Debreceni Egyetem
Informatikai Kar

A PhysX alkalmazási lehetőségei

Témavezető:
Dr. Tornai Róbert
Egyetemi adjunktus

Készítette:
Ormos Béla
Programtervező matematikus

Debrecen
2010

Tartalomjegyzék

Bevezetés

A diplomamunkámban szeretném bemutatni a PhysX technológiát, hogy mire akarták használni, és hogy mire használják manapság.

PhysX történelem

Az AGEIA cég 2006 közepén kereskedelmi forgalomba hozta a PhysX nevű chippel ellátott hardver kártyáit. Ezek a kártyák arra voltak hivatottak, hogy a számítógépes fizikai számításokat meggyorsítsák. A kártya elsődleges célpontja ekkor még a számítógépes játékipar volt, hogy a számítógépes játékokat minden addiginál élethűbb fizikai modellel lássák el. A technológia alapja egy PPU-nak (PhysX Processing Unit) nevezett feldolgozó egység volt, mely a többmagos processzorokhoz hasonló párhuzamos adatfeldolgozásra volt képes. Tervezésének és felépítésének köszönhetően rendkívül gyors fizikai számításokra volt képes, melyekkel a CPU-k ma sem képesek felvenni a versenyt. A társkártya nem tudott nagy népszerűségre szert tenni, viszonylag magas ára és szükségességének megválaszolatlan kérdése miatt. Bár a technológiát egyre több számítógépes játékszoftver támogatta, az AGEIA anyagi helyzete nem volt jó. Az AGEIA-t 2008 elején felvásárolta az NVIDIA, így a PhysX technológia a GeForce videokártyák részévé vált.

Témaválasztás

A PhysX az NVIDIA segítségével nagy teret nyert magának. Az Egyesített Shader Architektúra segítségével a videokártyák képesek a fizikai számítások elvégzésére is, így nem meglepő gondolat a két hardverelem egyesítése. Az NVIDIA ezen kívül elkészítette a CUDA nevű ingyenes szoftverét is, mellyel könnyedén lehet olyan programokat készíteni, amelyek a videokártyákban rejlő hardveres erőforrásokat fizikai számításokra használják fel, és ezzel nem csak a számítógépes játékok fejlesztéséhez használható a PhysX, hanem professzionális programokban is alkalmazhatóvá vált. A technológia így továbbra is rendelkezik hardveres gyorsítással, akár egy átlag felhasználó számítógépén is. Ez rendkívüli jelentőségű tény, hiszen jelenleg ez az egyetlen ilyen tulajdonsággal rendelkező fizikai eszközrendszer. A fizikai szimulációkban nagy segítséget jelenthet, ha a számításokat lényegesen gyorsabban és részletesebben lehet elvégezni, így az elméleti kutatások eredményeit jobb vizsgálati módszerekkel lehet ellenőrizni, és pontosítani, javítani mielőtt azok ténylegesen a gyakorlati

alkalmazásokba kerülnének. A szórakoztató iparban is jelentős előrelépésre ad lehetőséget, hiszen a játékokban lévő fizikai számítások gyorsabb, és életszerűbb játékokat eredményezhetnek, melyeket adott esetben professzionális szimulációkra is lehet használni. Például egy igen részletes és pontos fizikai motorral rendelkező autós játék, a megfelelő átalakítások után, melyek sokkal egyszerűbbek, mint egy teljesen új szoftver elkészítése, alkalmassá válik az autóvezetés oktatására, akár személyautó kisbusz, busz, vagy teherautó esetében is. A szimulátoron való gyakorlás után a kezdők nagyobb tapasztalattal, és magabiztossággal tudnának egy valódi jármű volánja mögé ülni. A szimuláció semmiképpen sem helyettesítheti a valódi vezetés élményét, és gyakorlatát, ahogyan ez Móricz Krisztián cikkéből [1] is kiderül. Ugyanez igaz a légi és vízi közlekedésre is. Az űrkutatásban is jelentős haszna lehet a PhysX-nek. A technológia segítségével elméletben gyorsabban, és biztonságosabban kipróbálhatóak bizonyos űrtechnológiák, mint például új űrállomások építése, működése, vagy új űrjárművek elméleti tesztjei, melyek lényegesen olcsóbbak, mint a valóságban kipróbálni őket. Az égitestek viselkedése is jobban és egyszerűbben modellezhetővé válik. Segítségével hamarabb fedezhetőek fel, érthetőek meg vagy modellezhetőek le jelenségek, vagy az univerzum és a galaxisok működése. Célom ezen alkalmazási lehetőségek megvizsgálása, körüljárása, hiszen a közeljövőben vélhetőleg egyre több helyen tűnnek fel ezek a lehetőségek. Megvizsgálom, és nyomon követem a szórakoztató iparban betöltött szerepét, hogy milyen hatásokkal vannak ezen megoldások a szoftverekre, és hogy ennek következtében azok alkalmassá válnak-e tudományos célú kísérletekre, vagy professzionális felhasználásra. Megvizsgálom továbbá azt is, hogy a technológia alkalmas-e új felfedezésekre, mint például, ha egy csillag vagy galaxis mozgását megfigyeljük, ezen adatokat egy modellező szoftvernek megadva, az nyújthat-e új információkat azon csillag vagy galaxis környezetéről, anyagaikról, esetleg a galaxis felépítéséről, felfedezhetőek-e eddig ismeretlen égitestek. Bár ez utóbbi kérdés eléggé költői, hiszen ma már bizonyítottan léteznek olyan égitestek, amelyeket kizárólag fizikai számolásokkal lehet kimutatni (például a fekete lyukak, melyek elég nagy számban fordulnak elő a világegyetemben). Joachim Herrman könyvéből [2] sok hasznos tanácsot meríthetünk ezen megfigyelésekhez, és a megfigyelésekből származó adatok segítségével olyan szimulációt készíthetünk, amelyből galaxisunk jövőjéről sok érdekes és hasznos ismeretet szerezhetünk, vagy akár távoli bolygókat is felfedezhetünk.

A PhysX és a szórakoztató ipar

A bevezetőben említett témakörök sorra vétele előtt, mindenképp szót kell ejteni egy alapvető dologról.

Mi is az a PhysX?

Első lépésként tisztázzuk, hogy mit is jelent a PhysX tulajdonképpen. A PhysX egy erős fizikai motor, mely valós idejű fizikai számításokat tesz lehetővé, az erre felkészített szoftverek segítségével, és a megfelelő hardverek által. Az eszközrendszer elsősorban az erősen párhuzamosított processzor rendszerekre lett kifejlesztve. Ez nem azt jelenti, hogy bármilyen több magos processzor, vagy több processzoros rendszeren hardvergyorsítással működik egy ilyen szoftver, hanem speciálisan a GeForce 8000+ videokártyákkal megtámogatott rendszerek képesek hardveresen támogatni a számításokat. Sajnos amennyiben egy számítógépben nincs ilyen videokártya, a tulajdonosa le kell, hogy mondjon a PhysX nyújtotta lehetőségekről.

A PhysX-ről kitalálói azt állítják, hogy a következő „nagy dolog” a játékvilágban. Az egész rendszer ahhoz nyújt megbízható támogatást, hogy a szoftverünk az általa modellezett világban, a különböző objektumok fizikai tulajdonságait, interakcióit, mozgásait gyorsabban és pontosabban számítsa ki. Bár a gyorsaság nagyobb szerepet játszik pillanatnyilag. A gyorsaság azért rendkívül fontos, mert egy viszonylag egyszerű térrész modellezése esetén is előfordulhat, hogy a modellezendő objektumok száma több ezer. Gondoljuk csak arra, hogy ha kiöntünk egy pohár vizet, akkor mi történik. Egy ilyen egyszerű, ám mégis rendkívül számításigényes jelenet lemodellezése igen bonyolult feladat, és igen sokáig tarthat a számítás. Vagy akár, ha egy hatalmas épületet kell lebontani, akkor gyakran robbantást alkalmaznak. Ilyen esetben is rendkívül hasznos lehet egy ilyen program, amely segítségével könnyebben meghatározható, hogy hova kell a tölteteket helyezni, és a törmelék merre fog repülni. A törmelékben lévő objektumok száma, pedig elérheti a milliós nagyságrendet is. Ezek az elemek, pedig igen gyakran fordulnak elő a számítógépes játékokban. Egy játékban, pedig igen fontos dolog, hogy a tárgyak ne úgy mozogjanak, mintha dróton húznák őket, hanem ténylegesen úgy mozogjanak, törjenek, essenek, ahogyan az a való életben is történne, vagy legalábbis, ahogy azt az egyszerű felhasználó elvárná tőlük. Ezeket az elvárásokat, pedig a PhysX igyekszik a legmesszemenőbben támogatni, és a szükséges számításokat elvégezni.

Így azok kezeléséhez nem szükséges saját magunknak megírni a kódokat, hanem csak meg kell adni a PhysX számára, hogy azok milyen fizikai tulajdonságokkal rendelkeznek, és a többit megoldja az eszközrendszer.

A játékokra gyakorolt hatás

Pár évvel ezelőtt, 2006 áprilisában a játékipar éves nagy konferenciáján, a GDC-n (Game Developers Conference), egy teljesen új technológiáról kezdtek vitatkozni a fejlesztők, ahogy ezt Halász Bertalan beszámolójában [3] is olvashatjuk. Ez az új technológia a PhysX volt. Az Ageia ekkor kezdte a köztudatba vinni a legújabb termékét, mely abban a pillanatban még csak a fejlesztők számára volt elérhető, de hónapokon belül el akarták kezdeni az előre összeszerelt számítógépek és társkártyák forgalmazását. A technológiához szép reményeket fűztek, és a játékfejlesztők fantáziája el is kezdett szárnyalni a lehetőség hallatán. Az ötletek egyébként nem voltak nagyok, leginkább olyan apróságnak tűnő dolgok, mint például az, hogy minden tárgy mozgatható legyen egy játékban, amely elég kicsi ahhoz, hogy a játék szereplői fel tudják emelni. 2006-ban a rendelkezésre álló eszközök fejlettsége még nem volt elég ilyen „nagy" volumenű megoldásokhoz. A PhysX pedig egy igen kellemes lehetőség volt ezen lehetőségek alkalmazásához. Mivel előre elkészített hardveresen támogatott szoftverről beszélünk, így a játékok készítőinek nem kellett ezeket az opciókat leprogramozniuk, csupán a meglévő eljárásokat alkalmazniuk. A fejlesztéshez szükséges idő, és a fejlesztés költségei így jelentősen csökkenhettek, ami vonzóbbá teheti az egyes programok finanszírozását a kiadók számára. A minőség is garantált, hiszen a szoftver képességeit előzetesen le lehet ellenőrizni a különböző techdemok segítségével, amelyek pont a szoftver tudását hivatottak szemléltetni. Így meg lehet győződni a program tudásáról, és nem „zsákbamacska" a megvásárlása. Nem csoda, hogy már akkor körülbelül 63 fejlesztő- és kiadóvállalat rendelkezett a PhysX felhasználási jogaival.

A 2006 előtti játékprogramok igen nagy részében az épületek, és bizonyos tárgyak nem csak hogy nem voltak mozdíthatóak, de még sérülést sem lehetett nekik okozni. A legnagyobb kárt, amit el tudtak szenvedni, legfeljebb egy lyukat ábrázoló textúra jelenített meg, és az sem végtelen mennyiségben. Egy bizonyos mennyiség után, az első sérülésnyomok elkezdtek eltűnni. Egy egyszerű játékosnak ezek az apróságok legtöbbször fel sem tűnnek, de egy törekvő fejlesztőnek ez nem elfogadható. A PhysX lehetőséget biztosít arra, hogy a játéktéren megtalálható bármely tárgyat apró darabokra törhessünk szét. Ez persze nem mindig

kívánatos, hiszen grafikus megjelenítésben is követni kell a darabokat, de ezáltal olyan tárgyak is rombolhatóvá válnak, amelyek eddig csak mereven álltak a helyükön, és azon kívül, hogy kikerültük őket, mást nem lehetett velük kezdeni. A PhysX megjelenése előtt is szerepet kapott a tárgyak rombolása. Igen gyakoriak voltak a széttörhető dobozok, ládák, és robbanó hordókat is használtak már 1993-ban. Az azonban nem volt megoldható, hogy a széttört ládák darabjaival bármit is tenni tudjuk, így például az 1998-ban megjelenő Half-Life című játékban sokféle logikai feladványt lehetett megoldani a ládák tologatásával, széttörésével, de miután darabokra vertük őket, a darabjaikkal nem csak interakció nem volt lehetséges, de egy rövid idő után szépen elhalványodtak, és eltűntek a játéktérről. Ez persze igen gyakran megtörtént más tárgyakkal is, leginkább egy-egy robbanás utáni törmelékkel, így a szétrepülő tárgyakkal a játékosnak nem kellett törődnie, mert nem léphetett velük interakcióba, és így sérülést sem okozhattak. A legyőzött ellenfelek is gyakran keltek át a semmibe. A PhysX használatával ez azonban megváltozott. A különböző díszítő elemek, mint például az asztalon lévő vázák, és egyéb tárgyak, már nem fixen az asztalhoz rögzített elemek.

1. ábra. Képlopás a Gears of War 2 című játékból

Külön objektumként definiálva őket, azokat is könnyen át lehet helyezni máshova, vagy akár szét is lehet törni őket. Így egy FPS-ben (First Person Shooter) egy megterített asztalt át lehet pakolni máshova, akár elemenként, vagy akár az egészet megfogva, óvatosan egyensúlyozva át lehet tenni az asztalt. Ha valami leesik, akkor nem tűnik el, hanem vissza lehet tenni a helyére, feltéve, ha nem volt törékeny a tárgy. Ha összetört, akkor sem válik az enyészeté, hanem ott marad a padlón, de akár a darabjait vissza lehet helyezni az asztalra. Persze ilyen apró részleteket nem feltétlenül helyeznek el egy játékban, de a lehetőség adott. Sajnos az erőforrások továbbra se végtelenek, de lényegesen nagyobb mennyiségű tárgy helyezhető el, és mozgatható a játéktéren. Ismerős probléma lehet még az is, hogy előfordult olyan eset, amikor két láda egymás tetejére volt helyezve. Mindkettő mozgatható, nem is ez a probléma. A gond az, hogy ha az alsót megtoljuk, akkor a felső mozdulatlan marad. A PhysX erre is ügyel. Ha egy tárgyat elmozdítunk, akkor a rajta lévő dolgok vele együtt mozognak, mint az előző asztalos példánál is. Természetesen nem fixen, hanem a valós életből ismert erőhatások itt is jelen vannak, és a mozgásenergiának csak egy része kerül át a felül lévő tárgyakba, az érintkező felületektől függően. Így a magasabb és vékonyabb tárgyak eldőlhetnek, a kerekek elgurulnak.

Egy másik fontos eleme a játékoknak a rombolás. A játékokkal játszó emberek természetéből fakadóan sok játék a harcra épít, így ezekben a szoftverekben előkerülnek a fegyverek. Az első rombolható környezetet szimuláló játék a 2001-es Red Fraction volt. A GEO-MOD motornak köszönhetően nem sok tárgy volt, amelyet ne lehetett volna apró darabokra törni. Nagyon nagy újításnak számított ez akkoriban, és sokféle új rejtvény tudtak alkotni általa. Például a továbbjutás érdekében egy fal megfelelő pontján lyukat kellett fúrni, és így juthattunk el a következő pályaszakaszra. A törmelék sajnos itt is csak látványelemként szolgált, de az már elérhető volt, hogy ha egy ajtót nem sikerült kinyitni, akkor egyszerűn lyukat lehetett mellette robbantani a falba, és nem gátolta további haladásunkat az ajtó. Más játékokban is szerepeltek hasonló megoldások, de azokban a lerombolandó falak előre szkripteltek voltak, azaz pontosan meg volt adva, hogy a meglőtt falrésznek hogyan is kell leomlania. A PhysX ezzel kapcsolatos legnagyobb újítása, hogy ha valamilyen épületet leromboltunk, akkor a GEO-MOD-dal ellentétben, a törmelék megmarad a játéktéren, kézbe lehet venni, de ha figyelmetlen a játékos, és túl közel áll egy felrobbanó oszlophoz, akkor a szétrepülő törmelék is megsebezheti, de akár a maga a ledőlő oszlop is agyonütheti. Ezt persze ki is lehet használni, és logikai feladványokat lehet köré építeni. Például egy nagyobb

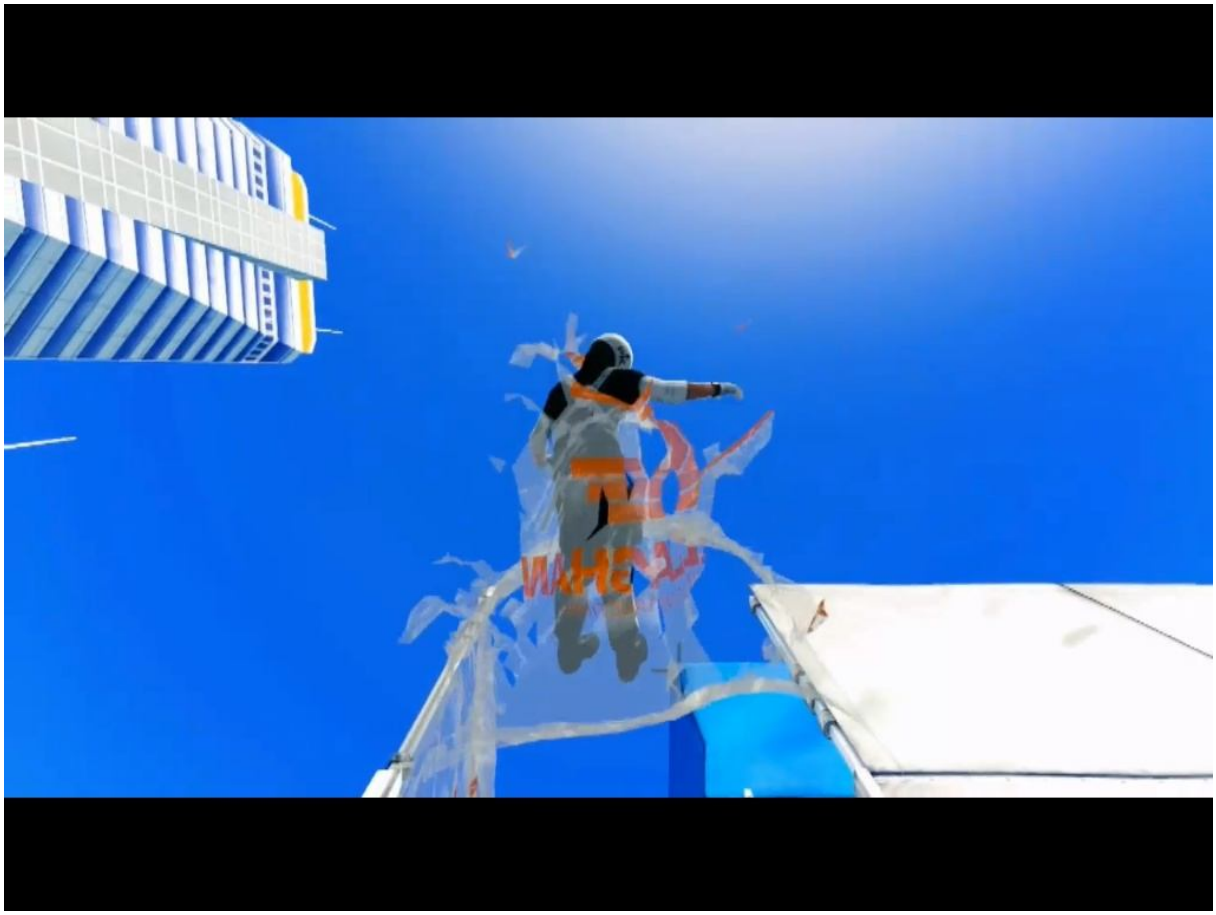
és erősebb ellenfelet csak úgy lehet legyőzni, ha egy épület maga alá temeti. Persze akadnak kihívói a PhysX-nek, mint például a GEO-MOD 2.0. A GEO-MOD 2.0 szintén rendelkezik az előbb említett effektekkel, így a Red Faction: Guerrilla című játék pályatervezőinek már nem lehetett teljesen szabadjára engedni a képzeletüket. Fontos volt betartani néhány alapvető szabályt, hogy az épületek ne dőljenek azonnal össze a játék kezdetén. A játékmenet hatására ez később persze lehetséges. A legfőbb különbség a két fizika motor között az, hogy míg a GEO-MOD 2.0-t egy szoftverfejlesztő cég készíti, addig a PhysX-et egy hardver gyártó cég fejleszti, így a PhysX hardveres támogatása lényegesen erősebb. Természetesen mindkét fizikai motor lehetővé teszi, hogy meghatározzuk bennük azt, hogy mi legyen rombolható és mi nem. A rombolhatóság egy másik hatása a játékmenetre, hogy a korábban teljesen biztos fedezékekből akár halálos csapda is lehet, vagy akár olyan is lehet az egész, mintha az adott fedezék ott sem lenne. A ládák eddig sem nyújtottak biztos fedezéket a játékokban, de ezentúl egy eddig teljesen biztonságosnak számító fabódé is csak takarást nyújt, de biztos fedezéket már nem. Eddig is voltak próbálkozások arra vonatkozóan, hogy a fegyverekkel át lehetett lőni egy gyengébb anyagot, de a hatás még közel sem volt valósághű. A Counter-Strike nevű igen népszerű játékban a faládákon könnyen át lehetett lőni, de erősebb fegyverrel, akár a falakon keresztül is áthatolhatott a lövedék. Egyetlen probléma, hogy a találatnak semmilyen nyoma nem maradt, így ha valakire a falon keresztül nyitottak tüzet, annak esélye se volt meglátni merről lőnek, csak a találat irányjelzéséből sejthette, de még az is igen gyenge pontosságú volt. A Project IGI című alkotás is alkalmazott hasonló megoldást, ám ott már a lövés nyoma meglátszott a falakon mindkét irányból. A megvalósítás már egész jó volt, viszont a sérülés még mindig csupán egy apró lyukat ábrázoló textúra formájában jelent meg, ami még mindig nem túl feltűnő, és nem is túl valósághű. A PhysX ezen a téren is fejlődött, hiszen a fa forgácsolódik, a betonfalról lehullik a vakolat, a téglák, pedig kimozdulnak a helyükről. Ezt nem csak az anyagok tulajdonságainak megadásával lehet szabályozni, hiszen más egy téglafal szétesése, és megint más egy betonfal leomlása, de egy deszkakerítés is egész más módon hullik darabjaira, ha erőhatás éri. Erre egy igen egyszerű megoldást találtak ki az NVIDIA fejlesztői. A téglafal megalkotásakor minden egyes téglát külön kell összerakni, és a két tégla között a habarcs erősségét külön kell megadni. Azaz, a téglákat alkotó csúcspontok nem választhatóak szét, vagy csak nagy erőhatások révén, míg a téglákat összetartó erő sokkal gyengébb, így egy robbantás elsősorban a téglákat választja szét egymástól, és nem összetöri őket, ahogy ez a második ábrán látható. Természetesen maguk a

2. ábra. Képlopás a NVIDIA Supersonic Sled PhysX Demo programból

téglák is eltörhetnek, morzsolódhatnak. Egy betonfal esetén más a helyzet, hiszen az egy tömör összefüggő test. A betonnál nincsen megszabva, hogy pontosan hol fog kettétörni, ha erőhatás éri. Nem teljesen véletlenszerű a dolog, hiszen pontos számadatokból számolja ki a rendszer, de pár centiméteres eltérés esetén, alakulhat teljesen másként a rombolódás. A valós élet alapján elvárt törmelék itt is szanaszét repül, de azok mennyisége, alakja, és nagysága függ a betonfalat ért hatásoktól. Egy deszkakerítés viszont harmadikféleképpen reagál az őt érő hatásokra. Maga a kerítés felépítése hasonlít a téglafaléhoz, de az anyaga lényegesen gyengébb, és másképp is törik. A széttörő részek mechanizmusa inkább hasonlít a betonéhoz, mert a rögzítések és a deszkák keménysége hasonló, de mégis inkább az illesztéseknél válik ketté, így a deszkák félbetörhetnek, vagy akár egész nagy darabok is arrébb repülhetnek, pont úgy, mint az a valóságban is várhatóan megtörténne. A tényleges forgácsolódásra még várni kell, de annak is inkább a számítógépek teljesítménye szab határt. Az anyagok tulajdonságainak megadásával, azaz, hogy mennyire kemény, mekkora elemekből épül fel, és azok mennyire tapadnak egymáshoz, már egész komplex épületeket, tárgyakat, járműveket lehet létrehozni, ezzel is valóságosabbá téve a játékokat. Például egy autó már nem csak egy

téglatesthez hasonlít, hanem akár az üléshuzat fajtája is modellezhető, amely szakad, és nem törik.

A szövetek anyagai is másképp modellezhetőek le. Itt is meg kell adni, hogy a szövetet milyen poligonháló alkotja és az anyag tulajdonságait. A szövet szépen hajlik, gyűrődik, képes minden olyan mozgásra, amire egy valódi ruhadarab. Erre egy sima textúra eddig is képes volt bizonyos mértékben. A textúrákat is lehetett hajtani, gyűrni, bár nem olyan szépen. Viszont míg a textúra a program résztvevői számára láthatatlan dolog, addig a PhysX által modellezett szövetanyag tapintható is. Van egy másik dolog is, amire képes még egy PhysX-es szövet, amelyre más modellek még nem, ez pedig nem más, mint a szakadás. Az őt érő erők hatására megnyúlhat, vagy akár szét is szakadhat. Ha a csatlakozó felület nem elég nagy,



3. ábra. Képlopás a Mirror's Edge című játékból

akkor akár a saját súlyától is kettészakadhat. A játékokban ez a fajta felhasználás még sok szerephez nem jutott. Egyik neves képviselője a Mirror's Edge, amelyben inkább még csak díszítő szerepet kapott. Kétségtelenül jobb a hangulata ennek a megoldásnak, de a játékmechanizmust még semennyire sem befolyásolja, hiszen a szövetek csupán díszítő elemként voltak jelen a játékban. Igaz ugyan, hogy voltak olyan jelenetek, amelyben akár a

karakterek is átrepültek a vékony szöveteken, és ettől azok élethűen szétszakadtak, de a továbbhaladás szempontjából semmilyen lényeges módosító hatásuk nem volt. Később talán megjelennek azok a ruhamodellek is, amelyeket a szereplők ténylegesen „hordanak”, azaz koszolódik, gyűrődik a szereplők ruhája, és a harcok során akár cafatokban is lóghat róluk a sérül ruhadarab. A szövetek ezenkívül, rejtvények készítésére is szolgálhatnak úgy, hogy a továbbhaladás irányát takarják el, vagy valamilyen kapcsolót, vagy egyéb tárgyat lehet velük letakarni, így nehezítve a keresést. Persze vigyázni kell ezzel a lehetőséggel, nehogy frusztráló hatása legyen, mert elég idegesítő lehet az is, ha a sokadik kapcsolót is egy darab függöny takarja. A lopakodásban is lehet haszna a szöveteknek. Vegyük a régmúlt eseményeit alapul, amikor a háborúk során a katonák sátortáborokban voltak elszállásolva, ez persze ma sem lehetetlen, de talán nem annyira jellemző, és a sátrak között kell valamilyen fontos személyhez eljutni, vagy egyéb fontos, vagy fontosnak vélt tárgyat megszerezni. A szövetek itt is takarnak, bár az árnyékok esetleg átlátszhatnak rajtuk, de a takaráson kívül más segítséget aligha tudnak nyújtani.

A PhysX egy másik hasznos, de legalábbis érdekes, funkciója a részecske rendszer. A részecske rendszerrel sok látványos effektet lehet elkészíteni, legyen akár a szálló porról, ködről, vagy akár vízről, folyadékokról. A köd elég régóta képezi fontos részét a játékoknak. Az is igaz, hogy a köd szerepe eddig inkább hardverkímélő volt, nem igazán a játékélmény javítására szolgált. Hangulati elemnek is kiváló választás. Sok esetben az elsődleges szerepe azonban az volt, hogy eltakarja a terepet, és így lecsökkentse a látótávolságot, miáltal a számítási igénye csökkent a játéknak. Ugyanígy a környezet kidolgozatlanságát is lehetet takargatni, mondhatni az egész „a ködbe veszett”. Persze akadnak olyan játékok is, ahol egy pálya szerves részét képezte a köd, mint például a Delta Force 2. A PhysX ezen a téren is igyekszik továbblépni. A köd, a füst, a por, sőt még sok folyadék is, eddig nem volt interaktív. Leginkább szépen kidolgozott textúrákkal igyekeztek a készítők azt az érzést kelteni a játékosban, hogy kapcsolatba került ezekkel az elemekkel. A PhysX részecske rendszere az előbb említett koncepcióval ellentétben lehetőséget nyújt az interakcióra. Ezentúl a sivatagban száguldozó autó utáni porfelhő nem egy ügyes textúra lesz, hanem valós időben megjelenített porfelhő. Sok videó igyekszik szemléltetni ennek az effektnek a látványosságát, de játéktechnikai szempontból is fontos előrelépést jelent ez a tényező. Például ha két vagy több jármű versenyzik valamilyen poros terepen, akkor az elől haladók által felvert por nagyban befolyásolja a hátrébb lévők látótávolságát, sőt, akár azt is, hogy látnak-e egyáltalán valamit a

terepből. Leírva talán nem tűnik olyan nagy változásnak, de elképzeljük magunkat egy versenyautó volánja mögött, és egy sivatagos terepen száguldozunk, ahol előttünk nem sokkal egy másik autó halad, amely a port felveri, akkor egész más az élmény, mintha mindent tisztán látnánk. A kanyarokat esetleg csak onnan vesszük észre, hogy az előttünk haladó féklámpája derengeni kezd a porban. Reméljük, hamarosan ilyen játékokkal is lehet majd játszani. A füstök is változnak, így ha egy játékos átszalad a füstön, az megkavarodik mögötte, és ha nem is sokáig, de egy ideig láthatóvá válik merről szaladt be. Nem feltétlenül szükséges egy test mozogjon is a részecskék között, hiszen azok maguktól is mozognak, így látványos jeleneteket lehet velük létrehozni. A folyadékok viselkedése is változik. A játékokban pár évvel ezelőtt, mintha versenyezni kezdtek volna a fejlesztők a víz megjelenítésével. A különböző programokban igyekeztek minél élethűbben, minél szebben ábrázolni a tavakat, folyókat, tengereket, pocsolyákat. A megjelenítés szépségén túl néhol egyéb feladatot is kaptak a folyadékok. A könnyű tárgyak felemelkedtek a víz felszínére, a nehezek lesüllyedtek, a mozgó testek lelassultak. A sodrás elvihette a tárgyakat, de nem

4. ábra. Képlopás az NVIDIA PhysX Particle Fluid Demo nevű programból

keletkezhettek öblök, és nem akadhattak fel egymásban a tárgyak, hanem egy jól meghatározott görbe mentén mentek végig. A felszínen hullámzó tárgyakat is ritkán lehetett látni. A Half-Life 2 egy említhető kivétel, ahol elég élethűek voltak a folyadékok, de még ott is elég sok valószerűtlen dolog történt. A folyadéknak nem igazán volt saját fizikai jellemzőjük, hanem inkább módosultak bennük a fizikai törvények. A tárgyak könnyebbek lettek, a játékos inkább csak lassabban haladt benne. A PhysX ezt igyekszik javítani, és itt már valóban vannak fizikai tulajdonságai a folyadékoknak, így akár a nagy erővel előre törő víz olyan tárgyakat is elmozdíthat, amelyeket egyébként nem tudna. A negyedik ábrán látható is, ahogyan elsodor a víz néhány ládát. Az öblökbe besodródó testek ott forgolódnak, míg újra ki nem szabadulnak, és ez nem előre beprogramozott ideig tart, hanem ameddig ténylegesen kellene, hogy tartson. A játékmenet tekintetében ezek a dolgok sem hoznának sok előrelépés önmagukban. Képzeljünk el azért egy olyan jelenetet, ahol egy gát átszakad, akár a játékos szakítja át, akár valamilyen egyéb esemény, és a kizúduló víz nagy sebességgel tör a játékos felé. Filmekben már láthattunk ehhez hasonló jelenetet ugyan, de játékokban még ilyen sosem fordult elő. A játékos így valós időben számolt vízáradat elől menekülne, aminek a legnagyobb előnye az lenne, hogy sosem történne meg a dolog kétszer ugyanúgy. Az önismétlés a játékok legnagyobb ellensége, így igen nagy hasznát vennénk a PhysX ezen tulajdonságának. A víz különböző tereptárgyakat mozdítana el, épületeket döntene romba, mindezt a játékos szeme láttára. Sőt, az sem lenne túl nagy gond, ha elsodorná a szereplőt a víz. Egy csak megjelenített áradás esetén ez a játék végét jelentené, hiszen mi történne az elsodort személlyel ezután? A fizikai tulajdonságokkal rendelkező víz esetén ezután még sok lehetőség adódnak. Szerencsésebb esetben nem verné egyből oda valamihez a játékost, hanem sodorná egy darabig, majd ha enyhült a sodrás újabb feladat elé állítaná a szereplőt. A még mindig erős sodrásban a felszínre kellene úszni, és ott folytatni a játékot. A különböző feltűnő akadályokat kikerülni, és találni egy olyan öblöt, ahonnan nem sodor azonnal ki a víz, és a partra evickélni. Sokféleképpen érhet véget így az adott jelenet, ezzel is új élmények és kihívások elé állítva az arra vállalkozókat. Merőben más lenne az egész esemény az eddig megszokottaknál ezzel is növelve az újrajátszhatóságot. Mindezt a részecske rendszernek köszönhetően. A rendszer lényege, hogy a program kisebb nagyobb gömbökkel, ún. részecskékkel, szimulálja az egész eseményt. A részecskék mérete változó lehet, és a tulajdonságaik is attól függnek, hogy épp vizet, port, vagy füstöt szeretnénk megjeleníteni. A méretek dinamikusan számolódnak, attól függően, hogy mekkora részét lehet az adott

elemnek egy egységként kezelni. Egy nagyobb tavat például pár száz, vagy esetleg ezer körüli részecskével lehet modellezni, de amint beleesik egy nagy tárgy, a részecskeszám azonnal több tízezerre nőhet, hiszen a csobbanás alakját, és mozgását már kisebb részecskékkel lehet csak modellezni. A sok, vagy épphogy kevés, gömb a megjelenítést nem zavarja, hiszen a programok eddig is külön kezelték a fizikai „dimenziót" a látható dimenziótól. A kisebb nagyobb gömböcskék, így egy a megjelenítendő elemnek megfelelő alakot kaphatnak, és a víz valóban víznek látszik, a füst füstnek, és így tovább. Természetesen lehetőség van a részecskék megjelenítésére is, ha szeretnénk leellenőrizni, hogy valóban minden úgy zajlik, ahogy azt mi akarjuk.

A PhysX-től nagyon sokat vártak, majdhogynem a teljes játékipar megreformálását. Ez természetesen elmaradt, bár sok várakozás teljesült, de mégsem történt akkora változás, mint azt gondolták. A környezet valóban rombolható lett, mint azt némely játékok esetében tapasztalhatjuk, de nem vált általánossá, hiszen elvétve akad néhány játék, amely ezzel a lehetőséggel kezd is valamit. Sőt ezek közül nem is mind használja magát a PhysX könyvtárat. A környezet rombolhatóságával más lehetőséget is reméltek a játék kritikusok, mégpedig a rombolás ellentétét az építést. Ugyanis ha lerombolok egy hegyoldalt, a leomlott törmelékből építhetnék a közeli folyóra egy gátat, amely ettől felduzzadna, vagy más irányba kezdene el folyni, ezáltal megváltoztatva a játékteret. Ilyen, vagy ehhez hasonló még egyetlen játékban sem szerepelt, pedig érdekes lenne. Kis túlzással azért mégis lehet hasonlónak nevezni bizonyos helyzeteket, de azok inkább a hangulatot adó, vagy az új térképet magyarázó átvezető videók részeit képezik. A másik helyzet például a korábban említett asztalok, és egyéb tárgyak esete. A való életben sok helyen sok apróbb nagyobb tárgy található az asztalokon, amelyek ugyebár mind-mind önálló életet élnek, de legalábbis külön-külön elmozdíthatóak. A játékokra ez is már átterjedt, azaz a tárgyak a legtöbb esetben elmozdíthatóak, felvehetőek, de valamiért a játékok még ezen a téren sem teljesen valósághűek. Az ok meglehetősen egyszerű, konkrétan az, hogy sok helyen alig találni kis tárgyakat. Az asztalok meglehetősen kopárak, szekrénnyel pedig, szinte csak elvétve találkozik virtuális karakterünk. Ennek kettős oka van. Egyrészt ez általában fel sem tűnik az embernek, a játékélményre figyel, és az ennyire apró dolgok nem feltűnőek, így nem szükséges ezeket mind létrehozni, mely időt és pénzt is spórol a fejlesztőknek. Másrészt a sok apró tárgy, aminek a játékmenetre szinte nulla a hatása, eléggé erőforrás igényes lenne. Sok apró tárggyal bár szebb lenne egy szoba, de hiába a szépség, hogyha percenként 1 képkockát

láthatnánk belőle. A rombolás és átépítés esetén ugyanez a probléma. Az elképzelés ugyan érdekes, és élvezetes is lehetne, de ez mind-mind számolásokat igényelne, amitől a játék sebessége csökkenne, így tönkretéve a szórakozást. Az ütközésdetektálás már szinte hibátlan, és gyorsan meg is történik a játékokban, és ez egy jó tulajdonsága a PhysX-nek. Viszont ha sok tárgyat rakunk egy helyre, az sok számolást igényel. A hardverfejlődés üteme nem lassult, az elmúlt közel 4 évben sem, de még a mai hardverek mellett is elég könnyű egy-egy ilyen jelenetet annyira túlzsúfolni, hogy nemhogy irányítani, még nézni se lehessen a lassúsága miatt. Ráadásul a videokártyáknak nem csak a PhysX számításait kell elvégezniük, hanem a megjelenítendő képet is nekik kell szolgáltatni, így ez egy másik probléma. Ezt ellensúlyozandó arra is van lehetőség, hogy 2 vagy akár 3 videokártyát is az alaplapra helyezzünk, így míg az egyik, amelyre a monitorunkat kötöttük, a kép előállításával foglalkozik, addig a másik teljes mértékben a PhysX számolásait végezheti. Három kártya esetén, pedig a harmadik szükség szerint besegít a másik kettőnek. Ez mind szépen hangzik, de egy átlagos játékos ritkán engedhet meg magának egy ilyen számítógépet. A magas ár mellett az is elriasztó hatással van, hogy bár a többlet teljesítmény szépen mutatkozik, és a méréseknél tényleg elképesztő a különbség, de egy-egy játékban a változásokat nem igazán

5. ábra. A processzorok és SLI rendszereke teljesítmény/ár összehasonlítása
Forrás: http://www.nvidia.co.uk/object/sli_technology_games_uk.html

vehetjük észre. Vannak olyan játékok, amelyek erre építenek, és tényleg szembetűnő a két lehetőség közötti eltérés, de két olcsóbb kártya nem feltétlenül nyújt olyan teljesítményt, mint egy drágább, és az alaplapnak is támogatnia kell a több kártya lehetőségét, így olyan is előfordul, hogy a gyengébb változat kerül többe. Persze némi keresgéléssel megtalálható az a konfiguráció, ami mind árban, mind teljesítményben megfelelő, és a játékokat is találunk hozzá bőven. A mérések is igazolják, hogy két, vagy több videokártya jobb játékélményt nyújt, mint egy jobb processzor, és nem utolsó sorban olcsóbb is, de ezek a játékok nem mind vannak PhysX támogatással is ellátva, így ebből a szempontból a kérdés másként fest.

Mezei Károly 2006. júliusi cikkéből [4] is kiderül, hogy akkoriban is már sejteni lehetett, hogy a PhysX kártya nem marad önálló hardver elem, hanem valamelyik grafikus kártyára felkerül. Akkoriban még nem lehetett eldönteni, hogy mindent elsöprő siker, vagy hatalmas bukás lesz a vállalkozás. Az elmúlt 4 évben inkább bukásnak tűnt a dolog, mivel sokáig alig lehetett hallani a PhysX-es támogatásról. A megjelenő játékok nem nagy csinnadrattával jelentették be a támogatást, hanem mellékesen megjegyezték, hogy ilyet is lehet vele. Bukásnak mégsem volt nevezhető, hiszen a fejlődés folyamatos volt, a PhysX sosem tűnt el a játékokból, csak nem kapott olyan hangsúlyos szerepet, mint amilyenre a szakma gondolt annak bejelentésekor. A fizikai motorok nem mindegyike lett hardveresen támogatott, bár most 4 év után úgy tűnik, a hardveres fizikai gyorsítás mégis teret hódít magának. A Havok fizikai motor is „gazdára” talált az AMD grafikus kártyáinak köszönhetően, bár ebben az esetben a támogatás nem feltétlenül kizárólagos, ugyanis a Havok motor az OpenCL szabvány segítségével használja ki a grafikus kártyák képességeit, és itt inkább két cég közötti együttműködésről van szó. A PhysX annyi előnyt élvez ezzel szemben, hogy az NVIDIA saját maga fejleszti a hardvert és a hozzá tartozó drivereket, így jobb a támogatottsága, de mégis szükséges hozzá egy konkrét hardver elem. Érdekes verseny alakulhat ki a két motor között, melynek várhatóan a felhasználók lesznek a győztesei, hiszen egyik cég sem akar majd lemaradni a másiktól és folyamatosan várhatóak a jobbnál jobb megoldások. Természetesen még versenyben vannak a teljesen szoftveres megoldások is, mint például a GEO-MOD engine, amelyeknek az OpenCL szabvány segítségével szintén lehetőségük van a hardveres kapacitások kihasználására, ám ezeket már nem támogatja egyik nagy videokártya gyártó cég sem. Kérdés mit hoz a jövő, és mire lesznek képesek a hardverek és szoftverek. Azt sem lehet teljesen kizárni, hogy vagy az NVIDIA vagy az AMD, de még akár az Intel is beszállhat a

versenybe, hiszen nekik is vannak grafikus processzoraik, nem karol fel még egy fizikai enginet, ezzel is próbálva növelni a versenytársaival szembeni előnyét.
A PhysX képességeinek jobb megismeréséhez érdemes szemügyre venni pár játékot, amelyek támogatják, hogy lássuk, mit tudnak a programok vele, és mire képesek nélküle.

Infernal

Az Infernal volt az első játékok egyike, amelyen már tényleg érezhető volt a PhysX fizikai engine jótékony hatása. Előtte is jelentek meg már játékok, amelyek támogatták a PhysX könyvtárat, de sajnos a kártya kiforratlansága miatt, azokon még az is előfordult, hogy a hardveres támogatás lassított is rajtuk. Az Infernal hosszú csönd után újra felhívta a figyelmet, az akkor még AGEIA-hoz tartozó megoldásra. Maga a játék körülbelül 1 éves csúszás után jelent meg, az első bejelentet megjelenítési időhöz képest, melyért a PhysX támogatás beépítése okolható. Az első kritikák szerint megérte a várakozást, hiszen valóban sokat dob a játékélményen a fizikai gyorsító kártya. Bár maga a játék a szaksajtótól nem kapott túl jó értékeléseket, mint Téglás Gábor írásából [5] is kiderül, a játék az elsők között szerepelt azok között, melyért érdemes volt fontolóra venni egy PPU kártya beszerzését.
A játék motorja jól volt optimalizálva, így első ránézésre a PhysX kártyával rendelkező és a PhysX kártyával nem rendelkező gépeken nem sok különbséget lehetett felfedezni játék közben. Alaposabb összehasonlítás után azonban észrevehető a különbség. Az első és legfontosabb, amit meg kell említenünk az a processzor terheltsége. A hardveres gyorsítás lényege abban rejlik, hogy a processzort mentesíti bizonyos számolások alól, amitől a szoftver gyorsabban, esetleg részletesebb tudja rendelkezésünkre bocsátani az eredményeket. Ezt a kitételt az Infernal teljesíti is. Bár ennek a mérése nem egyszerű dolog, de a megfelelő mérő programok segítségével elvégezhető. Lehetőség van természetesen a kártya kiszerelésére is, és úgy kipróbálni a játékot, melynek hatására előfordulhatnak a számításigényes jeleneteknél lassulások. Ezt szintén elég nehéz észrevenni, de a már említett mérő programok ismét csak a segítségünkre lehetnek. Ettől függetlenül elég nehezen bizonyítható ez az állítás, ugyanis a processzor általában így is teljesen ki van használva. A gyorsító kártya teljesítményére általános esetben ritkábban van szükség, és olyankor is gyakran elég rövid időre, hiszen a robbanások, szerkezetek összeomlása nem túl gyakori, és nem is hosszadalmas folyamat. A „minden tárgyat apró darabokra robbantunk” elgondolás is megvalósult az Infernalban. Az üveg, fa és műanyag tárgyak könnyedén széttörhetőek, de ezen kívül a főszereplő különleges

6. ábra. PhysX támogatással fölül, nélküle alul. A különbség nem szembetűnő.
Forrás:
http://images.eurogamer.net/assets/articles//a/7/6/5/1/5/ss_preview_infernal.jpg.jpg?slideshow=true

képességeinek hála ajtókat, és szekrényeket is könnyű szerrel apríthatunk miszlikre. Sőt! Vélhetőleg a PhysX szemléltetése érdekében, előfordulnak a játékban különböző állványok, vagy építmények, amelyeket többnyire ellenség is tartózkodik, és szintén a főhős különleges képességeinek köszönhetően ezeket a szerkezeteket is romba dönthetjük. Igen látványos, ahogy ezek az építmények szépen megroggyannak és összeroskadnak. Persze ezek között az

események között nem látszik nagy különbség hardveres támogatással és nélküle, ahogyan a hatodik ábrán is látszik. Az összhatás talán mégis elég meggyőzőre sikerült.

Medal of Honor: Airborne

A következő program, amelyet megvizsgálok, a Medal of Honor: Airborne. Ez a játék nem tűnik ki elsőségével, mint az Infernal, és talán nem is annyira látványos benne a fizika motor, de mivel volt szerencsém ezt a játékot kipróbálni, így az összehasonlítás is jobb.

Rombolhatóság terén a MoH:A alig nyújt valamit. A készítők nem azzal próbáltak kiemelkedni a tömegből, hogy a háború által lerombolt területeket még jobban szét lehet rombolni, hanem inkább a testek és fegyverek mozgását igyekeztek minél élethűbben kidolgozni. Bár talán a játék fejlesztőinek ezzel csak annyi feladata volt, hogy a PhysX technológiát minél jobban beépítsék a programba, hiszen ezek kezelése teljes egészében a fizikai motor dolga. Minden küldetés úgy kezdődik, hogy egy repülőből kiugrik az ejtőernyős osztag, akik között ott van a játékos által megszemélyesített katona is. Az ejtőernyő kinyílása után megkapjuk az irányítást. A PhysX már itt megmutatja tudását, hiszen az ernyő fizikáját, mozgását is igyekeztek minél valósághűbben elkészíteni a fejlesztők. A szél belekaphat, és ha a játékos nem figyel, elsodorhatja veszélyes területre, vagy csak távol a ledobási zónától. Vigyázni kell a manőverezéssel is, mert ahogy a valóságban, itt is egy túl éles fordulás hatására összecsuklik az ernyő és szabadeséssel ér földet karakterünk. Ennél a jelenetnél sajnos megint csak nem lehet teljesen eldönteni, hogy mit ér a PhysX, hiszen hardveres támogatás nélkül is ugyanebben az élményben van része a játékosnak. A földet érésnél is igyekszik kihasználni az API támogatását a program, így szintén különböző módokon foghatunk talajt. Talán szembetűnőbb a különbség a robbanások között. Hardveres támogatás nélkül is szépen szóródik a törmelék, de hardveres támogatással a robbanásnak sokkal erőteljesebb hatása van. Míg támogatás nélkül olyan érzése van embernek, hogy amit lát az egy szépen megrajzolt néhány textúrából álló effekt, addig ez hardveres támogatás mellett sokkal elementárisabb hatást vált ki. A porból álló fátyol nem csak simán elhalványul, hanem inkább szétoszlik. Talán ez az egyetlen eleme a játéknak, ahol a különbséget észre lehet venni. A holtestek és fegyverek mozgása szintén jól működik, nem lóg bele semmi a környezetbe. Bár két testnél előfordulhatnak, hogy nem működik tökéletesen az ütközésdetektálás, erre viszonylag ritkán látni példát a nagy terek miatt. A fegyverek nincsenek odaragasztva ellenfeleinkhez, így egy robbanás hatására az eszköz külön irányba indulhat el, mint korábbi

gazdája. A gránátok pattogása kicsit túlzottnak tűnik a több tárgyhoz képes. A fizikai motor legszembetűnőbb hatása az, amiről egyébként bemutató videó is készült, hogy szemléltesse milyen nagy tudású is a rendszer, hogy a holtesteken lévő ruha beakadhat például egy drótkerítésbe, így nem egyszer előfordul, hogy egy elesett katona valamilyen szúrós helyről lóg az uniformisával felakadva. Az ejtőernyők sajnos nem tudja ugyanezt a jelenséget produkálni, így az simán áthalad bármilyen sűrű dolgon. Talán a játékmenet megkönnyítése érdekében, hiszen elég zavaró lehet, ha az ernyőnél fogva lógunk valahonnan, és semmit se tudunk kezdeni a helyzettel. A probléma itt is ugyanaz, mint az Infernal esetében. A látvány szép, a fizikai motor jól kidolgozott, de a hardveres támogatás jótékony hatása itt is inkább a processzor tehermentesítésében jelentkezik elsősorban, amely így ismét körülményesen bizonyítható. A másodpercenkénti néhány képkocka, amellyel többet tud, szintén alig látszik. Bár ez dicsérheti a jól optimalizált fizikai motort is, amely segítségével gyorsan számolhatóak az események, megint kérdésessé válik, hogy szükség van-e a hardveres támogatásra.

Gears of War

A Gears Of War eredetileg Xbox 360 exkluzív program volt, azaz semmilyen más eszközön nem lehetett használni, csak a Microsoft konzolján. Ez pár évvel később megváltozott, és elkészült a játék PC-re is. Mivel a keretrendszer elérhető az Xbox 360 konzolokon is, így a PhysX API támogatása átkerül a PC-s verzióba is.

A helyzet kicsit hasonló a MoH:A helyzetéhez. A környezet felépítése adott, a fegyverekkel nem sok mindent lehet bennük módosítani. Bár az is igaz, hogy általában nem is elég erősek ehhez a fegyvereink, hogy valóban kárt tehessünk bennük. De itt már akadnak bizonyos tárgyak, amelyeket össze lehet törni. Erre legalkalmasabb a fegyverünk láncfűrésze. Néha még a továbbjutáshoz is ketté kell fűrészelni egy-egy szekrényt, ágyat. Bár ezek az elemek sem a fizikai látványt szolgálják. Előfordulnak ajtók, amelyeket be lehet rúgni, vagy olyanok is, amelyeket csak robbantással lehet kidönteni a helyéről, vagy ha elsőre az nem sikerülne, akkor ezeknél az ajtóknál is lehetőségünk van utat rúgni magunknak. A holttestek fizikája van talán a legjobban kidolgozva. A GoW-ban már nem járkál utánunk valaki lesben, hogy az elesett ellenfelek testét összeszedje, így könnyítve meg a processzor dolgát. Ha valaki elhullik, akkor bizony ott is marad. A robbanások hatására látványosan repülnek, sőt akár még szét is szakadhatnak. Nincsenek teljesen összeragasztva, ha valahol túl nagy erőhatás éri őket a test két része két irányba indulhat el. Nem csúsznak bele semmilyen tereptárgyba, és itt is

előfordulhat, hogy fennakadnak valamiben. A testek egymást is figyelembe veszik, így olyan sem fordul elő, hogy két holttest egymásba csúszik. A még élő karakterek is beleakadhatnak elesett bajtársaikba, vagy épp ellenfeleikbe, és a harc hevében tovább sodorják őket a földön. Sajnos még nem tökéletes ez a fizikai modellezés, és a gyakran látott probléma itt is megfigyelhető. A egyre tökéletesedő ragdoll fizikának persze örülünk, de azért az nem szép látvány, amikor egy ökölcsapástól métereket repül az ütésbe belehaló ellenfél, ellenben ha túléli még a lábát sem teszi hátrébb, hogy az elvesztett egyensúlyát próbálja visszanyerni. Így bár dicséretes, hogy a holttestek reagálnak a környezetükből érkező hatásokra, amelyet kevés más program mondhat el magáról, azért az zavaró, hogy ha embereink beleakadnak egy testbe, akkor szintén métereken keresztül húzhatják maguk után anélkül, hogy a járásukon, a mozgásukon, vagy a haladásukon egy picit is meglátszana az a plusz 50-100 kiló, amit a lábukba akadva vonszolnak maguk után. A megoldást talán a testek megnövelt tömege jelentené, bár valószínűleg nem ezzel van a probléma, és például a berúgott ajtók esetében is jól látható a tömeg szerepe, ahogy egy vaskosabb ajtó ránézésre is nagyobb súlyúnak tűnik és másként lendül, mint hasonló módon járt drótkerítéses társa. Sokkal inkább az, hogy a mozgásra képes testek, személyek nem rendelkeznek erőkorláttal, vagy valamilyenfajta terheltségi mutatóval, hogy ha nagyobb súlyt kell mozgatni, akkor lassabban tegyék ezt, és ne csak annyival legyen megoldva a dolog, hogy ha valami elég könnyű, azt el lehet mozdítani, gyakran játszi könnyedséggel, ha pedig valami túl nehéz, akkor azt semmilyen módon se lehet megrezdíteni sem. Ez még nem túl reális, de remélhetőleg a PhysX fejlesztői dolgoznak ezen a problémán, és a közeljövőben erre is lesz valamilyen megoldás, mert a hardverek kapacitása már elegendő ehhez, hiszen ez nem sok plusz számítást igényel egy egyszerűbb esetben. A fegyverünkön helyet kapó láncfűrész segítségével ellenfeleinket ketté is tudjuk fűrészelni, mely szintén a fizikai modellt dicséri, mert az egyben mozgó szilárd testből gond nélkül kettő lesz. Bár itt is megfigyelhető a szkripteltség, mert nem vághatjuk akárhogyan ketté ellenfelünket, de a lehetőség adott, és semmilyen fennakadást nem okoz a program futásában. A gránátok és fegyverek mozgása az elvárható módon történik, bár ezekbe a tárgyakba már különös módon nem lehet beleakadni. Ha valahol földet érnek, akkor ott is maradnak, míg valaki fel nem veszi őket.

Unreal Tournament 3

Az Unreal Tournament 3 esete az előzőekben tárgyal 3 másik játékéhoz nem hasonlítható. Bár ugyanaz a fejlesztőgárda készítette, mint a GoW-ot, de az UT3 egy egészen más utat járt be az NVIDIA jóvoltából. Az UT3 ugyanis az a játék, amelynek segítségével az PhysX fejlesztői rengeteg bemutató videót készítettek, a fizikai motor tudását szemléltetve. Nagyon sok látványos videó található az Interneten, amelyek mind-mind azt hívatottak szemléltetni, hogy milyen képességei, és lehetőségei vannak az API-nak. A játékhoz magához, egy kiegészítő is letölthető, amely tele van olyan pályákkal, ahol a PhysX-nek köszönhetően eddig ritkán látott élményben lehet része a játékosoknak.

A környezet rombolhatósága megvalósításra került. Bár még itt sem formálható teljesen át a környezet, de szép számmal akadnak olyan helyek, ahol a falakba belelőve szép nagy lyukakat lehet készíteni. Sok helyen lehet lerövidíteni az utat ezzel a megoldással, és az ellenfeleinknek némi meglepetést is okozhatunk azzal, hogy hirtelen eltűnik a fal, és mögüle egy másik játékos nyit tüzet. Nem csak a falak rombolhatóak, hanem vannak bőven egyéb tárgyak is, amelyeket kis darabokra zúzhatunk szét fegyvereinkkel. Hidakat, szobrokat dönthetünk le, ezzel is kellemetlen perceket okozva a másik csapatnak. Bár a rombolhatóság elég sok helyen szerepel a pályákon, azért még sem tökéletes a megvalósítás. Több olyan hely van, ahol a leggyengébb fegyverrel is komoly károkat lehet okozni, ami azért egy kicsit túlzás. Ezenkívül, olyan helyek is akadnak, ahol a legerősebb fegyvereinkkel se tudunk még egy kis karcolást sem ejteni a környezetben. Vélhetően erre a legegyszerűbb elgondolás adja a magyarázatot. A mai hardverekkel is elég nehéz lenne megvalósítani a pályák teljes rombolhatóságát, hiszen igen nagy számolási kapacitás szükséges ahhoz, hogy egy ekkora pályát teljes egészében valós időben alakíthassunk át. Ráadásul ez történhet a pálya több részén is teljesen egy időben, hiszen egyszerre akár több mint tíz játékos is lehet a pályán, így ha mind a tízen rombolni kezdenek, az igen nagy számításigénnyel járna, amelyet még a hardveres támogatással se lehetne tökéletesen szimulálni. Az osztott erőforrások se oldanák meg teljesen a problémát, hiszen elképzelhető, hogy nem online játszik a játékos, így gyakorlatilag mindenki mást a mesterséges intelligencia irányít, amely plusz számítási igényekkel jár. A szövetek is helyet kaptak a játékban. Van, ahol csak díszítő elemként szolgálnak, de olyan hely is akad, ahol erősebb szövetek vannak kifeszítve, és azokon maga a játékos is járhat. Persze ha kilyukad a szövet, akkor a súlyosabb tárgyak alatt ez is leszakad, ahogy azt a valóságban is elvárnánk. Ám olyan helyek is akadnak, ahol ugródeszkaszerűen

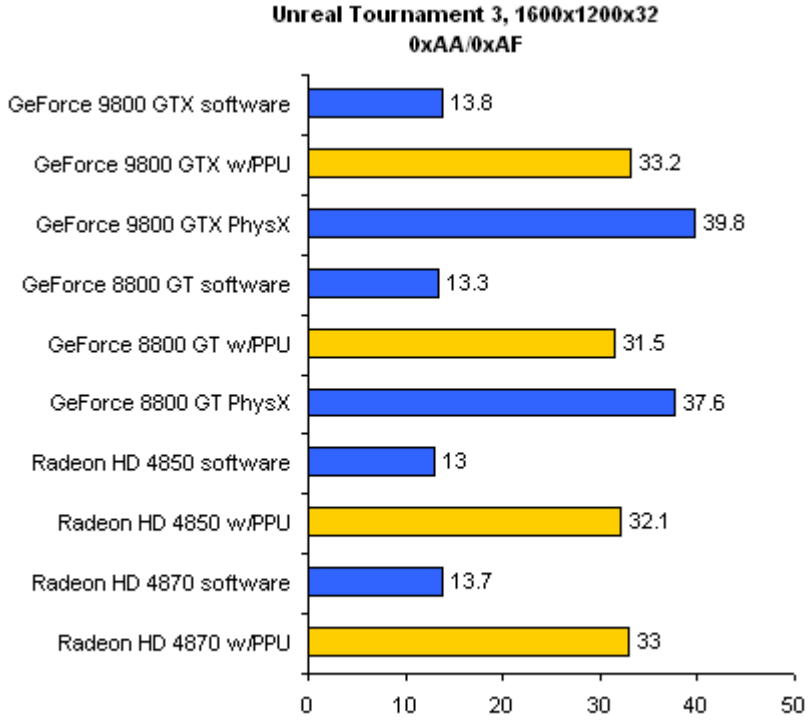
lehet rajtuk ugrálni, pattogni. Ezek a szövetek se bírják a végtelenségig, de lényegesebben nehezebb őket átszakítani. Fegyvereinknek azonban ez sem okoz nagy gondot, bár furcsa módon, a falakkal ellentétben a leggyengébb fegyver nem biztos, hogy elsőre átüti a szövetet. A részecske effektek is ki lettek használva. Található a kiegészítőben olyan pálya is, ahol jégeső hullik az égből, és a jégdarabok a felgyülemlett vizet fröcskölik. Bár ez azon kívül, hogy nagyon látványos, sok hatással nincs a játékmenetre. A lövedékek által keltett erőhatások is szerepet kaptak. A lövések ereje képes odébb sodorni az apróbb, könnyű tárgyakat, amely szintén elég látványos, és némi hatással is van a játékra, hiszen a lassabban mozgó, így tovább mozgásban lévő törmeléket, könnyebben észre lehet venni, és hamarabb

7. ábra. Egy tornádó tombol a pályán, és lebont mindent
Forrás: http://sev1512.files.wordpress.com/2008/08/ut3_torn_large_2.jpg

meg lehet találni a lövést leadó játékost. Itt is találhatóak ajtók, melyek között a könnyebbeket teljesen szét lehet rombolni, a nagyobb, erősebb, tömörebb ajtókat, pedig a Gears of War-hoz hasonlóan ki lehet lökni. Ez konkrétabban annyit jelent, hogy ha beléjük lövünk a különböző erejű fegyvereinkkel, a különböző nagyságú lökések hatására, különböző mértékű lendületet kap az ajtó, és így gyorsabban, vagy lassabban átlendül egy másik szögbe.

A PhysX az UT3 kiegészítőjében csillogtatja meg igazán tudását, kihasználtak minden lehetőséget, amit csak be akartak mutatni a fejlesztők, és egy egészen másfajta játékélményt mutattak be, amelyhez hasonló még nem sok akad a világon. Persze vannak olyan játékok, amelyek hasonló élményt próbálnak kínálni, de egyik sem ennyire összetett, mint a PhysX. Van, amelyik többet tud, egy-két dologban, viszont csak az UT3-ban láthattuk mindezt egyszerre.

Még egy fontos eredmény tudhat magáénak ez a kiegészítés. Egy akkor már elég régi vitára volt sorsdöntő hatással. Sokan már temetni kezdték az AGEIA kezdeményezését, mely ekkor már az NVIDIA tulajdonában állt. Mégsem volt túl sok bizonyíték arra, hogy valóban megéri, csak ezért egy PhysX kompatibilis hardvert beszerezni számítógépünkbe. Több tesztet is készítettek a fizikai motor hatékonyságáról, és bár voltak nagyobb eltérések, az azonos tudású konkurens videokártyák között nem volt olyan jelentős erőbéli fölény, hogy egyértelművé tegye az egész vállalkozás létjogosultságát. A régebbi AGEIA PPU-k is jól szerepeltek a teszten, de az újabb NVIDIA-s videokártyák még rajtuk is felülkerekedtek. A szoftveresen számolt fizikai modellek, mint a konkurens RADEON kártyákkal ellátott konfigurációk, majdnem 2.5-szer rosszabbul teljesítettek, mind hardveres társaik. Ezek után újra a figyelem középpontjába került a hardveres fizikai gyorsítás, már nem csak az NVIDIA fejlesztőinél, hanem a konkurens cégek is elkezdtek hasonló megoldásokon gondolkozni.



8. ábra. Az Unreal Tournament 3 PhysX kiegészítőjével készült mérések, CPU, PPU és GPU számítás segítségével
Forrás: http://firingsquad.com/hardware/physx_performance_update/default.asp

A PhysX és a flash

Mielőtt továbbhaladnék a játékok területéről a PhysX egyéb felhasználási lehetőségei felé, egy új, nemrég felmerült lehetőséget is szeretnék megvizsgálni. Az NVIDIA igyekszik tovább terjeszkedni, és olyan új, eddig nem használ felhasználási módot találni, amire korábban még nem volt példa. Az ötlet igen egyszerű. A nemrégiben lezajló gazdasági válság hatására az egyébként is népszerű ingyenes, online használható flash alkalmazások az eddigieknél is nagyobb népszerűségre tettek szert, hiszen csak egy Internet kapcsolattal rendelkező nem túl erős személyi számítógép szükséges. Sőt napjainkban már nem csak a számítógépek képesek ezen alkalmazások futtatására, bár ez a vizsgált alanyunk szempontjából mellékes, inkább a népszerűségének növekedésében játszik némi szerepet. Miért ne próbálnánk kihasználni videokártyánk erejét ezen a területen is? Az Adobe már korábban is próbálkozott egy hasonló megoldással, ugyanis a flash player-nek van egy olyan kapcsolója, amellyel ha futtatjuk az alkalmazásunkat, akkor a lejátszó a megjelenítendő képet a grafikus kártyával igyekszik még szebbé varázsolni. A lehetőség sajnos kihasználatlan maradt, ugyanis az flash alkalmazások nagy részének nem volt rá szüksége, mert vagy nem volt mire használniuk, vagy a kép feljavítása látható különbséget nem eredményezett, és még az alkalmazás is lassabb lett tőle. Éppen ezért az Adobe is csak akkor ajánlotta ezt a lehetőséget, ha a flash lejátszón keresztül néztünk nagyobb felbontású videókat. A flash igen nagy elterjedése miatt az NVIDIA is érdeklődni kezdett a lehetőség után. Ennek eredményeként nemrégiben megszületett a 10.1-es flash player, amelyet a legújabb NVIDIA videokártya illesztő programmal párosítva a nagyfelbontású videókat már valóban nem a processzorunk játssza le, hanem ténylegesen a videokártya jeleníti meg hardveresen. Ez azt jelenti, hogy az Interneten nagy népszerűségnek örvendő videó megosztó oldalakon még nagyobb felbontású videókat lehet megtekinteni olyan minőségben, mintha DVD-ről néznénk. Legalábbis hasonló ígéretek vannak ezzel kapcsolatban, de mivel még béta fázisban tart a projekt, előfordulnak kisebb nagyobb problémák a dologgal.

De ha már lehet flash videókat úgy nézni, hogy közben a videokártya hardverét használjuk, miért ne próbálnánk meg kihasználni, a grafikus processzor nyújtotta egyéb lehetőségeket is, mint például a hardveres fizikai gyorsítás. Az Interneten számos olyan flash játék található, amely teljes egészében fizikai fejtörőkre épít. Nagyon sok ötlet van, hogyan is lehet a különböző fizikai törvényekkel logikai feladványokat készíteni, de olyan is akad, amely az ügyességünket igényli. A flash player viszont nem a gyorsaságáról nevezetes, sőt a

sebességével kapcsolatban a fejlesztőknek az első gondolata az, hogy lassú. A lassúság következménye, hogy nem célszerű túl sok fizikai számítást igénylő játékot kreálni, mert az teljesen élvezhetetlenné válik a játékos számára, így az egész munka kárba vész. De ha a fejlesztő kerüli is az ilyen helyzetet, akkor is előfordul számtalan játékban, hogy a játék természetes mechanizmusa folytán egyre több és több test kerül be a játéktérre. Bár a helyzetet nagyban egyszerűsíti, hogy többnyire csak két dimenzióban dolgozunk, ettől független már 100 test mozgatásánál is előfordulhat, hogy az egész játék úgy belassul, hogy másodpercek is eltelnek, mire két képet kirajzol a program. Így nem egy elvetendő ötlet bevetni a hardveres gyorsítást erre a problémára és minden eddiginél több test és forma kerülhet a játéktérre, még érdekesebbé és összetettebbé téve a játékot.

Az ötlet nem tűnik túl bonyolultnak, így vegyük is sorra mik erre a jelenlegi lehetőségek. Az NVIDIA és az Adobe álláspontja a témával kapcsolatban nem ismert, sehol nem találni arra vonatkozó információt, hogy erre lesz-e valaha is lehetősége a flash fejlesztőknek. A szakma természetesen már érdeklődik a lehetőség iránt, de jelenleg erre nincs még mód. Legalábbis nem áll rendelkezésünkre olyan lehetőség, hogy magában az Action Script API-ban találjunk olyan osztályt, amelynek segítségével a flash playeren keresztül elérhetővé váljon az operációs rendszer bármilyen szolgáltatása. Természetesen arra van lehetőség, hogy a flash player külső eljárásokat hívjon meg. Erre az `External` csomag nyújt lehetőséget. Előzetes példányosítás nélkül, csak beimportáljuk az osztályt a saját osztályunkban, és az `ExternalInterface.call()` utasítással már hívhatjuk is a külső eljárást, feltéve, ha a flash player erre lehetőséget nyújt. Ha egy böngésző programon belül fut a lejátszó, akkor erre a legtöbb esetben van mód, így ez önmagában nem probléma. A meghívott eljárás is lehet bármilyen külső könyvtár része, csak az adott böngészőnek ismernie kell azt. A legnagyobb probléma, hogy a böngészők egyike sem képes használni a PhysX API-t, hiszen semmi szükségük nincsen rá, így magunknak kell egy olyan plug-in-t készíteni, amely képes erre. Egy plug-in elkészítése nem a legegyszerűbb dolog, és nagyon fontos az is, hogy minden böngészőhöz külön kell megírni, sőt olyan is előfordulhat, hogy két különböző verzióhoz is különböző plug-in szükséges, amely jelentősen bonyolítja a dolgot. Ha elég elszántak vagyunk, akkor megoldható ez a probléma, viszont elég bonyolult ahhoz, hogy bárki csak úgy belevágjon, így elég sok szaktudást igényel. Új böngésző készítésére is van lehetőség. Ha sikerül megírni a plug-in-okat, és azok esetleg már képesek közvetlenül használni a PhysX API-t, akkor innentől kezdve szabadnak látszik az út. De van egy másik jelentősnek látszó

probléma, nevezetesen a következő. A flash alkalmazás meghívja a lejátszó szolgáltatását, ami kísérletet tesz egy külső eljárás meghívására, amit a böngészőbe beépített plug-inünk észlel és reagál a kérésre, úgy, hogy gyakorlatilag semmi mást nem tesz, csak meghívja a PhysX megfelelő eljárását, amely lefut, válaszol a plug-in-nek, ami tovább adja az eredményt a lejátszónak, ami a kapott eredményt visszaadja az alkalmazásunknak. Ez viszont nem kevés idő, hiszen minden rész a másikra vár. Az időt az is megnyújtja, hogy a flash alkalmazásunk a processzoron fut, és a PhysX meghívása után a számolandó adatokat át kell vinni a videokártya GPU-jához, ami szintén nem a leggyorsabb megoldás. Az OpenCL alkalmazásoknál is ismert probléma, hogy ha a GPU-t be akarjuk fogni valamilyen számításra, akkor célszerű ezt nagyobb adathalmaz esetén megtenni, különben az átviteli idők miatt nemhogy nem lesz gyorsabb a számítás, hanem még lassabb is lesz. Erre az átviteli időre jön rá a hívási lánc plusz ideje, és valljuk be, egy flash játékban azért nincs olyan tetemesen sok adat, hogy az ilyen módon megérje átvinni. Persze ha megfelelően nagy léptékben gondolkodunk, akkor talán átlépjük azt a határt, amelytől már megéri ezt a megoldást választani. Egy másik probléma az átvivendő adatokban rejlik. Ugyanis megoldható, hogy a flash alkalmazás csak számokat küldjön át, de ekkor egy külön PhysX alkalmazás szükséges, ami akkor fölöslegessé teszi az egész megoldást. Így a flash játékban külön le kell kezelni a PhysX-xel közlendő adatokat, és adattípusokat, és az egészet valamilyen módon össze is kell kapcsolni a flash megjelenítő rendszerével, hiszen azt már nem a videokártya végzi, bár ez még a legegyszerűbb ebben az esetben. Így a PhysX számára csak annyit kell közölni, hogy létrehoztuk a testet, és időnként lekérni tőle, hogy az adott testtel éppen mi történt, hol található, egyben van-e még, vagy hozzáér-e valamihez, nekiütközött-e valaminek, és ehhez hasonló dolgok. Egyszerűbb esetben az adatok megmaradnak mindkét API által használt típusokban, viszont az átvitt adatok mennyisége továbbra is szinte jelentéktelennek számít, így a legtöbb időt az átvitel jelenti, és gyakorlatilag, amit nyernénk a gyorsabb számolással, azt elveszítjük az átvitellel. A fentebb említett problémák mellett még fontos megemlíteni a kompatibilitásbeli eltéréseket is. Alapvetően nincs sok eltérés a különböző böngészőkbe készített flash lejátszók között, és ha a plug-in-eket is sikerült jól megírni, akkor az eltérések tovább csökkenthetőek. De fontos különbség még így is a sebességbeli eltérés. A különböző böngészők, különböző sebességgel képesek végrehajtani a rájuk bízott feladatot, sőt a különböző telepített egyéb kiegészítők is nagy befolyással vannak arra, hogy a böngésző mennyi idő alatt képes végrehajtani a

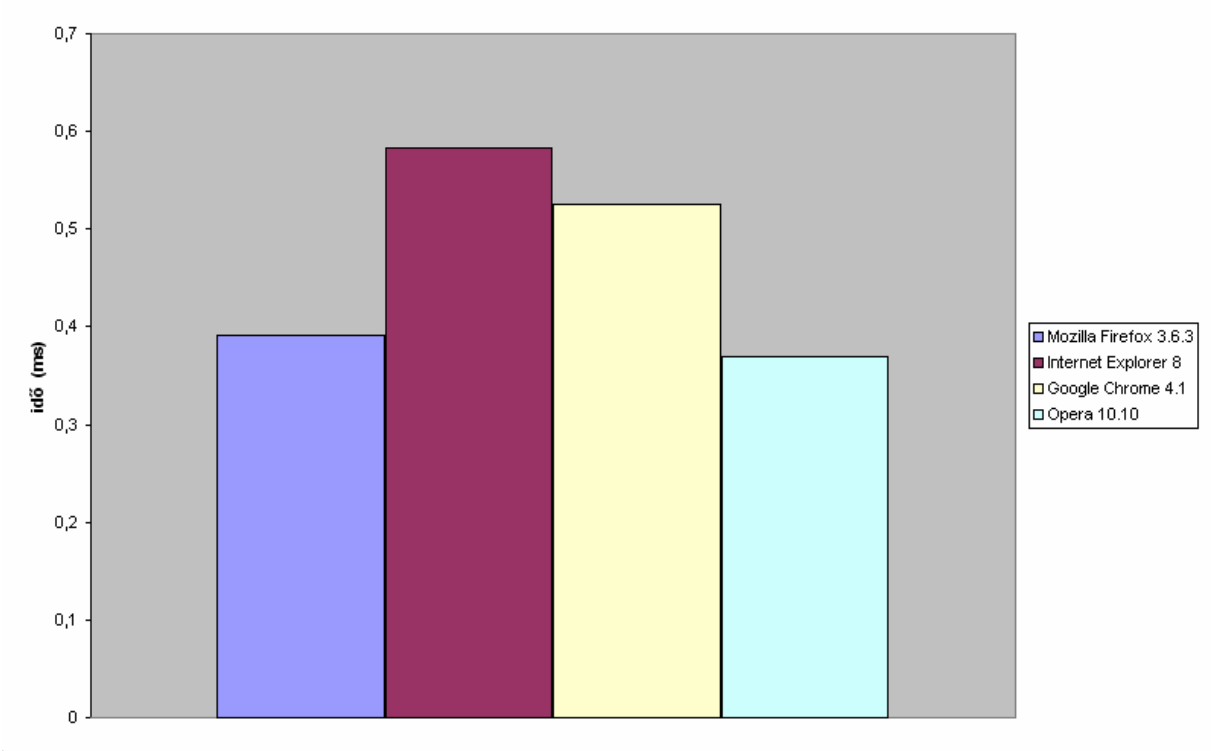
különböző tevékenységeket. Ezt szemléltetendő készítettem egy egyszerű tesztet, melyben lemértem, hogy a legnépszerűbb böngészők, milyen sebességgel képesek végrehajtani a kommunikációt, a flash alkalmazás, és a javascript motorok között. A mérés alatt egyszerű feladatot hajtattam végre a böngészőkkel. A végrehajtott kód a következő Action Script 3.0-s része a következő:

```
private function szamol():int {
    var timer:int = getTimer();
    var s:String = ExternalInterface.call('getit', 1);
    var timer2:int = getTimer();
    var time:int = timer2 - timer;
    return time;
}
```

A kód rendkívül egyszerű műveleteket hajt végre. Első lépésben a `getTimer()` eljáráshívás lekéri a lejátszó inicializálása óta eltelt időt milisecundumban és elmenti az int típusú `timer` változóban. Ezután történik a korábban már említett osztály segítségével a külső eljárás meghívása, amely neve a getit és paraméterként kapja az 1-et. A visszatérési értéket, pedig az `s` változó tárolja el. Majd ezután ismét lekérjük az eltelt időt, és a két időmennyiség különbségéből megkapjuk a híváshoz és a külső kód lefutásához szükséges időt. A külső kód szintén igen egyszerű elemekből áll, mégpedig a következő módon:

```
function getit(j){
    return j;
}
```

A külső eljárás megkapja a paramétert, és mindenféle számítás nélkül visszaadja az eredményt, hogy minél kisebb legyen a hatása a különböző javascript motoroknak a kapott eredményre. Ám már ennyiből is látszik, hogy milyen eltérések vannak a böngészők között. Bár maga az idő még így sem olyan rendkívül sok, de sok kicsit sokra megy, és itt még csak a böngészőig jutott a kérés, nincs se plug-in, se PhysX API hívás, amelyből kifolyólag az egész művelet a processzoron belül zajlott le, figyelembe véve a rendkívül kis adatmennyiséget is. A mérés során ezt a rövid kódot ismételtem meg 3000 alkalommal. A kísérlet alatt a processzor minden böngésző mellett azonos, 0%-os terhelést kapott, hogy ez se befolyásolja az eredményt. A grafikonról jól leolvasható, hogy a különböző böngészők válaszideje között az eltérés akár közel kétszeres is lehet. Ez, pedig a program hordozhatóságát befolyásolja,



9. ábra. A különböző böngészők körülbelüli válaszideje flash alkalmazásból történő külső eljáráshívásra

hiszen nemcsak az nem mindegy, hogy van-e GeForce videokártya a gépben, hanem az is, hogy mely böngésző alatt fut az alkalmazás. És jelen esetben, ha nincs a megfelelő hardver a gépben elhelyezve a szoftveres támogatást is a plug-in-nek kellene biztosítania, ami annak a méretét és a telepítésének a bonyolultságát nagyban megnöveli. A böngészők különbözőségét is meg lehet próbálni elkerülni, mégpedig egy külön alkalmazás készítésével. Ebben az esetben a plug-in-t és a böngészőt egy darab alkalmazás váltja fel, amely a kompatibilitást teljesen megoldja, hiszen ekkor, ha szükséges szoftveres támogatást is tud nyújtani a flash alkalmazás számára. A külön program viszont egy egészen másfajta problémát vet fel. Ha önálló programot futtatunk, akkor önálló flash player-re is szükségünk van, mely ugyan rendelkezésre áll, de tudatnunk kell vele, hogy számára is adott a külső eljáráshívás lehetősége, hiszen alapesetben nincs semmilyen más program körülötte, így ez le van tiltva benne. A külön program másik problémája, hogy gyakorlatilag egy interfész, amely csak a kommunikációt, és esetleg a szoftveres támogatást szolgálja, és külön meg kell hozzá írni magát a flash alkalmazást. Viszont ha már írtunk egy interfészt a PhysX API és a flash lejátszó között, akkor ehelyett ugyanakkora erőfeszítésből az egész alkalmazást el lehet készíteni a PhysX SDK segítségével, és fölösleges hozzá a futását lassító flash player, és a

különböző nyelvű kódok közötti kommunikáció, hanem minden ilyen teendőt elvégez maga a PhysX-es program. Így az egész munkát leegyszerűsíti, hiszen csak egy programnyelven kell kódolni, a hordozhatóság megmarad, a sebességgel se lesznek különösebb problémák, és még pluszban lehet háromdimenziós is az alkalmazás. Bár ma már a flash alkalmazások is lehetnek háromdimenziósak, mivel a megjelenítést a fentebb tárgyalt okok miatt még mindig elsősorban a processzor végzi, így a háromdimenziós alkalmazás még lassabb, mint a kétdimenziós. Külön flash lejátszó programot, pedig szintén a meg nem térülő plusz munka miatt nem érdemes készíteni, hiszen az elkészült programunkat is fel lehet tölteni az Internetre, ahonnan ugyanúgy le lehet tölteni, mint bármely flash alkalmazást, legfeljebb a böngészőbe nem tud beépülni, és külön kell futtatni.
A PhysX flash játékokban történő alkalmazása tehát egy érdekes téma, és a szakma is mutat némi érdeklődést utána, de olyan kis mértékben, hogy jelenleg nem is lehet tudni, elérhető lesz-e valamikor ezen szórakoztató kis programok számára ez a lehetőség. Ez elsősorban az Adobe fejlesztőin múlik, hiszen már van is egy fejlesztés, amely még igen kezdeti szakaszban van, és még nem ajánlják a használatát, legfeljebb csak tesztelésre. Az Adobe Alchemy célja, hogy a már készen található C/C++ forrásnyelvű programokat újra fel lehessen használni a flash alkalmazásokban. Még egyáltalán nem biztos, hogy valaha is lesz ebből a szoftverből „késztermék”, de talán remény nyújthat a jövőben.

A PhysX használata a játékok területén, elég elterjedtnek tekinthető, bár egyáltalán nem piacvezető szerepű. A lehetőség természetesen adott, hogy valami nagy horderejű eszköz legyen belőle, de egyelőre ez még nem mondható el róla. Az NVIDIA igyekszi a megfelelő marketinggel népszerűsíteni, és van is potenciál az eszközrendszerben. Nagy előnye, hogy a játékokban lehetőséget nyújt a fizikai számítások hardveres gyorsítására, és olyan új effekteket tud, amelyet a vetélytársai még nem. Bár miután megjelenik valami egy játékban, nagyon valószínű, hogy nem sokkal később más játékokban is fel fog tűnni, így ezek az előnyök csak időlegesek, nem tartanak örökké. A PhysX azonban képes a fejlődésre is, és az NVIDIA olyan területeken igyekszik előnyt szerezni, ahol a vetélytársai még elég nagy hátrányban vannak. A keretrendszer elérhető más platformokra is, tehát nem kizárólagosan a személyi számítógéppel rendelkezők élvezhetik a „PhysX élményt”, hanem akár bármelyik játékkonzol tulajdonosok, sőt még iPhone-on is lehet használni az API-t. Az API tudása elég sokrétű, de képességekben nem egy vetélytársa akad.

Látvány terén a PhysX jobban teljesít, mint a többi fizikai motor, de ez talán kétes dicsőség, hiszen a látvány nem a legfontosabb szempont, hogyha fizikai számításokat végzünk. A játékélmény szempontjából sokat dobhat az érzésen az, ha a fizikai számítások látványosan jelennek meg, és erre a legjobb példa már korábban említett Mirror's Edge című játék. A különbségek igen látványosan jelennek meg, abban az esetben, ha a PhysX gyorsítás elérhető, és ha nem. Nagyszerű példa, amikor a gőz a szellőző csatornák álján lassan tovább száll, és egy nyíláshoz ér, akkor lefelé indul meg. Ez más motorokkal még nem kivitelezhető, így ha a PhysX nem működik, akkor egész egyszerűen ez a részlet meg sem jelenik. A füstök szépen hömpölyögnek az ég fel, és a helikopterek rotorjának szele megkavarja őket, míg hardveres gyorsítás nélkül szintét gomolyog a füst, de már nem olyan élethűen. A szövetek is szebben lengedeznek, bár erre a vetélytársak is nyújtanak már megoldást. Az egésszel csak egyetlen probléma van. Bár mindegyik fentebb említett példa igényel fizikai számításokat, de mindegyik csak látványelem, ezért nem okoz gondot, hogyha valaki nem rendelkezik a megfelelő hardverrel.

Itt el is érkeztünk a PhysX talán legnagyobb hátrányához, a hardveres kötöttséghez. A PhysX képességei nagyon meggyőzőek, de ha egy játékban a játékmenetet döntően befolyásoló elemet szeretnénk vele elkészíteni, akkor szembesülnünk kell azzal a problémával, hogyha az nagyobb erőforrásigényekkel rendelkező elem, és emiatt mindenképpen szüksége van a hardveres támogatásra, akkor ez az elem csak GeForce kártyákon lesz elérhető. Így pedig más lesz a játék, ha NVIDIA kártyán fut, és más lesz, hogyha RADEON videokártya végzi a megjelenítést. Ez, pedig komoly gond, hiszen adott esetben a pluszt, amiért akár magát a játékot megveszi egy játékos, a videokártyához kötjük. A PhysX használata ettől még lehetséges, viszont akár komoly bevételkiesést is okozhat, amit egy fejlesztő vagy kiadó sem vállal fel, és az NVIDIA sem tud akkora anyagi támogatást nyújtani. A hardveres kötöttség, pedig abból adódik, hogy a PhysX API hozzá van „láncolva" a CUDA rendszerhez, azaz nélküle nem képes működni. A CUDA rendszert, pedig kizárólag a GeForce kártyák képesek használni. Ez a függőség, pedig komoly gátat szab a rendszer elterjedésének. Az NVIDIA megteheti azt, hogy a PhysX-et átírja az OpenCL szabványnak megfelelően, így az OpenCL szabványt támogató valamennyi videokártyán elérhetővé válnak a PhysX, ami akár új lendületet is adhat a terjedésének. Az is elképzelhető, hogy elégedett a jelenlegi helyzettel, és szeretné megtartani a PhysX-et a saját hardverei számára, remélve, hogy ha a vevő választ, akkor ez a saját irányukba dönti a választás mérlegét. A konkurencia viszont nincs ilyen

dilemmában, így az ő megoldásaik között egyre több helyen tűnik fel az OpenCL szabvány, mely lehetővé teszi a legtöbb grafikus kártyán az egyforma működést. Az egyforma működés, pedig játékmenetbeli változtatásokat hozhat, ami a jövőben hatalmas előnyt jelenthet az adott rendszer számára.

A PhysX egy másik egyedi tulajdonsága, hogy lehetőség van az egyik videokártyát csak és kizárólag a fizikai számításokhoz rendelni. Ez a lehetőség viszont kettős képet fest. Szépen hangzik a lehetőség, és van is értelme, mert vannak olyan játékok, ahol sokat segít egy ilyen megoldás, gyorsabb lesz tőle az alkalmazás. A kettőség ott jelenik meg, hogy ez nem minden programban tapasztalható. Elsősorban az Unreal Engine 3-mal készült programoknál figyelhető meg a gyorsulás, tehát a programokat külön fel kell arra készíteni, hogy ha ez a megoldás a rendelkezésükre áll, akkor használják is ki, vagyis nem a driverek végzik elsősorban ezt, hanem maga a program. Így megint egy olyan lehetőséggel állunk szemben, ami bizony komoly előnyt nyújthatna a PhysX számára, de jelenleg még nem kellőképpen kidolgozott ahhoz, hogy ez tényleg így is legyen.

Akárhogy is legyen, a PhysX jelenleg jelentős versenytárs a piacon, és bár fizetős megoldás, mégis ez lehet az egyik legelterjedtebb alkalmazás, ha az NVIDIA a megfelelő módon használja ki, és bocsátja számunkra a benne rejlő lehetőségeket. Ám ha makacsul ragaszkodik a saját hardveréhez, akkor akár el is tűnhet, ahogyan ez már több zárt szabvánnyal is megtörtént.

A PhysX és a szimulátorok

A játékokon kívül a PhysX-nek egy újabb felhasználási módja a szimulátorok készítésében képzelhető el. Habár a játékok között is vannak szimulátorok, van, amelyik teljes mértékben igyekszik valósághű lenni, van, ami a játszhatóságot szem előtt tartva igyekszik a valóságot a „megfelelő” mértékben lemodellezni. Azonban ha szimulátorokról beszélünk, fontos kitérni azokra az eszközökre is, amelyek nem normál számítógépen futó szoftverek, hanem külön erre a célra épített berendezések. A sok ilyen berendezés teljes egészében célhardver így ezekre külön kitérni sok értelme nincsen, hiszen ezekbe az eszközökbe nem kerül külön beépítésre NVIDIA-s videokártya. Ha mégis kapcsolatban állnak valahogy a GeForce grafikus processzorokkal, akkor is külön csak hozzájuk készített szoftverek segítségével működnek, így a jelen dolgozat tárgyául vett PhysX technológiáról nem igazán beszélhetünk. Vannak azonban olyan szimulátorok is, melyekhez a hagyományosnak tekinthető személyi számítógép mellé valamilyen egyéb periféria kerül, amely a szimulációt igyekszik hatékonyan elvégezni, illetve a szimuláció emberre gyakorolt hatásait érzékelteti magával a felhasználóval. Ezek között a szimulátorok, és a számítógépes játéknak tekintett szimulátorok között elsősorban ez a speciális periféria a különbség. Természetesen nem szabad összekeverni a két kategóriát, hiszen míg az előbbi szimulátorokat elsősorban az oktatásban használják fel, utóbbiak a nagyközönség szórakoztatására hivatottak. A speciális eszközzel rendelkező szimulátorok talán legismertebb alkalmazása a repülőgép szimulátorok. A pilóták oktatásánál nagyon fontos szempont, hogy a pilóta úgy üljön be egy repülőgép botkormánya mögé, hogy előtte a lehető legtöbb veszélyhelyzetet már átélte egy szimulátor kabinjában. Ezek a szimulációk nagyon precízek, és a valósághoz képesti legapróbb eltérés is komoly gondokat okozhat. Így ezeknél a szoftvereknél a valósághűség a legkomolyabb szempont. A pontos modellezés mellett szükség van arra, hogy rengeteg paramétert külön be lehessen állítani, amelyekkel a legkülönfélébb helyzeteket lehet produkálni szimuláció közben. A változásokra, pedig a szoftvernek adott esetben nagyon gyorsan kell reagálnia, így jól jöhet némi hardveres támogatás a számára. Ezeket a szimulátorokat ma már nagyon szerteágazó módon használják. A pilóták oktatásán kívül már készülnek olyan eszközök is, amelyekkel autóvezetést, hajókormányzást, mozdonyvezetést, és egyéb nagy és drága eszközöket lehet az irányításunk alá vonni, mindenféle személyi veszélyeztetés és anyagi kockázat nélkül. A

szimulátoros oktatás napjainkban egyre elterjedtebb módszer, hiszen a tanuló így már „nagyobb tapasztalattal” ülhet be a valódi eszközbe, és már nem lesz számára teljes mértékben ismeretlen a helyzet. Sok olyan helyzetre is fel lehet készíteni a tanulót, amelyekkel a valóságban az esetek túlnyomó többségében nem is találkozik az ember. A kis valószínűséggel bekövetkező esetek gyakorlása azért fontos, mert egyrészt sokkal magabiztosabb lesz a szerkezet kezelője, mert látott már ilyen esetet. Másrészt, pedig rendkívül hasznos tud lenni, ha mégis bekövetkezik ez az esemény a való életben.

A PhysX-ről eddig ezzel kapcsolatban nem sok szót ejtettünk. Ennek oka kettős. Egyfelől a számítógépes szimulációs játékok esetében nem túl elterjedt a PhysX használata, hiszen az egész program lényege a szimuláció, ami a legjobban kidolgozott eleme az egész szoftvernek. A játék fejlesztői, így ezzel a résszel törődnek a legtöbbet, és maguk készítik el az összes ilyen elemet. A PhysX persze ettől még hasznos lehet, de tényleges szerepet csak abban az esetben kaphat, hogyha a számítógép rendelkezik PhysX kompatibilis eszközzel, mert a szoftveres részt mindenképpen a fejlesztők maguk készítik el. Hasznát csak abban az esetben veszik, hogyha van éppen arra az esetre, amit épp használni akarnak, egy PhysX nyújtotta szolgáltatás. Ez a szolgáltatás inkább csak a testek tömegének és ebből származó mozgásának szimulációja esetén elérhető, hiszen a többi lehetőség vagy túl hardverigényes ilyen programok esetén, vagy inkább csak a látványt szolgálja. A látványos megjelenítéssel alapvetően nincsen gond, de egy szimulátornál, legyen az játék vagy oktató szimulátor, a látvány nem az elsődleges szempontok közé tartozik. Játékok esetében még beszélhetünk a megjelenítés fontosságáról, de az oktató szimulátorok esetében a szimuláció közben generált képek nem túl látványosak. Sőt, sok esetben láthatunk olyan képeket egy szimulátorban, amelyeket valódi képekből állítottak össze, így a PhysX által nyújtott látványos elemek nem illenek bele a szimulációba. Másrészről, pedig a látvánnyal az a másik gond, hogy nem fizika. Tehát hiába van szépen kidolgozva, és megjelenítve egy szimulátorban a szimulált világ, ha a fizikai jellemzők nem olyan részletesen kerültek beépítésre a szoftverbe. A PhysX éppen emiatt kerül háttérbe a szimulátorok esetén, hiszen ha a fizikai számításokat érintő általános szoftvert készítünk, akkor nem a legjobb megoldás, hogy magát a működést befolyásoló lehetőség nem érhető el minden számítógépen. Ha egyik gépre telepítjük a programot, akkor így működik, ha a másikra, akkor úgy működik. Ez egy szoftvernél nem megengedhető megoldás. Ha pedig speciális célhardverek készülnek, akkor általában kevés ilyen gép készül,

sorozatgyártásról igen ritkán beszélhetünk, és ennek folytán egy általános grafikus vezérlő nem jellemző része ezeknek az eszközöknek.

A PhysX tehát a szimulátorok esetében háttérbe szorul amiatt, hogy nem a lehető legáltalánosabb módon készült el a hardver támogatása, hanem az NVIDIA igyekszik zárt szabványként használni, ami az iparágban többnyire a háttérbe szorul.

Tudományos felhasználás

A PhysX tudományos felhasználása nem annyira magához a PhysX API-hoz köthető, hanem sokkal inkább a korábban már többször említett CUDA rendszerhez. A két keretrendszer között az a legfőbb különbség, hogy a PhysX is a CUDA rendszert használja, és maga a CUDA rendszer, pedig a PhysX rendszer általánosítottabb verziója, azaz a CUDA a PhysX-ből lett kifejlesztve.

A tudományos felhasználás köréből elsőként a videofilmek, animációk készítése kiemelendő, mert ez az egyik legelterjedtebb felhasználása a játékok után. Ez nem sokban különbözik a játékokban történő felhasználástól, hiszen az animáció készítő programok használóinak célközönsége is nagyjából megegyezik a játékok célközönségével. Sok meglepő dolog itt sincs, de van egy fontos különbség a játékokhoz képest. Az animációknál az elsődleges számításokat, az animáció készítésénél végzik, és nem a lejátszás közben, míg a játékoknál a program futása közben kell a legtöbbet számolni. A felhasználó gépének már csak a videó lejátszása a feladat, ami lényegesen egyszerűbb, hiszen minden a filmen látható elemnek megvan a helye, és előre ki lett számítva. Ez lehetőséget ad arra, hogy a játékokban látható jeleneteknél sokkal nagyobb részletességű, és sokkal több szereplős jeleneteket állítsunk össze. A tárgyak fizikájának számolási terhe nincs időkorláthoz kötve, nem kell adott időre pontos adatokkal rendelkezni, ami megteremti a lehetőségét annak, hogy minden eddiginél több tárgy vegyen részt a jelenet kialakításában, mint eddig. A részecske effektus is nagyobb szerepet tölthet be, hiszen több részecskét lehet így kezelni, és a ködök, füstök még részletesebben kerülhetnek a monitorokra. A folyadékok is egészen újszerű módok kerülhetnek ábrázolásra, és minden eddiginél nagyobb mennyiségben. A rombolás is másképp működik. Az animációknál ténylegesen minden tárgy, minden elem apró darabokra szedhető, és nem okoz nehézséget számítógépünknek, ha akár milliós nagyságrendben jelennek meg a képen. Mindezt annak köszönhetjük, hogy a PhysX motor segítségével az animációt készítő fejlesztők előre le tudnak számolni minden egyes apró részletet. A keretrendszer lehetőséget nyújt a számítások elvégzésére, és mivel időkorlát nem köti a kezünket, így akár napokig is tarthat egy-egy számításigényesebb jelenet elkészítése, a végeredmény a felhasználó gépén ugyanolyan minőségű lesz. Így a PhysX szempontjából ez a felhasználási mód kisebb kihívást jelent, hiszen nem kell azonnal produkálni az eredményt, hanem a főbb szempont az, hogy az eredmény minél pontosabb, és jobb legyen, és a működés közbeni hibák száma minél kisebb

legyen. Tehát a sebesség mellett itt a PhysX stabilitása, megbízhatósága kerül nagyfokú figyelembe. A keretrendszer megfelelő működését mi sem bizonyítja jobban, hogy az Autodesk 3ds Max nevű szoftveréhez is van több PhysX kiegészítő.

Az animáció készítő programokon kívül a PhysX tudományos felhasználása a manapság egyre népszerűbb processzoridő adományozásban teljesedik ki. Pontosabban a PhysX és a CUDA rendszerek segítségével számítógépünk videokártyájának számítási kapacitását adhatjuk kölcsön, például a rákellenes kutatás számára. Az NVIDIA honlapjáról letölthető egy program, melynek segítségével a Stanford egyetem rendszeréhez csatlakozva, a hálózat részeként számítógépünk adatokat kap a központi szervertől, majd az elvégzett számítások után a kapott eredmény visszaküldi, ezzel is segítve a kutatások menetét. GPU-nk idejét még a BOINC (azaz a Nyílt forráskódú önkéntes, illetve hálózatos elosztott számítási rendszer) számára is felajánlhatjuk. A BOINC a Berkeley egyetem kezdeményezése melyhez ma már több más egyetem és vállalat is csatlakozott, többek között a Magyar Tudományos Akadémia számára is adhatunk GPU-nk számítási kapacitásából. A BOINC rendszer felhasználása igen sokrétű, kezdve a matematikától, számításoktól, játékoktól, a mesterséges intelligencián át a biológiai és gyógyszertani kutatásokon keresztül, az összetett alkalmazások fejlesztésektől és az asztrológiai, fizikai, kémiai kutatásoktól a földtani tudományokkal bezárólag nagyon sok minden. A videokártyák számítási kapacitásának igénybevételével ez a számítási mód 24 óránként átlag 5330.48 TeraFLOPS számítást végez, ahogy ez a rendszer honlapjáról is kiderül [URL1]. A legnépszerűbb kutatás jelenleg a MilkyWay@home projekt. Ebben a projektben szinte a világ összes csapata részt vesz. A cél, pedig nem más, mint a Tejútrendszer nagyon pontos és részletes háromdimenziós modelljének az elkészítése a Sloan Digital Sky Survey által begyűjtött adatokból. Így került összefüggésbe a PhysX a csillagászattal. Több program is foglalkozik csillagászati számításokkal. Figyelembe kell venni azt is, hogy nem minden projekt esetén beszélhetünk a PhysX támogatásáról, hiszen nem minden program készül úgy, hogy a PhysX, avagy CUDA, nyújtotta szolgáltatásokat kihasználja. A PhysX-et használó projektek között például lehetőségünk adódik földönkívüli intelligens életformákat keresni rövidhullámú jelek feldolgozásával, melyet a világszerte elhelyezkedő rádióteleszkópok adataiból gyűjt ki a rendszer. Egészen pontosan a rádióteleszkópos megfigyelések adatainak alapján a rendszer megpróbál rövidhullámú jeleket izolálni, melyeknek jelenleg egyetlen ismert természetes forrása sincs, így jelenleg azt a földönkívüli technológiák egyértelmű bizonyítékának tekintik. Ha valamelyik számítógépnek

sikerül elkülöníteni egy ilyen jelet, a tulajdonosa büszkén vallhatja, hogy ő az, aki felfedezte az első földönkívüli életformát. Persze az emberiség már számtalan eszközt bocsátott fel a világűrbe, így ki kell tudni szűrni, az emberi kéz alkotta eszközök jeleit is. A bevezetőben feltett kérdésre így egyértelmű választ kaphatunk: igen, lehetséges új dolgokat felfedezni a világűrben a PhysX rendszer segítségével.

A lelkesedést talán egy picit lelombozza az a tény, hogy mindez nem egészen a PhysX rendszer segítségével történik, ugyanis a PhysX egy elég speciális platform a videojátékok fizikai modelljeinek az elkészítéséhez, így nem alkalmas arra, hogy közvetlenül ilyen számításokat végezzünk vele. Ezekre a számításokra a CUDA rendszert használja a BOINC és a Folding, így bár közvetetten a PhysX is szerepet játszik ezeknek a szoftvereknek a kifejlődésében, de közvetlenül ezt nem lehet ennyire egyértelműen kijelenteni.

A PhysX API

Egy dolog maradt hátra: egy egyszerűbb PhysX alkalmazás leírása. Még mielőtt elkezdeném, fontosnak tartom megemlíteni, hogy az API mellé az NVIDIA mellékel egy távoli hibakereső eszközt is, mellyel a hibakeresés válik egyszerűbbé, és ezt, akár egy másik számítógépről is megtehetjük. A könnyebb kezelhetőség érdekében az API könyvtárszerkezete logikai struktúrát alkot, így könnyebben meg lehet találni a szükséges fájlokat.

Az API 5 részből tevődik össze. Az első modul a PhysXLoader. Ez a modul oldja meg a PhysX verziókezelést, és ez készíti el, kezeli és bontja le az alapvető PhysX objektumokat. Lényegében ez a modul indítja el a PhysX alkalmazást. A második modul a Foundation. Ez a modul definiál különböző tároló, matematikai és segéd funkciókat, amelyeket majd a többi modul használni tud. A harmadik modul a PhysX. Nevéből is kitalálható, hogy ez a modul tartalmazza a PhysX lényegi részét. Ezzel a modullal lehet létrehozni az alkalmazásban szimulált környezetet, testeket, szereplőket, mindent, amit csak a PhysX tud. A negyedik modul a Character. Ez a modul segítséget nyújt a karakterek szimulálására. A PhysX többi moduljával nehéz lenne létrehozni a karaktereket, így erre egy külön modult ad az NVIDIA. Ennek az oka az, hogy a karakterek egész másként is viselkedhetnek, mint az egyéb tárgyak, ezért van külön moduljuk. Például a karakterek tudnak saját akaratukból mozogni, míg a tárgyakat csak mozgatni lehet, maguktól nem tesznek semmit. Az utolsó modul a Cooking. Egy konvex vagy konkáv hálózat létrehozása igen időigényes tevékenység, így ez a modul lehetőséget nyújt arra, hogy ezeket a hálózatokat az alkalmazásunk elindítása előtt létrehozzuk.

A PhysX SDK elindítása rendkívül egyszerűen hatható végre. A következő egyszerű kódrészlet megteszi ezt számunkra:

```
bool initialized = false;
NxPhysicsSDK * pPhysicsSDK =
    NxCreatePhysicsSDK(NX_PHYSICS_SDK_VERSION, NULL,NULL);
if(pPhysicsSDK != NULL){
    pPhysicsSDK->getFoundationSDK().getRemoteDebugger()-
>connect("128.0.0.1", 5425);
    initialized = true;
}
```

Az `NxCreatePhysicsSDK` eljárás létrehoz egy NxPhysicsSDK objektumot, és egy mutatót ad vissza, ha a művelet sikerült, különben pedig `NULL` a visszatérési értéke. Mint látható a függvényhívás után megvizsgáljuk, hogy a visszatérési érték megfelelő-e, és ha igen, akkor az `initialized` változóban eltároljuk, és aktiváljuk a remote debuggert, ami a korábban említett távoli hibakereső eszköz. Jelen esetben a saját számítógépen lévő eszközhöz csatlakozunk az 5424-ös porton, ami a standard port. Ezután elkezdhetjük létrehozni az alkalmazásban szimulálni kívánt környezetet. Lehetőségünk van több környezetet is létrehozni egy alkalmazásban, de a különböző környezetek között nem lehetséges semmiféle interakció. Egy PhysX környezet definiálására az NxScene objektum szolgál.

```
NxSceneDesc sceneDesc;
sceneDesc.gravity.set ( 0, 0, -9.81f );
NxScene* pScene = pPhysicsSDK->createScene(sceneDesc);
```

Első lépésben létrehozunk egy NxSceneDesc környezet leírás objektumot. Majd beállítjuk a gravitációt, amelyet a `sceneDesc.gravity.set` függvényhívással tehetünk meg. A három paramétere a x, y, és z tengelyt jelenti. Jelen esetben a z tengelyen negatív irányba egy a nagyjából a valóságnak megfelelő értéket adunk. Ezután a korábban létrehozott pPhysicsSDK objektum segítségével, létrehozzuk az aktuális környezetet. A környezet most már készen áll szereplők fogadására, akikkel a szimulációt végre kívánjuk hajtani. A PhysX tartalmaz néhány primitív formát. Ezek a következőek: box (téglatest), plane (sík), height field (kiemelkedő lapos terület), capsule (kapszula), sphere (gömb), wheel (kerék). Lehetőségünk van arra is, hogy a szereplőnket több primitív formából rakjuk össze. Az NxActor objektummal tehetjük meg ezt. A következő módon létrehozhatunk egy síkot:

```
NxPlaneShapeDesc planeDesc;
planeDesc.normal = NxVec3 (0, 0, 1);
planeDesc.d = 0.0f;
NxActorDesc actorDesc;
actorDesc.shapes.pushBack(&planeDesc);
NxActor* pActor = pScene->createActor(actorDesc);
pActor->userData = NULL;
```

Az NxPlaneShapeDesc objektum segítségével hozhatjuk létre a síkot. Az `NxVec3()` függvényhívás segítségével, megadhatjuk a sík normál vektorár, mely egy vektor objektum. A

három paraméter szintén az x, y, és z tengelyeknek felelnek meg. A `d` változó pedig az origótól való távolságot adja meg. Az `actorDesc.shapes.pushBack()` metódushívással hozzáadjuk a formát a szereplőnkhöz. Majd a `createActor()` metódussal létrehozzuk a szereplőnket az aktuális környezetben. A `pActor->userData` egy void típusú mutató, melynek segítségével a szereplőhöz bármilyen objektumot hozzákapcsolhatunk, ezzel segítve a szereplő alkalmazásbéli funkcióinak kezelését. Ezután egy tetszőleges tárgyat adhatunk hozzá a környezetünkhöz. Ezeket a megfelelő objektumok segítségével lehet létrehozni. A box-ot az NxBoxShape reprezentálja a sphere-t az NxSphereShape, a capsule-t az NxCapsuleShape, a height field-et az NxHeightFieldShape a wheel-t az NxWheelShape, a plane-t pedig az NxPlaneShape, ahogy ezt az előbb már láttuk. Ezeken kívül lehetőségünk van még konvex forma létrehozására is, melyet az NxKonvexShape objektum segítségével tehetünk meg. A szereplőnk hozzáadása előtt érdemes beállítani, az alapértelmezett anyagtulajdonságokat.

```
NxMaterial* defaultMaterial =
     pScene->getMaterialFromIndex(0);
defaultMaterial->setRestitution(0.3f);
defaultMaterial->setStaticFriction(0.5f);
defaultMaterial->setDynamicFriction(0.5f);
```

Az NxMaterial objektum szolgál az anyagok leírására. A `pScene` környezet objektumunkból lekérjük az alapértelmezett anyagleírást, és beállítjuk neki a rugalmasságát, `setRestitution()`, a nem mozgó felületek közötti súrlódás nagyságát, `setStaticFriction()`, és a mozgó felületek közötti súrlódás nagyságát, `setDynamicFriction()`. Ezután már létrehozhatjuk a szereplőnket. A mozgó szereplőnél szükség van még további paraméterek beállítására is. A position a test helyzetét adja meg, a velocity a kezdő sebességet, az angular damping a forgási ellenállást határozza meg, azaz minél nagyobb ez az érték, annál stabilabb lesz a test, és annál nehezebben kezd forogni. A density pedig az anyag sűrűségét, és így, a tömegét határozza meg. Egy test lehet static, azaz statikus, vagyis nem mozgatható. Lehet dynamic, azaz dinamikus, azaz mozgatható. Valamit vagy egy harmadik típus is, a kinematic, azaz mozgó tárgy, ami az jelenti, hogy a tárgy tud mozogni, de minden őt érő erőhatásra érzéketlen. Ezek ismeretében létrehozhatunk a környezetünkben egy dinamikus dobozt.

```
NxActorDesc actorDesc;
NxBodyDesc bodyDesc;
bodyDesc.angularDamping = 0.5f;
bodyDesc.linearVelocity = NxVec3 (1 , 0 , 0);
actorDesc.body = &bodyDesc;
NxBoxShapeDesc boxDesc;
boxDesc.dimensions = NxVec3(2.0f,  3.0f, 4.0f);
actorDesc.shapes.pushBack(&boxDesc);
actorDesc.density = 10.0f;
actorDesc.globalPose.t = NxVec3( 10.0f , 10.0f, 10.0f);
pScene->createActor(actorDesc)->userData = NULL;
```

Először is létrehozzuk a szereplőleírást, majd egy NxBodyDesc leírást, ami egy testet jelent, melynek megadhatjuk a kezdő sebességet, és a forgási ellenállást, melyet később hozzárendelünk a téglatest alakhoz. A téglatest alak méreteit egy vektor objektum segítségével adhatjuk meg. Ezután a formát hozzárendeljük a szereplőhöz, melynek megadjuk a sűrűségét, és a térbeli elhelyezkedését. Majd végül létrehozzuk a szereplőnket az aktuális környezetünkben. Így készen áll az egyszerű környezetünk arra, hogy kezdődhessen a szimuláció. A szimuláció elindítása szintén egy igen egyszerű, mely egy külön végrehajtási szálon történik. A `pScene` objektumnak a `simulate()` metódusával történik, amelynek egyetlen paramétere, a szimulálandó idő másodpercben megadva. Ez általában szinkronban kell, legyen a megjelenítéssel, így célszerű akkora időintervallumot megadni, amennyi időnként egy képkockát meg szeretnénk jeleníteni. Ezután az eljáráshívás után megjelenítjük a képet DirectX vagy OpenGL kóddal, és begyűjtjük a szimuláció eredményét, azaz a megjelenítendő objektumok következő képen lévő helyét. Ezt a következő utasítás párral érhetjük el:

```
pScene->flushStream();
pScene->fetchResults(NX_RIGID_BODY_FINISHED, true);
```

A szimuláció végén pedig töröljük a felhasznált memóriát.

```
if(pPhysicsSDK != NULL){
        if(pScene != NULL)
            pPhysicsSDK->releaseScene(*pScene);
        pScene = NULL;
        NxReleasePhysicsSDK(pPhysicsSDK);
```

```
        pPhysicsSDK = NULL;
}
```

A PhysX lehetőséget nyújt még kapcsolatok létrehozására is. A kapcsolatok segítségével tudunk több primitív formából egy összetett testet létrehozni, például egy autót. Sokféle kapcsolat létezik. A spherical kapcsolat összeköti a két testet, de bármely irányba elfordulhat a két összekapcsolt test. A spherical kapcsolatot az NxSphericalJointDesc objektum segítségével hozhatjuk létre. Ha kész a kapcsolat leírása, akkor a `pScene` változónk `createJoint()` metódusával hozhatjuk létre. A revolute kapcsolatot gyakran ne csuklópánt kapcsolatnak, mert olyan, mint az ajtók csuklópántja. Az elnevezési séma következetessége miatt, könnyű kitalálni, hogy az NxRevoluteJointDesc objektum segítségével hozhatjuk létre. A prismatic kapcsolatok a mozgást csak az elsődleges tengelyük mentén teszik lehetővé. A cylindrical egyfajta keveréke a revolute és a pricmatic kapcsolatnak, így az elsődleges tengelyük körül foroghatnak, és elfordulhatnak annak mentén. A fixed kapcsolat fix, azaz semmilyen mozgást nem tesz lehetővé. A distance kapcsolat két távoli testet köt össze egy rúd segítségével. Így súlyzószerű formák jönnek létre vele. A point in plane (PIP) kapcsolat egy síkhoz kapcsol egy másik testet. A test a sík mentén mozoghat, és foroghat is. A point on line (POL) kapcsolat egy egyeneshez köti a testet, mely test az egyenes mentén mozoghat csak, de ugyanúgy foroghat. A pulley kapcsolat egy csigát szimulál. Például egy csiga két oldalára két zsákot akasztunk. A kapcsolatoknak megadhatunk paramétereket is. Megszorításokkal szabályozhatjuk a kapcsolatok mozgási szabadságát, például egy revolute kapcsolatnak megadhatjuk a két szélső állását, amik között mozoghat, mint egy rendes ajtó esetén is van. Megadható az is, hogy a kapcsolat törhető-e, azaz, hogy a megfelelő erőhatásra elengedjen. Rendelhetünk motort is a kapcsolathoz, például, hogy a pulley kapcsolat emelje feljebb, vagy eressze lejjebb a felakasztott testeket. Ez az NxMotorDesc objektum teszi lehetővé. A velTarget a maximum sebességet jelenti, a maxForce a maximum erőt jelenti, amellyel megpróbálja elérni a maximum sebességet. A freeSpin segítségével megadható, hogy az összekapcsolt tárgyak foroghatnak-e (kilengés a tehetetlenség miatt). A kapcsolat rugózása is megadható.

Az ütközésdetektálás egyik formája a raycasting. A raycasting sugarak segítségével keresi meg, hogy mely testek ütköztek. A `raycastAnyBounds()` és `raycastAnyShape()` eljárásokkal lekérdezhető, hogy egy sugár mentén van-e ütközés. A `raycastClosestBounds()` és `raycastClosestShape()` eljárásokkal lekérhető a

legközelebbi ütköző forma, és ennek az eseménynek a távolsága. A `raycastAllBounds()` és `raycastAllShapes()` eljárásokkal pedig lekérhető minden forma, amit a sugár érint. Az előbb említett összes eljárás paraméterül vár egy NxRay objektumot, amely a sugarat reprezentálja.

A fejlesztőnek lehetősége van triggerek beállítására, melyek segítségével eseményekhez eseménykezelő eljárásokat tud rendelni. A triggert az NxTriggerReport objektumok segítségével lehet létrehozni, és az `onTrigger()` metódust kell implementálni az esemény kezeléséhez. A kiváltó eseményeket négyféle konstanssal írhatjuk le. Az `NX_TRIGGER_ENABLE` minden eseménynél aktiválja az eseménykezelőt. A események háromféle csoportba sorolhatóak: `NX_TRIGGER_ON_ENTER`, `NX_TRIGGER_ON_LEAVE`, `NX_TRIGGER_ON_STAY`. A három konstans sorban a következőket jelenti: esemény belépéskor, ha egy test bekerül egy területre, esemény kilépéskor, ha kilép arról a területről, és a harmadik, ha bent marad, ahogy a neveikből ez könnyen kitalálható.

A PhysX erőhatások, és nyomatékok segítségével szimulál. Az erőket és nyomatékokat, az `addForce()`, `addLocalForce()`, `addTorque()` és `addLocalTorque()` eljárások segítségével hozhatunk létre. A local verziók a lokális koordinátákat várják. Két paramétere van ezeknek az eljárásoknak, az első az erő vagy nyomaték nagysága, vektorral megadva, a második pedig az erő „módja”. Lehetőség van adott pontokon is erőhatásokat megadni, az `addForceAtPos()`, `addForceAtLocalPos()`, `addLocalForceAtPos()`, `addLocalForceAtLocalPos()` eljárásokkal. Ekkor egy harmadik paraméter lesz az erős „módja” és a második, pedig az adott pozíció. Az erőhatások módjai a következők lehetnek: `NX_FORCE` egy sima erőhatást jelent a lépés első allépésében. Az `NX_IMPULSE` egy impulzust jelent a lépés első allépésében. Az `NX_VELOCITY_CHANGE` sebességváltozást jelent (gyorsítás, lassítás) a lépés első allépésében. Az `NX_SMOOTH_IMPULSE` egy lépés minden allépésében hozzáadja az impulzust. Az `NX_SMOOTH_VELOCITY_CHANGE` sebesség változást jelent a lépés minden allépésében. Végül, pedig az `NX_ACCELERATION` pedig gyorsulást jelent a lépés első allépésében.

Legutoljára maradt a PhysX matematikai támogatása, mely tartalmazza a különböző matematikai konstansokat. Az NxMath egyszerű számokra alkalmazható, az NxMat33 3x3-as mátrixokkal tud dolgozni, az NxMat34 egy NxMat33 és egy NxVec3 objektumot kombinál. Az NxVec3 pedig, ahogy arról már korábban is többször szó volt, egy 3 dimenziós vektort

reprezentál. Az utolsó matematikai segéd objektum pedig, az NxQuat. Az NxQuat a kvaterniók PhysX reprezentációját jelenti. A kvaterniók a komplex számok négy dimenzióra történő nem kommutatív kiterjesztései. A Hamiltoni Aritmetika szerint a kvaterniók a valós számok feletti vektortér. Az alkalmazott matematikában a kvaterniókat a háromdimenziós forgás reprezentálásának egyszerűsítésére használják, ezért is a PhysX része.

A PhysX főbb tulajdonságait ezzel sorra is vettük az SBGames 2006-os Tutorial Ageia PhysX című könyve alapján [6].

Összefoglalás

A PhysX tehát egy fizikai motor, melyet a számítógépes játékok világába terveztek, nem kisebb céllal, mint, hogy megreformálja a számítógépes játékok világát. Ezt a célját részben érte el, hiszen valóban olyan változást indított el, ami a mai napig zajlik, és elég jelentőségteljes a maga nemében. A PhysX ötlete volt ugyanis a fizikai motor hardveres megtámogatása, mely az ötlet születésekor még egyedülálló volt. A hardveres támogatás is csak a PhysX-nél volt elérhető, és szintén az AGEIA munkája nyomán kezdődtek el a tárgyalások a videokártya gyártó cégekkel, hogy PPU-val felszerelt grafikus kártyák jöjjenek létre. Ez a grafikus kártyáknak is fontos szempont lett, hiszen ma már a processzorok is tartalmazhatnak GPU-t, és valamilyen módon meg kell maradni a piacon. A PhysX évekig versenytárs nélkül volt a piacon, de az felhasználók érdeklődése sem volt túl nagy ahhoz, hogy eléggé elterjedhessen. Az OpenCL szabvány megszületésével a fizikai motorok hardveres támogatása elkezdett általánossá válni. A PhysX ekkor már az NVIDIA tulajdonában volt, így a konkurens AMD az OpenCL szabvány segítségével próbált részt szerezni magának a PPU versenyben, és több a PhysX konkurensének számító fizikai motort is támogatni kezdett. Kizárólagos támogatást persze egyiknek sem nyújt, mert a PhysX nagy hátránya pontosan az, hogy zárt, és csak az NVIDIA videokártyáin érhető el hozzá a hardveres támogatás, ami jelenleg a legnagyobb fékezőerőt jelenti az elterjedésében. Az NVIDIA-nak persze lehetősége van tenni ez ellen, de jelenleg nem tudni, hogy van-e ilyen célja a cégnek. A zárt rendszer hátrányai pontosan a zártságából fakadnak. A PhysX bejelentésekor a játékipar nagyon felrezzent a hír hallatán, és nagyon sok stúdió jelentette be, hogy ők már az új motorral dolgoznak. Természetes volt, hogy nagy változásokat vártak tőle a játékok működésében. Akkor még annyi kötöttsége volt, hogy kellett egy plusz hardver a gépbe, ha teljes egészében élvezni szerettük volna a rendszert. A játékmenetbeli változás viszont sajnos elmaradt, mert az egész AGEIA-t felvásárolta az NVIDIA, így a rendszer a GeForce kártyákat erősítette, mely a felhasználók körülbelül felét kizárta a PhysX nyújtotta lehetőségek kihasználásából. A játékfejlesztő cégek éppen emiatt nem tehették le szavazatukat a PhysX mellett, hiszen ha nagy piaci sikert szerettek volna, nem korlátozhatták magukat csak egyféle hardverre, és emiatt a játékmenetbeli változtatások gyakorlatilag lehetetlenné váltak. A PhysX effektek megmaradtak a megjelenítés szintjén, és a legnagyobb igyekezet ellenére sem nagyon találni olyan fejlesztő stúdiót, amelyik bevállalja a PhysX mélyebb, nagyobb

fokú integrációját. Persze van rá példa, hogy egy program PhysX nélkül nem megy jól, de az is csak egy kis kiegészítő, és nem egy teljes értékű játék. Megvizsgáltam ezen kívül, a PhysX elérésének lehetőségeit a flash alkalmazásokból, de ahogy a játékmenetbeli változásoknál, a rendszer korlátozott elérhetősége miatt a flash-nél sem érdemes ebbe sok energiát fektetni. Bár van egy ígéretes lehetőség, és az Adobe már dolgozik egy olyan megoldáson, amelynek segítségével a PhysX elérhetővé válik a flash alkalmazások számára, még az Adobe is nyíltan vállalja, hogy talán ez a program sosem készül el, hanem csak próbálgatják. Az NVIDIA-nak ez esetleg egy újabb lehetőség a keretrendszer népszerűsítésére, és egy olyan hely meghódítására, amelyre mások még próbát sem tettek, de a hardveres támogatás még grafikai szinten is problémás, amire már az Adobe és az NVIDIA egy közös projekt keretében külön nyújt hatékonyabb lehetőséget, de jelenleg még az is tesztelés alatt van.

A szimulátorok között sem sikerült nagy népszerűségre szert tennie a PhysX-nek, hiszen ott a játék lényegi részét a szimuláció jelenti, amit a fejlesztők maguk készítenek el, persze nagy könnyítést jelenthet, ha a már létező PhysX API-t integrálják a szoftverbe, de erre egyelőre kevés példa akad. Az oktató szimulátorok között is terjedhetne a PhysX, hiszen megvan rá a lehetősége, de mivel azoknak a gépeknek a nagy része célhardver, így külön kellene beléjük építeni egy NVIDIA-s grafikus chipet, ami viszont eddigi kutatásaim alapján példa nélküli. Nem kizárható a létezése, de nincs jól látható módon a nagyközönség elé tárva.

A PhysX tudományos felhasználására is találunk példát, bár mivel maga a PhysX motor szándékoltan játékok készítésére lett kifejlesztve, így nem jelenthető ki határozottan, hogy ezekben az esetekben is mindig a PhysX-ről beszélhetünk. Az animáció készítő programoknál még egyértelműen a PhysX API kerül kihasználásra, a tudományos kutatásoknál már inkább a CUDA rendszer, vagy az OpenCL szabvány van integrálva. Tudományos felfedezést a PhysX segítségével csak közvetetten tudunk tenni, de a lehetőség adott, és nap, mint nap több százezer ember él is ezzel a lehetőséggel, persze nem mindegyikük a CUDA segítségével.

Irodalomjegyzék

[1] Gamestar Magazin, 2008 decemberi szám, Móricz Krisztián (Marco),
Live For Speed Vs. Drift, 122-123 o.

[2] Joachim Herrmann: *Csillagok*, Magyar Könyvklub, 1997

[3] Gamestar Magazin, 2006 áprilisi szám, Halász Bertalan (Boe), GDC 2006 (Game Developers Conference) , 28-31 o.

[4] Gamestar Magazin 2006 júniusi szám, Mezei Károly (ZeroCool),
AGEIA PhysX, 112-113 o.

[5] Gamestar Magazin 2007. áprilisi szám, Téglás Gábor (Ashe), Infernal, 80-83 o.

[6] SBGames 2006, Tutorial, Ageia PhysX

[URL1] http://boinc.berkeley.edu/index.php

[URL2] http://www.nvidia.com/object/physx_new.html

[URL3] http://developer.nvidia.com/object/physx_features.html

[URL4] http://developer.nvidia.com/object/physx_release_notes.html

[URL5] http://developer.nvidia.com/object/physx_partners.html

[URL6] http://developer.nvidia.com/object/physx_tips.html

[URL7]
http://prohardver.hu/teszt/geforce_physx_es_ami_mogotte_van/konkurensek_osszegzes.html