită providenței. Important este ca odată porniți pe acest drum să nu renunțați și să continuați să credeți în ideea voastră, în proiectul vostru și, nu în ultimul rând, în succesul vostru.

În numerele viitoare voi aborda alte aspecte delicate ale acestei probleme. În același timp aștept întrebările voastre legate de acest subiect pe adresa redacției:

office@dark-times.com

și promit că vă voi răspunde în cel mai scurt timp cu putință.

Claude

Jocul - Zeul Mileniului III

Înainte de a mă apuca să vă povestesc ce trebuie să faceți pentru a avea succes în inițiativa voastră de a produce un joc, va trebui să vă atrag atenția asupra câtorva aspecte ce sunt ignorate sau asumate în mod eronat.
În primul rând foarte mulți dintre cei care încearcă să producă un joc în România au proasta impresie că abordarea unei companii care să le cumpere și să le distribuie jocul este floare la ureche. GREȘIT!!!
Pentru ca o astfel de companie să fie interesată de proiectul vostru nu este suficient să îi informați despre existența lui. Va trebui să le dovediți câteva lucruri.
Primul dintre acestea este că ideea și conceptul vostru de joc se va vinde. Un publisher este în primul rând interesat de profitul pe care îl poate face de pe urma oricărui proiect, nici unul dintre ei nu va investi doar de dragul de a da bani unor tineri români.
Pentru aceasta va trebui să le prezentați un „design document“ foarte detaliat care să cuprindă ideea jocului, concepția sa, modul de dezvoltare a jocului, termenele de finalizare pe fiecare element în parte (engine, animații, cutscene-uri, modelare, poveste, etc.), un studiu de marketing al pieței căreia se va adresa jocul respectiv, precum și o prognoză financiară preliminară.
În al doilea rând va trebui să dovediți că sunteți capabili să duceți la capăt proiectul pe care l-ați propus. Pentru aceasta va trebui să le trimiteți un CV complet al echipei de lucru, unde să poata fi regasită experiența și cunoștințele fiecărei persoane implicate în proiect, alături de un tech-demo. Voi reveni în alt număr cu detalii despre cum trebuie să arate și ce trebuie să conțină un tech-demo.
Revenind la percepțiile greșite, la fel de mulți tineri consideră că este suficient să trimită e-mail-uri către publisher în care să prezinte o idee de joc pentru ca aceștia să fie subit interesați de proiect. Din nou GREȘIT!!!
Poate nu vă vine să credeți dar oamenii care citesc aceste mesaje primesc zilnic sute, chiar mii de astfel de informări, astfel încât șansa ca AL TĂU să îi atragă atenția este infimă. Cel mai bun mod de a intra în această lume și de a contacta oamenii cheie este participarea la evenimentele internaționale (precum E3 sau ECTS). O altă metodă cu șanse de reușită este trimiterea unui plic care să conțină toate elementele amintite mai devreme (design doc, CV, tech-demo) prin poștă către publisher (în această situație este de preferat să cunoașteți numele persoanei care se ocupă de selecția proiectelor).
Dacă în aproximativ 30 de zile nu primiți nici un răspuns la apelul făcut, atunci mai trimiteți un e-mail în care să îi întrebați politicos dacă au avut posibilitatea să se uite peste proiectul vostru. În cazul în care nici acest demers nu dă roade nu are rost să mai insistați, înseamnă că proiectul vostru nu a fost luat în considerare.
Pentru a avea succes în demersul către publisher este bine să țineți cont de câteva aspecte.
Unul dintre acestea este reputația și notorietatea companiei căreia vă adresați. Cu cât compania este mai mare și mai puternică cu atât scad șansele ca proiectul vostru să ajungă

pe biroul factorului de decizie. Este puțin probabil ca un proiect prezentat de Ionescu Games să treacă măcar de portarul de la Electronic Arts.
Un alt aspect, demn de luat în seamă, este vechimea pe piață a companiei. O firmă tânără va fi mai receptivă la proiecte noi, chiar dacă vin din partea unor necunoscuți.
La fel de importante sunt și jocurile distribuite deja de către companii. Dacă au mai distribuit jocuri în genul celor propuse de voi, aveți șanse să le treziți interesul.
O altă impresie eronată este aceea că odată trezit interesul unei companii semnarea contractului este doar o formalitate. După ce o companie se arată interesată de proiectul vostru vor urma câteva luni (între 3 și 12) de negocieri, tatonări și schimburi de informații până când se va ajunge la o propunere concretă. Din acest moment intră în acțiune avocații, și nici nu vreți să știți ce urmează mai departe.
Chiar și semnarea unui contract cu o companie nu constituie o garanție că jocul vostru va ajunge pe piață. Publisher-ul va putea tăia finanțarea în orice moment dacă prognozele și studiile de piață efectuate nu sunt favorabile jocului dezvoltat. Ce trebuie să rețineți este că producția de jocuri este o treabă foarte serioasă și dificilă.
În numerele viitoare voi aborda alte aspecte delicate ale acestei probleme. În același timp aștept întrebările voastre legate de acest subiect pe adresa redacției:

office@dark-times.com

și promit că vă voi răspunde în cel mai scurt timp cu putință.

Claude

Jocul - Zeul Mileniului III

După ce v-am bătut atat la cap cu generalități (mare parte din ele negativiste) a sosit momentul să intrăm în pâine, cum se spune. Am să încep astăzi să vă povestesc câteva lucruri legate de unul din primii și cei mai importanți pași: design-ul.
Cu toții am jucat jocuri, cu toții am descoperit elemente care ne-au plăcut, altele care nu ne-au plăcut și, nu în ultimul rând, cu toții am visat la anumite lucruri pe care le-am dori în jocuri. Însă atunci când începem să punem pe hârtie ideile pe baza cărora se va dezvolta ulterior design doc-ul trebuie să ținem minte faptul că jocul pe care îl vom crea este destinat „lor", nu „nouă". Am spus că design-ul este un pas extrem de important. Asta deoarece el decide în proporție de 80% succesul viitorului joc.
Prima întrebare care ar trebui să și-o pună un game designer este: Ce se așteaptă de la un joc? Răspunsul, probabil evident, este o experiență unică, exuberantă, antrenantă, plăcută și, mai ales, relaxantă. Toate bune și frumoase... dar cum reușim acest lucru?
Teoretic este simplu, trebuie să eliminăm din joc toate acele elemente frustrante și care pot determina jucătorul să renunțe și să creăm situații unice și distractive în care jucătorul să se manifeste după pofta inimii.

Unicitate
Mulți designeri se iau cu mâinile de păr când aud că trebuie să conceapă soluții unice pentru diverse situații. Să fim serioși, s-au produs enorm de multe jocuri astfel încât apariția unei idei, cu totul nouă și unică, este improbabilă. Greșeala constă însă în modul de interpretare a acestei cerințe. O soluție unică nu înseamnă neapărat ceva la care nimeni nu s-a mai gândit să facă până acum. Ea înseamnă de fapt să permiți jucătorului să folosească, pentru rezolvarea unei situații din joc (un quest, o luptă, etc) soluțiile sale unice. Haideți să luăm un exemplu că să înțelegeți mai bine.
În foarte multe jocuri RPG am fost nevoiți la un moment dat să trecem de un anumit obstacol cu ajutorul unor plăci de presiune (o ușă secretă care se deschide atunci când pe o anumită dală din pardoseală se așează o greutate). O soluție evidentă este să duci personajul deasupra dalei pentru a o acționa cu greutatea lui. O altă soluție, la fel de folosită și de evidentă, este amplasarea pe dala respectivă a unui bolovan sau a unei lăzi găsite prin apropiere. Ce se întâmplă însă dacă jucătorul se decide să depună pe dală o armă sau o armură? Va fi trapa acționată? Dar dacă acel jucător controlează un mag ce poate crea un monstru, va acționa greutatea monstrului „chemat" dala respectivă? Dar dacă magul respectiv lovește dala cu „Meteor Shower"? Toate acestea sunt soluții perfect valabile la care un jucător se poate gândi. Dar dacă trapa respectivă a fost configurată din concepție să fie acționată numai de greutatea personajului sau a bolovanului, atunci jucătorul se va afla în situația de a fi găsit o soluție logică la care jocul nu va răspunde așa cum ar trebui.
Și atunci vă veți întreba care ar fi soluția. Ați putea, de exemplu, să atribuiți fiecărui element din joc o greutate, indiferent dacă este un obiect real sau

unul magic. Apoi să setați pentru trapa respectivă o greutate minimă necesară pentru a fi acționată. Astfel în momentul în care jucătorul se va apuca să depoziteze „obiecte“ pe trapa respectivă va reuși să o acționeze în momentul în care greutatea critică a fost atinsă.
Acest procedeu poate fi folosit pentru aproape orice situație dintr-un joc. Va trebui să încercați să anticipați aproape orice lucru la care se va gândi un jucător, chiar dacă pare imposibil.
În numerele viitoare voi aborda alte aspecte delicate ale acestei probleme. În același timp aștept întrebările voastre legate de acest subiect pe adresa redacției:

office@dark-times.com

și promit că vă voi răspunde în cel mai scurt timp cu putință.

Claude

Întrebarea săptămânii

Să presupunem că tocmai v-ați dus la un interviu pentru a vă angaja la un studiou ce urmează să producă un joc PC.

Care este postul pe care doriți să-l ocupați și de ce?

Aștept răspunsurile și argumentele voastre pe adresa: office@dark-times.com

Jocul - Zeul Mileniului III

Vă povesteam săptămâna trecută cât de important este să lăsați jucătorului dreptul de a-și folosi propriile idei și soluții pentru rezolvarea unei situații de joc. Acesta a fost unul din motivele principale pentru care jocuri precum Civilization sau Master of Orion au avut un succes atât de mare. Astăzi vom vorbi despre un alt aspect la fel de important: soluțiile multiple.

În trecut, foarte multe jocuri erau abandonate deoarece jucătorul se împotmolea la un moment dat. Fie că nu găsea soluția potrivită pentru rezolvarea unui puzzle, fie că monstrul pe care trebuia să-l învingă era prea puternic. Atunci și-au data seama producătorii de jocuri că este benefic pentru joc să permită rezolvarea situațiilor prin mai multe metode, renunțându-se astfel la soluția unică.

Ca să înțelegeți mai bine vă voi prezenta unul dintre cele mai simple cazuri, cel al ușii încuiate. Soluția evidentă este aceea de a lua cheia din sertarul de alături și de a descuia ușa. Dacă aceasta ar fi singura soluție de trecere atunci jocul ar fi nu numai nerealist, dar ar și creea probleme (frustrări) jucătorilor care nu deschid acel sertar în care se află cheia. Din acest motiv acea ușa ar trebui să poată fi deschisă și prin alte metode:

1. Folosirea unui șperlaclu.
2. Dărâmarea ușii cu piciorul.
3. Bătut la ușa pentru a determina persoana din cameră să deschidă ușa.
4. Dinamitarea ușii sau distrugerea ei cu armamentul din dotare.
5. Deschiderea geamului de lângă ușă.
6. Capturarea proprietarului cheii și forțarea lui să descuie ușa.

Iar lista de exemple poate continua. Cu toate acestea ne lovim deseori în jocuri de situații în care nu reușim să găsim rezolvarea potrivită și nici nu avem posobilitatea să facem altceva până când ne vine inspirația. Asta în ciuda libertății de mișcare pe care ne-o promit producătorii la fiecare lansare. Acest lucru se datorează unor considerente economice. Orice element inserat într-un joc presupune niște costuri (mai mari sau mai mici). Și atunci se ridică următoarea problemă: Ce rost are să investești resurse (financiare, umane, etc) într-o soluție alternativă a cărei probabilitate de a fi utilizată de către jucător este foarte mică. Din punct de vedere al game design-ului aceste alternative sunt obligatorii. Din punctul de vedere al contabilului, însă, se poate renunța la ele. Și cum în ultima perioadă contabilul are un cuvânt mai greu de spus decât game designer-ul atunci este evident de ce ne mai blocăm din când în când în câte un joc.

Alte două aspecte, legate oarecum de acesta al soluțiilor multiple, sunt libertatea de mișcare și solutiile logice. Despre acestea vom discuta în numerele viitoare ale revistei.

Aștept în continuare întrebările voastre pe adresa de e-mail a redacției:

office@dark-times.com

Tot la această adresă aștept răspunsurile la întrebarea din această săptămână:

La ce gen de joc doriți să lucrați?

Jocul - Zeul Mileniului III

După cum v-am promis în numărul trecut am să continui seria de articole dedicate game design-ului. Astăzi vom vorbi despre importanța păstrării logicii și realității în rezolvarea situațiilor de joc.

Jocurile PC sunt un mediu virtual, fantastic în care de cele mai multe ori limitele realității sunt împinse la maxim și chiar depășite. Din această cauză mulți designeri sunt tentați să exagereze în crearea obstacolelor pe care jucătorul va trebui să le depășească. Rezultatul acestor exagerări constă în apariția unor situații ridicole sau imposibile care nu servesc deloc scopului jocului. Una dintre aceste posibile rezultate îl constituie puzzle-urile nerealiste. Un exemplu elocvent în acest caz este jocul Indiana Jones and The Infernal Machine. Dacă l-ați jucat atunci vă amintiți cum trebuia pe alocuri să cărați blocuri imense de piatră pentru a-i permite lui Indie să treacă mai departe. Ei bine unele dintre aceste blocuri erau de doua ori mai înalte decât Indiana Jones (un cub cu latura de vreo 3 metri). Ținând cont de densitatea stâncii ne dăm repede seama că acel bloc ar trebui să cântărească aproximativ 20 de tone. Și atunci cum poate Indiana Jones să-l tragă cu mâinile goale? Chiar dacă este un personaj incredibil, totuși nu este Superman. Dacă în schimb s-ar fi folosit un sistem de pârghii pentru a muta blocul respectiv atunci situația ar fi fost mai aproape de realitate.

Un alt exemplu de situație ridicolă și nepotrivită îl constituie Might and Magic VIII. În finalul jocului descopeream o navă extraterestră de pe care culegeam arme laser și combinezoane. După ce ai jucat tot timpul într-un univers fantasy să faci un salt atât de brutal la cel SF este o decizie absolut neinspirată. Ca să nu mai amintim de faptul că armele respective erau extrem de slabe, tot săbiile și topoarele făcând legea. Sunt convins că designerii jocului ar fi putut veni cu un final mult mai interesant și mai potrivit, de ce l-au ales tocmai pe acesta nu pot să înțeleg. Tot în categoria exemplelor negative se încadrează și puzzle-urile cu rezolvări ilogice, neintuitive sau cele care necesită cunoștințe specifice. De exemplu un quest în care pentru a trece mai departe trebuie să știi numele ultimilor cinci primari din Bolintinul de Vale este unul foarte prost. Dacă ai locuit în Bolintin s-ar putea să deții această informație, dar nu poți avea pretenția ca un american sau un japonez să știe aceste lucruri. La fel sunt quest-urile ilogice, cele de gen combină dalta cu guma de mestecat pentru a obține dinamita. Exemplul meu este extrem dar sunt destul de multe jocuri în care se întâlnesc situații asemănătoare.

Jocurile adventure vă aduc aproape inevitabil în situația de a descifra o combinație de simboluri (fie pentru a deschide un seif, fie pentru a stopa vreun mecanism, etc). În foarte multe cazuri aceste combinații (cel mai adesea de patru) nu erau deloc intuitive și nici nu exista prin joc vreo informație care să te lămurească. Practic odată ajuns în situația respectivă jucătorul trebuia să încerce toate combinatiile posibile până o găsea pe cea potrivită. Acesta constituie un exemplu negativ ce trebuie evitat. În cazul introducerii unei astfel de situații asigurați-vă că jucătorul va avea de unde să culeagă informații despre algoritmul necesar descoperirii combinației potrivite.

Aștept în continuare întrebările voastre pe adresa de e-mail a redacției:

office@dark-times.com

Jocul - Zeul Mileniului III

Ați remarcat probabil cât de des am pomenit expresia game design document în articolele mele de până acum. Acest lucru se datorează faptului că acest document, care din păcate este ignorat cu desăvârșire de către cei mai mulți dintre voi, este piatra de temelie a oricărui proiect. Nu numai că fără un astfel de document, serios și bine conceput, nu sunteți luat în seamă de nici un eventual distribuitor, dar veți vedea pe parcurs cât de anevoios se desfășoară producția în lipsa sa.

Toate ca toate, dar ce este un game design document? Cum se realizează? Cine îl face? Ce conține acesta? Acestea sunt câteva întrebări, pe care probabil vi le-ați pus și la care voi încerca să vă răspund acum.

Ce este un game design document?

Un game design document este acel document ce conține toate detaliile legate de concepția, producția, managementul, marketingul și distribuția jocului. Și când am spus toate detaliile m-am referit la absolut toate, indiferent cât de nesemnificative par. Daca vrei să păstrezi 10 cutii cu jocuri pentru verișorii tăi atunci trece acest lucru în document.

Cine realizează acest document?

Firesc, game designerul. Acesta trebuie să aibă o relație foarte strânsă cu toate departamentele pentru a se putea documenta și a înjgheba acest document. Nu vă speriați. Acest lucru nu înseamnă că această persoană trebuie să fie un fel de guru care să le știe pe toate. Este suficient să aibă o idee clară despre joc și strategia de producție și marketing. Detaliile specifice, de genul, cui se va distribui jocul, ce tip de engine se va folosi, ce resurse tehnice sunt necesare pentru programarea jocului se pot obține de la cei ce lucrează în departamentele specifice (adică marketing, resurse umane, programare, etc...).

Ce conține acest document?

După cum am spus deja el trebuie să conțină absolut tot ce este legat de acel joc, indiferent cât de nesmnificativ pare, indiferent dacă se va folosi sau nu până la urmă. Ca o idee generală (voi reveni asupra acestui aspect cu altă ocazie) trebuie să conțină următoarele capitole: ***Introducere*** (o scurtă descriere a proiectului cu elemente cheie), ***Scopul și Clientul țintă*** (cui se adresează și de ce), Descrierea proiectului (cu toate amănuntele), ***Planificarea procesului de producție*** (termenele de execuție), ***Strategia de popularizare*** (cum faci jocul cunoscut), ***Strategia de vânzare*** (cum vinzi jocul), ***Echipa*** (descrierea fiecărui om cu responsabilitățile și atribuțiile fiecăruia), ***Costurile de producție***, ***Prognoza financiară*** (estimarea câștigurilor - preferabil pe 2-3 ani).

Acest document nu este un element fix, el evoluează pe parcurs deoarece vor fi adăugate mereu elemente și idei noi în măsura în care acestea apar. Un alt element important, ce trebuie să fie conținut de către un design document sunt imaginile. Nu contează dacă sunt schițe, printuri publicitare, artwork-uri, capturi de engine sau imagini din joc. În momentul citirii lui de către distribuitor aceste imagini vor valora cât o mie de cuvinte și pot face diferența între o ofertă și un refuz.

Aștept în continuare întrebările voastre pe adresa de e-mail a redacției:

office@dark-times.com

Ne vedem din nou săptămâna viitoare cu alte informații.

Jocul - Zeul Mileniului III

În urmă cu câteva numere v-am întrebat dacă v-ar place să lucrați într-un studio ce produce jocuri PC și ce funcție ați dori să ocupați acolo. Am primit foarte multe răspunsuri la această întrebare, însă am fost dezamăgit de ceea ce am citit. Toți fără nici o excepție vor să fie programatori sau modelatori. Trăiască plebea!!!

Văd că, deși această revistă se adresează unei generații tinere și care vine din urmă, aceasta nu a reușit să scape de metehnele comuniste. Toți, fără nici o excepție, preferă să primească ordine și să le execute fără să încerce să se implice mai mult sau să-și asume niște responsabilități.

Nici unul nu dorește să fie game designer, nici unul nu dorește să fie Lead Programmer sau Lead Artist. Asta ca să nu mai amintesc de funcții precum Marketing Manager sau PR Manager care se pare nici nu există într-un astfel de studio. Și mai stăm să ne mirăm că producția de jocuri este un domeniu economic inexistent în România, asta în ciuda faptului că poate aduce nenumărate satisfacții, atât profesionale cât și materiale.

Știu că este mult mai ușor să stai la calculator și să scrii linii de cod sau să generezi un model în Studio Max, dar nu pot să cred că asta este tot ceea ce puteți face, că aici se oprește creativitatea tineretului românesc.

Chiar și pe meseriile amintite, cele de programator și grafician 3D, cei mai mulți sunt convinși că se pot descurca în orice situație. Că pot fi un fel de Meșterul Știe-TOT. Ei uită că IT este un domeniu prea vast pentru a putea să cunoști toate detaliile. N-am primit nici un răspuns în care cineva să-mi fi spus că el vrea să fie programator de AI sau să modeleze monștri. Nu, toți suntem niște generaliști, le știm pe toate și putem să facem de toate. Ei bine, am vești proaste. Astfel de oameni sunt extrem de puțini, iar cei care există valorează cât greutatea lor în aur. Ceilalți nu au altă opțiune decât să se specializeze pe o ramură anume a domeniului ales și să încerce să învețe cât mai mult, punând totodată în practică ceea ce a citit sau aflat. Numai așa vom putea crea programatori AI, level designeri, modelatori 3D, modelatori faciali sau texturieri capabili să ducă la bun sfârșit un proiect PC.

Haideți, este timpul să lăsăm în urmă toate lucrurile proaste care le-am învățat în cei 50 de ani de comunism. Este timpul să ne descătușam, este timpul să avem curajul de a face ceea ce vrem, de a ne exprima creativitatea, dar și de a ne asuma mai multe responsabilități.

Aștept în continuare întrebările voastre pe adresa de e-mail a redacției:

office@dark-times.com

Întrebarea săptămânii

Presupunem că sunteți implicați în crearea unui joc pentru PC.

În ce univers ați amplasa acțiunea acestuia și de ce?

Aștept răspunsurile și argumentele voastre pe adresa:
office@dark-times.com

Jocul - Zeul Mileniului III

Una din problemele majore cu care se confruntă studiourile românești de producție este păstrarea personalului și a motivației acestora. În Occident oamenii care lucrează în industria de jocuri video sunt considerați voluntari. Acest lucru se datorează faptului că oricare dintre ei ar putea câștiga de 2, 3 sau chiar 10 ori mai mult dacă ar lucra în alte domenii.

Acest voluntariat este și mai evident în țara noastră unde cam toți cei care se apucă de un proiect sunt tineri ambițioși și visători, ce se vor lovi curând de problemele realității.

Este evident pentru oricine că un om are nevoie de o motivație pentru orice acțiune a sa. Mănânci pentru că îți este foame, agăți fete în bar pentru că te stresează hormonii. La fel lucrezi la un joc pentru că ai o motivație. În lipsa unui stimulent financiar românii implicați în producția de jocuri trebuie să fie motivați în alt mod.

Cu toate că societatea modernă este caracterizată de un pragmatism feroce (banul dictează), sunt destul de mulți oameni care renunță la posibilitatea de a câștiga mulți bani pentru a lua parte la anumite proiecte. Acești oameni sunt motivați de alte lucruri precum satisfacția unei realizări, aprecierea acordată muncii lor, recunoașterea meritelor, etc.

Cu astfel de oameni trebuie puse bazele unui studio de producție. Faptul că ați reușit să strângeți un nucleu de astfel de persoane nu înseamnă că ați rezolvat problema. Deoarece odată cooptați acești oameni trebuie păstrați cu orice preț, iar aceata este sarcina cea mai dificilă.

Pentru ca acestor oameni să le fie greu să părăsească proiectul trebuie să faceți în așa fel încât ei să simtă că bucățica aceea la care lucrează este a lui, să se simtă parte integrantă a proiectului. După cum bine știm cu toții nimănui nu-i place să lucreze pentru altul. În schimb dacă lucrezi pentru tine se simte deja o schimbare majoră de atitudine.

Studiourile occidentale cu experiență în producția de jocuri folosesc două metode principale pentru a induce acest sentiment angajaților: brainstorming și czar chart. Nu cred că este nevoie să mai explic ce este un brainstorming și ce efecte are el.

Czar Chart este o modalitate de organizare internă a resurselor umane pe orizontală, în locul celei verticale, clasice. Această organizare are avantajul de a genera un sentiment de importanta în rândul angajaților în ciuda faptului că posibilitatea de avanasare este practic nulă. Mai multe detalii data viitoare.

Aștept în continuare întrebările voastre pe adresa de e-mail a redacției:

office@dark-times.com