Renderidő 1 óra. Sok vagy kevés? (Készítette M. "Youth" Ákos)

Vass Gergely

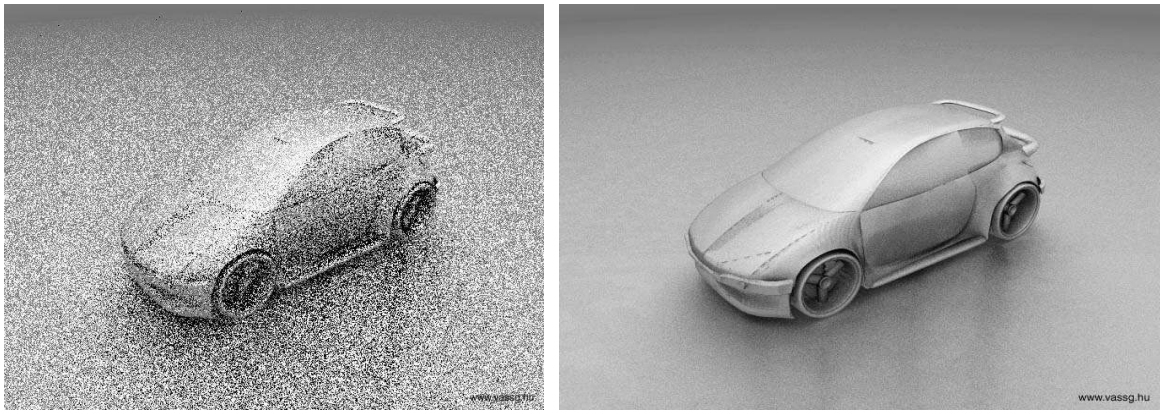
A 3D képgenerálás komplexitása

avagy miért tart olyan iszonyú sokáig???

A következőkben arra keressük a választ, hogy miért ennyire lassú – ha egyáltalán lassú – a globális illumináció számítása, valamint miért adódhat nagyságrendi különbség a képkiszámítási idők között algoritmusonként, vagy akár azonos algoritmussal de más paraméterek mellett.

Manapság minden hobbi 3D-s a *globális illumináció (GI)* lázában ég. Inkább elviseljük a sok-sok perces vagy akár órás renderidőt, de nem mondunk le a fizikailag korrekt fényvisszaverődésekről, fényszóródásról, elmosott árnyékokról stb. Produkciós környezetben azonban még mindig előszeretettel alkalmazzák – a már sokak által elavultnak tekintett – *lokális illuminációs (LI)* algoritmusokat, ahol a gyors számítási sebesség oltárán fel kell áldozni a fizikailag korrekt számítást. Ha szép és valósághű képet akarunk készíteni, bizony sokat kell dolgoznunk a fények és ál-fények, az árnyékok és az anyagi jellemzők beállításán.

Legelőször tisztáznunk kell, hogy mit jelent az, hogy "lassú" egy algoritmus. Mi módon tudjuk jellemezni azt, hogy mennyi ideig tart kiszámolni valamit? Egy egyszerű megoldásnak tűnik az algoritmus futásának idejét megadni, mondjuk másodpercben megadva. Ebben az esetben azonban meg kell pontosan határozni, hogy milyen adatokat dolgoztunk fel (pl. mi volt az a 3D modell amit virtuálisan lefényképeztünk), illetve azt is, hogy pontosan milyen számítógéppel végeztük a számítást. Ez a módszer jónak tűnik különböző konkrét alkalmazások összehasonlítására, tesztelésére, de alkalmatlan egy általános algoritmus hatékonyságának, sebességének jellemzésére.



Ha spórolunk a sugarakkal borzasztó lesz a kép (bal oldal), viszont sok sugár lekövetése – különösen nem erre "kihegyezett" algoritmussal – akár 1 napig is eltarthat (jobb kép, standard Maya sugárkövetéssel).

Természetesen mint minden elképzelhető problémával, így ezzel is hosszú évek óta foglalkozik a tudomány. Az *algoritmusok elmélete* vagy a *bonyolultság elmélet* kicsit túlozva csak arról szól, hogy különböző problémák megoldhatóak-e a gyakorlatban, és ha igen – minimum – mennyi idő alatt, mennyi erőforrás felhasználásával (itt nem dízelolajra kell gondolni, hanem memóriára, processzor időre stb.), illetve azt is vizsgálják, hogy a megoldásra kitalált konkrét algoritmus – maximum – mennyi ideig fog futni. Természetesen megtartva a tisztes távolságot az ijesztő matematikai definícióktól, vessünk egy pillantást arra, hogy a matematikusok milyen mércével mérik azt, hogy milyen gyors egy algoritmus.

Nagy ordó

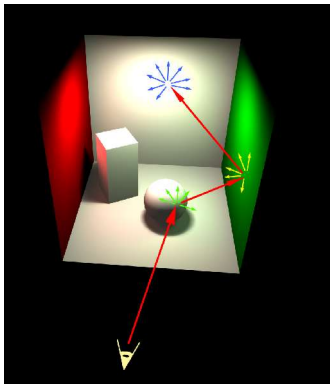
A 3D grafika világánál maradva tekintsük a részecske-rendszerek ("particle systems") szimulációját, és vizsgáljuk meg, hogy bizonyos képzeletbeli algoritmusok milyen gyorsan futtathatók. Mivel a szimuláció nehézsége alapvetően attól függ, hogy mennyi részecskével kell számolni, a részecskék számának függvényében kell megadni a számítás időigényét. Első példánk legyen a részecske-rendszerre ható gravitációs erő. Ez a viszonylag egyszerű feladat azt jelenti, hogy minden egyes részecskét minden szimulációs lépésben – mondjuk kockánként – 9.8 m/s^2-el kell gyorsítani a föld irányába. Ha egy pont feldolgozása mondjuk 5 időegység, akkor a teljes rendszer szimulációja $5n$ ideig tart.

Ha a részecskékre nem gravitációval, hanem egy turbulens, véletlenszerű erővel hatunk, akkor valószínűleg kicsit többet kell számolnunk, azaz a teljes számítási idő mondjuk $15n$ lesz. Igen gyakran a részecskéket ütköztetni szeretnénk egy felülettel, mondjuk egy sík lappal. Ez szintén minden pontra fix idő alatt elvégezhető, tegyük fel $25n$+10 számítási idő alatt (a 10 extra lépés legyen független a részecskék számától). Utolsó feladat a részecskék ütközésének számítása egymással(!). Itt – a legegyszerűbb, és egyben legbutább algoritmussal – minden részecske esetében meg kell vizsgálni az ütközést az összes többivel, ez pedig (n x (n-1))/2 vizsgálat, azaz legyen mondjuk $10n^2$ -$10n$ időegység. Összefoglalva:

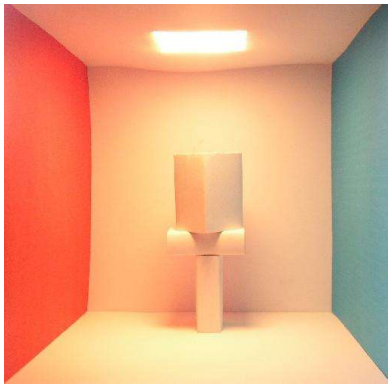
- gravitáció: 5n számítás
- turbulencia: 15n számítás
- ütközés síkkal: 25n+10 számítás
- részecskék ütközése: $10n^2$ -10n számítás

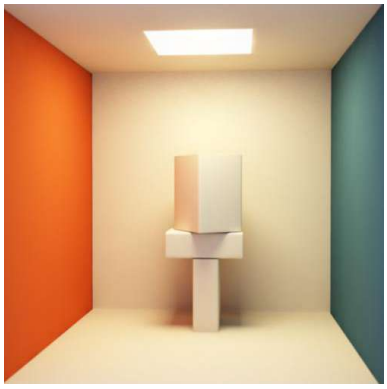
"Egy algoritmus futásának idejénél nem számít sokat, hogy plusz vagy mínusz 10, 100 vagy akár 1000 lépést meg kell tennünk."

Bár mindegyik algoritmus más számítás igényű, mégis rögtön látszik, hogy a negyedik feladat *sokkal* nehezebb mint az első három. Ha ezt mégsem látnánk n helyére írjunk be 10 000-et, ami nem is számít olyan nagyon nagy részecske felhőnek. Ekkor az első három számítás százezres- míg a negyedik billiós (!) nagyságrendű lesz! Egy algoritmus futásának idejénél nem számít sokat, hogy plusz vagy mínusz 10, 100 vagy akár 1000 lépést meg kell tennünk, sőt az sem, hogy 10X 100X vagy 1000X többet kell számolnunk. Az viszont nagyon sokat számít, hogy n növelésével *milyen ütemben* nehezedik a feladat! Gyakorlatias emberek ragaszkodhatnak hozzá,



Minden egyes visszaverődés számításához rengeteg új sugarat kell(ene) megvizsgálni.





Házi Cornell Box (Tari Péter fényképe), és a 25 perc alatt kiszámított virtuális megfelelője (Lipka József, www.abbys.hu)

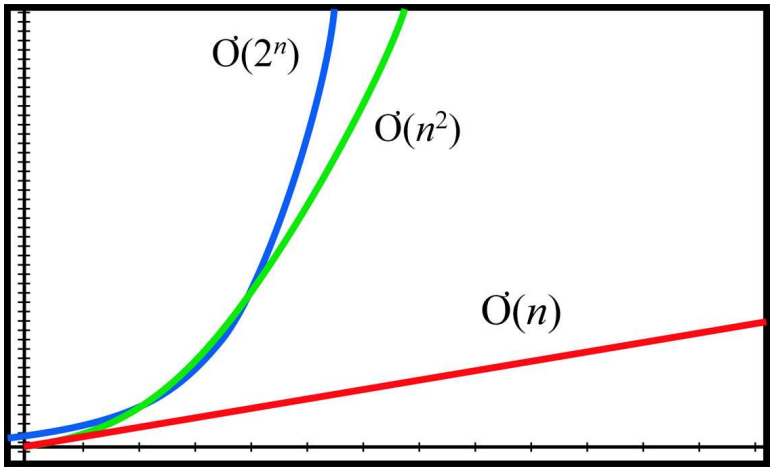
hogy igenis számít, ha 5X gyorsabb az egyik algoritmus a másiknál (ami valóban fontos), de biztosan használhatatlan az a módszer, mely kiszámolásához több tízezer év, vagy éppen a világ összes processzora szükséges!

A matematikában ezt a gondolatot követve vezették be a "nagy ordót" azaz a O-t a számítás erőforrásigényének nagyságrendjének jelölésére. Ezzel a jelöléssel az első három algoritmus $O(n)$ számításigényű, a negyedik pedig $O(n^2)$. Formálisabban fogalmazva, ha egy konkrét algoritmus időigénye $O(f(n))$, akkor ez az időigény mindenképpen kisebb mint $f(n)$ függvény "valahányszorosa". A nagy ordó tehát alkalmas arra, hogy *felső becslést* adjon konkrét algoritmusok számítási igényére. A pontosság kedvéért jegyezzük meg, hogy egy konkrét algoritmus bonyolultsága – melyet az ordóval tudunk jellemezni – illetve a probléma komplexitása nem ugyan az! Bármely problémára adhatunk nagyon rossz algoritmust, ami az idők végezetéig futni fog. A problémák nehézségét pont ezért nem felülről szokták becsülni – azaz nem használják az ordót –, hanem a számítási lépések minimális számát szokták megadni. Most, hogy jellemezni tudjuk a különböző eljárások komplexitását, fogalmazzuk meg, hogy mit nevezünk nehéz feladatnak, pontosabban mit nevezünk lassú algoritmusnak! Az $O(n)$ komplexitású algoritmusok a bemenet növekedésével egyenesen arányosan nehezednek, azaz kétszer annyi részecske feldolgozása remélhetőleg kétszer annyi ideig fog tartani. Ez nem tűnik rossznak. De mit történik azon problémák esetében, melyeket a legjobb esetben is csak $O(n^2)$ nehézségű algoritmusokkal tudunk megoldani? Ezek sajnos nehezebbnek bizonyulnak, de ha nem olyan nagy az n értéke, akkor még kezelni tudjuk (pl. néhány ezer részecske ütközése, még egy hagyományos PC-n is számítható). Képzeljük most el, hogy egy eljárás számításigénye $O(2^n)$! Ha n =2, akkor a számítás 4 időegység, ha n =10, akkor 1000 lépés. Nem is tűnik vészesnek! De mi történik ha n = 1000? Sajnos igen problémásan férne itt el ez a szám! Hogy az $O(2^n)$ komplexitású algoritmusok mennyire nehezek, azaz hogy a 2^n mennyire gyorsan nő n növelésével, azt az alábbi példa jól illusztrálja: vegyünk kezünkbe egy újságpapírt, és hajtsuk ketté. Ezt tegyük meg ismét és ismét (feltételezzük, hogy ezt akárhányszor meg tudjuk tenni). Ekkor az újságpapír vastagsága pontosan 2^n-el arányosan nő. Kérdés: 1000 lépés után milyen vastag lesz a papír? Meglepő módon nagyobb értéket kapunk mint a föld-nap távolság!

Exponenciális robbanás

Ezt a növekedést nevezik exponenciális növekedésnek, illetve az $O(2^n)$, $O(10^n)$ vagy $O(100^n)$ nehézségű algoritmust *exponenciális bonyolultságúnak.* Számítások esetén az exponenciális növekedés ilyen nagy mértékű, és sajnos a gyakorlatban biztosan kezelhetetlen növekedését pedig *exponenciális robbanásnak*. De mi közünk mindehhez? Sajnos igen sok. A 3D grafika alapvető problémája, a fotórealisztikus képek előállítása – a fénysugarak virtuális szimulációjával – pontosan ilyen nehézségű probléma!

Gondoljuk végig, hogy mit is kell kiszámolnunk minden egyes pixel esetében a renderelés folyamán! Meg kell határoznunk, hogy az illető pixelen keresztül mennyi virtuális fény kerül a virtuális kamerába. Amennyiben egy tetszőleges felületet látunk a



Az *n*-nel arányos növekedés igen barátságos, az exponenciális viszont kerülendő.

pixelen keresztül, azt kell kiszámolni, hogy erről a felület-darabról mennyi fény távozik a kamera felé. Ehhez viszont ismernünk kell, hogy a tér minden irányából mennyi fény jut a felületre, sőt elvileg azzal is számolnunk kell, hogy a felület belsejéből mennyi fény lép ki ("subsurface scattering"). Sajnos a számítógép nem tud egyszerre több irányban is számolni, így kénytelenek vagyunk kiválasztani különböző térbeli irányokat - melyek jól lefedik a tér összes irányát – és ezeket a kitüntetett fénysugarakat végigkövetni.

Tegyük fel, hogy (csupán) 50 sugárral szeretnénk a tér különböző irányait lefedni. Számoljuk ki, hogy mennyire lesz nehéz meghatározni a pixelre eső fénymennyiséget, ha maximum *n* számú visszaverődést veszünk figyelembe! A közvetlenül látható felület vizsgált pontjából 50 sugarat bocsátunk ki, hogy megtaláljuk azokat a felületeket, melyekről fény jut az illető pontba. Sajnos azonban az 50 sugár által eltalált *minden újabb* pontból is 50 sugarat kell kilőnünk, hogy meghatározzuk azok színét is, majd azokból is 50 további sugarat követünk stb-stb! Ez összesen 50 x 50 x 50 ... sugár indítását jelenti, azaz $O(50^n)$ nagyságrendű "fényátadást" kel megvizsgálni, ahol n a visszaverődések száma volt. Ez rémisztően nehéznek tűnik, különösen ha végiggondoljuk, hogy 50 sugár az összes térbeli irány lefedésére nem is túl sok. Igazából egy $O(m^n)$ komplexitású algoritmust kellene futtatni, ahol m (a teret lefedő sugarak száma) és n (visszaverődések száma) is nagyon nagy számok.

"Ez a feladat – hasonlóan minden keresési feladathoz – önmagában sem túl egyszerű."

Az előző gondolatmenetben csupán a követendő sugarak számát becsültük, arról egy szót sem ejtettünk, hogy minden egyes sugár esetében meg kell keresni azt a felületet, amelyet az illető sugár elmetsz. Ez a feladat – hasonlóan minden keresési feladathoz – önmagában sem túl egyszerű. Egyszóval, bőven van mit számolnia a processzor-unknak. Bár cikkünkben csak azzal foglalkozunk, hogy a borzasztóan sok fénysugár útját mi módon képesek a programok lekövetni, számtalan ötlet, módszer, algoritmus létezik az összes számítás felgyorsítására. Az egyik tipikus módszer a hatékony adatstruktúrák alkalmazása, melyek többek között a különféle kereséseket nagyon felgyorsítják. Találkozhatunk pl. Mental Ray-ben az un. BSP (binary space partitioning) fákkal, melyek segítségével a sugarak és felületek metszése hatékonyan számítható, illetve a jól ismert "photon map" sem más mint egy foton-találatok tárolására kialakított keresőfa.

Akkor hogyan?

Ha ilyen nehéz feladat a virtuális fényképek előállítása, akkor hogyan tudjuk egyáltalán megoldani? Alapvetően két irány létezik:

Leegyszerűsítjük a problémát, hogy kezelhető nehézségű legyen. Ügyes algoritmusokat alkalmazunk, melyek *valószínűleg* megközelítik a kívánt eredményt, ráadásul viszonylag rövid idő alatt.

Az első kategóriába tartozik az un. lokális illumináció, ami egyáltalán nem veszi figyelembe a többszörös fényvisszaverődéseket, fényszóródást stb. Ez a klasszikus "scan-line" renderelő algoritmusok alapja. Itt nekünk kell "kézzel" létrehozni a realisztikus árnyékokat, felületekről visszaverődő fényeket stb.

A második kategóriába tartoznak a globális illuminációs algoritmusok, melyek igyekeznek nem elhanyagolni semmit, többek között a többszörös fény-visszaverődéseket sem. (A radiosity módszer valahol a kettő között van – hiszen ott is komoly elhanyagolások vannak – de erre most nem térünk ki.) A nagyon komplex problémákra – mert ugye

nem gondoljuk, hogy csak a 3D képek előállítása ilyen nehéz – létezik sok eszköze az alkalmazott matematikának. Esetünkben az okozza a problémát, hogy egy folytonos tartomány – a tér minden iránya – mentén kell összegeznünk értékeket – a beérkező fénymennyiséget – úgy, hogy ráadásul minden egyes pont, illetve irány megvizsgálása legalább olyan nehéz, mint az eredeti feladat. Ez azt jelenti, hogy a pixel kiszínezéséhez végtelen sok irányt kellene megvizsgálni, ahol minden irány vizsgálata olyan nehéz, mint az eredeti pixel kiszínezése. Vagyis – kis túlzással – minél többet számolunk, annál több van hátra. Mivel sok hasonló probléma létezik a matematika különböző területein, ezért nem meglepő, hogy már a 3D grafika születése előtt sikerült fogást találni a problémán. Két ilyen módszer a Monte Carlo és a Las Vegas módszer. Gondolom senkit nem lep meg, hogy mindkettőnek köze van a véletlen számokhoz. A Las Vegas módszer lényege, hogy az algoritmus mindig kiszámolja a jó eredményt, de nem tudjuk pontosan – csak valószínűsíteni tudjuk –, hogy mennyi idő alatt. A Monte Carlo módszer ezzel szemben fix ideig fut, de csak adott valószínűséggel fog korrekt eredményt adni. Ez utóbbi igen közkedvelt a GI algoritmusoknál, szinte mindenhol találkozhatunk vele. A lényege, hogy azokat a véletlenszerűen kiválasztott sugár-irányokat vizsgáljuk meg, arra bocsátunk sugarakat, ahova úgy tűnik "érdemes". Ezt nevezik fontosság szerinti mintavételezésnek. Azokat a sugarak, melyek már biztosan nem fognak jelentős szerepet játszani a pixel színének meghatározásában lelőjük, melyek viszont sok fényt gyűjtenek – vagy éppen hordoznak – tovább követünk. Így bizonyos irányokban akár 10-20 visszaverődést is követünk, más irányokban is nem is vizsgálódunk egyáltalán.

Bár első hallásra rosszul hangzik, hogy csak "valószínűleg" kapunk korrekt eredményt Monte Carlo módszer alkalmazása során, ez a valószínűség elég magas. Mindemellett ha mégsem tökéletes eredményt kapnánk – azaz a pixel nem olyan színű lesz, mint kellene – akkor sem történik katasztrófa, hiszen nagy valószínűséggel a korrekt eredmény közelében maradunk. Átfogalmazva: nagyon kicsi annak a valószínűsége, hogy nagyot hibázzon az algoritmus a pixel színének meghatározásakor. Persze léteznek esetek, amik kifognak az algoritmuson, de ezt már viszonylag kis rutinnal is elkerülik a 3D grafikusok. Az egyik ilyen példa a szűk kulcslyukon beszűrődő fény a sötét szobába. Sajnos igen kicsi a valószínűsége, hogy a kamerából indított sugarak megtalálják az utat a kulcslyukon keresztül a fényforrás felé, és hasonlóan kicsi a valószínűsége annak, hogy a fényforrásból indított fotonok is betalálnak a szobába. Ebben az esetben nagyon zajos lesz a kiszámolt kép. Bár létezik az algoritmusnak olyan változata, mely ezekkel az esetekkel is sikeresen megbirkózik (metropolis algoritmus), a gyakorlatban azonban nagyon ritka az olyan szituáció, amikor nem tudunk szép eredményt kicsiholni a Monte Carlo alapú programunkból.

Igen nehéz szituáció a render algoritmusok számára, de a Monte Carlo algoritmus megbirkózik vele.

Tanulságként mindenképpen kimondhatjuk, hogy azok a GI algoritmusok, melyek néhány perc alatt – esetleg percen belül képesek szép képeket generálni – kis túlzással a tudomány csodájának számítanak. Ne legyünk akkor sem türelmetlenek, ha órákig számol a számítógépünk egy fényszóródásokkal teli képet, örüljünk inkább, hogy nem kell év ezredekig várakoznunk és nem kell a földgolyó összes számítógépét munkába állítanunk!