Renderingstekniker i realtid

Richard Nyberg

Kandidatavhandling

Åbo Akademi

FNT Datavetenskap

Handledare: Jan Westerholm

2017

Referat

Den här avhandlingen beskriver hur strålkastning (eng. ray casting), strålföljning (ray tracing) och radiosity fungerar. Avhandlingen ger en sammanfattning av den grundläggande idén bakom renderingsteknikerna från några av de tillängliga källorna som behandlar ämnet. Några olika realtidsimplementationer av renderingsteknikerna undersöks och jämförs med motsvarande icke-realtidsimplementationer. För radiosity som baseras på hur ljus fungerar i verkligheten undersöks mera realistisk datorgrafik och matematiken bakom olika fenomen. Syftet med med här avhandlingen är att läsaren ska få en inblick i hur de olika renderingsteknikerna fungerar, vilka skillnader som finns mellan realtid och icke-realtid inom samma renderingsteknik och en idé om hur fysiska fenomen överförs till datorgrafik.

Nyckelord: Ray Casting, Ray Tracing, Radiosity, Computer Graphics, Rendering

1. Inledning

Enligt statistik från Anandtech [1] såldes år 2015 totalt 44 miljoner grafikkort, största delen av dem är avsedda för realtidsrendering, dvs. i huvudsak för rendering i datorspel. Med nya och mera krävande teknologier som högre bildfrekvenser eller 4K upplösning som består av ca 4 gånger fler pixlar än 1920x1200 finns det orsaker att undersöka olika renderingstekniker.

Datorgrafik omfattar alla metoder för att visa och behandla bilder och video genom att använda datorer. Rendering innebär den slutgiltiga processen som skapar bilder från datastruktuer som kan innhålla information om ytor, texturer, ljus och skuggor från en synvinkel. Rendering i realtid handlar om att snabbt skapa bilder och visa dem, så att åskådaren uppfattar rörelse. Med hjälp av de visuella effekter som oftast implementeras i renderingsprocessen strävar man efter att skapa realistiska bilder.

Enligt Peter Shirley et al. i boken *Fundamentals of Computer Graphics* från 2009 [7] är rendering inom datorgrafiken en av de grundläggande uppgifterna som krävs för att visa en bild på en datorskärm. En renderingsteknik är ett specifikt sätt att av en 2D- eller 3D-modell, skapa en tvådimensionell bild som kan visas på en datorskräm. I den här avhandlingen har jag valt att behandla några vanliga tekniker: strålkastning (eng. ray casting), strålföljning (eng. ray tracing) och radiosity. Teknikerna kommer

att gås igenom enskilt, men i praktiken kombineras flera tekniker i en renderingsmotor t.ex. för orörliga och rörliga objekt används olika metoder för att uppnå optimal prestanda eller bildkvalitet.

De allra flesta bilder som kan visas på en dators bildskärm är rasterbilder, dvs. varje pixel har en färg och tilsammans skapar alla pixlar en bild. En pixel i en rasterbild är oftast en kombination av olika intensiteter av röd, grön och blå. Andra exempel på rasterenheter är scannrar, digitala kameror och printrar. Ett annat sätt att beskriva bilder är med vektor bilder, istället för att lagra värden för pixlar som behövs för att visa en bild, behövs endast instruktioner hur bilden ska produceras. Eftersom realtidsrendering handlar i huduvsak om rasterbilder, skriver jag endast ut bilder men det är alltid frågan om rasterbilder.

2. Datorgrafik

För att kunna presentera olika renderingstekniker behövs några föklaringar av begrepp som används inom datorgrafiken. Det här kapitlet är inte avsett för att ge en helhetsöverblick över allt som datorgrafiken omfattar, utan är bara menat för att beskriva just såna begrepp som används senare i avhandlingen. Användning av den korrekta terminologin möjliggör en bättre beskrivning av teknikerna.

Renderingsekvationen används i viss renderingstekniker som sträver efter att producera så realistiska bilder som möjligt. Enligt Peter Shirley et al. [7] handlar renderingsekvationen om sambandet mellan utgående ljus från ett område och summan av inkommande ljus från flera riktningar. Eftersom ekvationen är en någorlunda komplicerad integral finns några olika sätt att approximera de värden som behövs. Monte Carlo-metoden är en stokastiskt metod som estimerar integralen och används t.ex. i strålkastning och vägföljning (eng. path tracing) som diskuteras i kapitel 3 och 4. En del av renderingsekvationen är att bestämma mängden ljus som reflekteras i en viss riktning, i kapitel 5 diskuteras hur det beräknas.

Enligt Akenine-Möller et al. i boken *Real-Time Rendering 3rd ed.* från 2008 [19] innebär Global belysning (eng. global illumination) behandling av t.ex. genomskinligheter, reflektioner och skuggor, ordet global syftar på att information om ljus kommer från andra ytor till den som behandlas. Dessutom kan de olika fenomenen som hör till global belysning delas in enligt typen av yta som behandlas,

t.ex. så reflekterar en slät metallyta proportionellt mera än en träyta. Däremot handlar lokal belysning om hur enklare renderingstekniker fungerar eftersom de bara klarar av att beräkna ljus direkt från en ljuskälla eller i godtycklig renderingsteknik där en ljuskälla är avsedd för att avge en koncentrerad belysning.

Före någonting kan renderas behövs någon sorts yta med en textur. Ytor inom datorgrafiken byggs upp av någon enkel basstruktur, oftast trianglar, och desto fler trianglar som används produceras bättre kvalitet. Akenine-Möller et al. [19] beskriver ett exempel på hur olika varianter av texturer påverkar vår uppfattning om hur realistiskt någonting ser ut: texturen för en tegelvägg med en slät yta ser bra ut tills någon bestämmer sig för att undersöka hur den ser ut på nära håll. Utan att ändra på den underliggande strukturen som bygger upp ytan kan en algoritm ändra på ytnormalerna och använda dem i beräkningar för ljus och skuggor så att ytan ser ut att innehålla ojämnheter.

3. Strålkastning (Ray Casting)

En av de tidigaste renderingsteknikerna för realtidsrendering är strålkastning. Strålkastning nämdes först inom datorgrafiken av Scott D. Roth 1982, den grundläggande idén bakom renderingtekniken är att från observerarens öga skickas strålar som på något avstånd träffar en yta eller bakgrunden. Avståndet till träffen beräknas och träffar på kortare avstånd tar upp mera plats på skärmen och träffar på längre avstånd tar upp minde plats på skärmen. Storleken, formen och färgen på det som visas på en bildskärm ges av texturen på en yta som strålarna träffade. Eftersom en stråle går rakt genom alla ytor, kan inte refraktion eller reflektion uppstå eftersom det skulle innebära att strålen reflekteras eller fortsätter i refraktionsvinkeln som bestäms av materialen som ljuset färdas genom.

3.1 Implementation

I boken *Computer Graphics: Principles and Practice* (JH, 2014, s. 403-417) [2] byggs ett strålkastningsprogram. Strålkastningsprogrammet som presenteras i boken kan inte rendera i realtid utan tar några sekunder på sig att rendera en bild. Från det här programmer krävs några kompromisser och optimeringar för att komma till realtidsrendering, vilka dessa är kommer att bli uppenbart då olika implementationer jämförs.

Enligt beskrivningen av strålkastning, byggs programmet upp genom att för varje stråle söka efter första träff. Eftersom det här programmet är modernare än den grundläggande idén, kan det räkna ut hur mycket ljus som kommer från flera källor vid varje träff. Programmet antar att om det finns någon yta mellan ljuskällan och ytan som ska renderas, når inget av det ljuset fram.

För att lägga till skuggor används en stråles genomskärningspunkt med en yta och en funktion testar därifrån om punkten är synlig från någon ljuskälla. Om en punkt inte är synlig från någon ljuskälla, beror det på att någonting är i vägen för ljuset eller så är punkten på den delen som är bortsvängd från ljuset. Dessa punkter bildar en sammanhängande yta som är lika mycket mörkare överallt där skuggan finns, skuggkanten är alltså inte diffus utan följer formen exakt.

Det krävs en utgångspunkt för varifrån renderingen ska börja, ögat är i centrum av bilden men renderigen börjar från ett plan som är vänt rakt mot ögat. Implementeringen av observerarens öga görs genom att definera ett tredimensionellt koordinatsystem där x-axeln pekar till höger, y-axeln pekar uppåt och på z-axeln som pekar mot skärmen kommer ögat och det är riktat i z- riktningen. Vidare defineras bredden på synfältet som en bråkdel av 360° och höjden av synfältet bestäms av bildförhållandet, t.ex. 4:3 eller 16: 9.

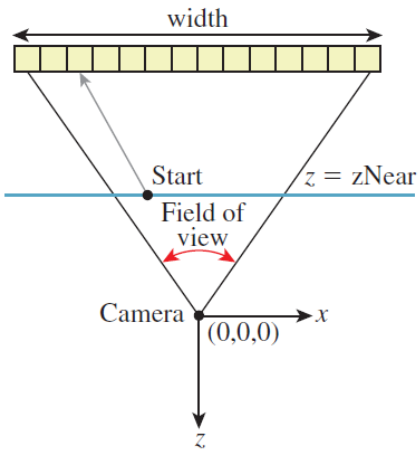


Figure 15.3: The ray through a pixel center in terms of the image resolution and the camera's horizontal field of view.

Figur 1: Synfältet sett uppifrån, renderingen börjar från blåa linjen. (*Computer Graphics: Principles and Practice)*

Det här programmet är gjort så att dess strålar endast kan genomskära ytor uppbyggda av trianglar, men kan utvidgas så att olika uppbyggnader av ytor kan renderas. Uppgiften att rendera en yta uppbygd av t.ex. en sfäriska eller cylinderformade strukturer är enligt författaren lättgjorda utvidgningar av programmet. Strålkastarens förmåga att rendera ett fåtal icke-triangulära objekt kan i vissa fall vara ett stort övertag över andra metoder som fungerar bäst med trianglar och kan behöva hundratals trianglar för att beskriva samma objekt.

3.2 Realtidsimplementation

Hur var det då möjligt att bygga en strålkastningsmotor som kunde köras på en Intel 286 processor som introducerades 1982? I en artikel av Lode Vandevenne från 2007 [3] förklaras hur strålkastningen fungerade i datorspelet Wolfenstein 3D och ges ut en fullständig strålkastningsmotor skriven i C++ . Programmet som undersöks är en modifierad version av originalet.

Den största skillnaden mellan den här realtidsimplementationen och det tidigare exemplet är att i realtidsprogrammet behövs bara en uträkning för varje vertikal kolumn av pixlar på på skärmen. Detta var möjligt eftersom renderingsmotorn endast måste avgöra om det finns en vägg och avståndet till den för varje vertikal linje. På grund av detta är kameran låst i höjdled och kan endast svängas åt höger och vänster i 360° och alla väggar var lika höga och hade raka kanter. Marken och taket är förhandsrenderade och ät helt och hållet enfärgade i olika nyanser av grå och alla objekt är tvådimensionella bilder vilket minskade belastningen på processorn.

Figur 2: För varje vertikal linje i Wolfenstein 3D avgör renderingsmotorn om det finns en vägg, avståndet till den och vilken textur den har. (från http://lodev.org/cgtutor/raycasting.html)

Processen för hur en stråle hittar en vägg görs genom att placera alla väggar så att de tar upp hela rutor i ett koordinatsystem och processen kontrollerar vid varje kant av varje ruta som den färdas genom om en vägg har blivit träffad av strålen. Alla beräkningar har programmerats som vektorer, dvs. spelarens position och avstånden till väggarna. Kameran eller ögat i en riktig 3D-motor behöver ett tvådimensionellt kameraplan, i det här fallet behövs endast en linje på grund av att hela kartan byggs up i en tvådimensionell räcka.

Renderingen producerar inga visuella effekter, inga reflektioner eller olika ljusmängder, istället är texturerna på vissa ytor en mörkare färg för att skapa skuggningseffekten. En mörkare textur kan genereras enkelt genom att dividera värdena på röd, grön och blå med 2. För en dator innebär division med 2 att skifta ett steg till höger, dvs. ta bort den sista biten.

3.3 VolumePro realtidsstrålkastningssystem

Figur 3: Volymrendering av en fot med genomskinlig hud (motsvarande skiktröntgen) [4].

Den riktiga utmaningen var förstås att producera strålkastning enligt den modernare högkvalitetsmodellen i realtid. I artikeln *The VolumePro Real-Time Ray-Casting System* (HP, 1999, Harvard) [5] beskrivs en strålkastningsalgoritm som kan rita upp volymer eller till viss mån genomskinliga objekt, liknande röntgenbilder, i 30 bilder per sekund. Artikeln beskriver ett system, som består av både mjukvara och hårdvara, I det här systemet är hårdvaran ett PCI-kort.

Enligt artikeln kan systemet rendera 500 miljoner interpolerade, Phong skuggade, sammansatta sampel per sekund. Vilket motsvarar 16 miljoner voxler i 30 bilder per sekund. Den här prestandan är möjlig på grund av parallell processering, mera specifikt handlar det om fyra stycken 125 MHz parallellkopplade datavägar som kan

beräkna en tredimensionell pixel, även känd som en voxel, per cykel. Enkel matematik bekräftar att 4 * 125 * 10^6 = 500 miljoner voxler. Vad som egentligen menas med interpolering och Phong skuggning förklarar Overveld V.och Wyvill B. i artikeln *Phong Normal Interpolation Revisited* [6] från 1997. Det handlar om att interpolera ytnormaler, dvs. fylla på med fler ytnormaler mellan redan kända ytnormaler med antagandet att ytan är menad av vara slät och beräkna färger för pixlarna i något objekt med hjälp av de intrpolerade ytnormalerna. Phong skuggningen är alltså en lokal metod, den påverkas inte av andra ytor. Jämfört med liknande metoder, t.ex. Gouraud skuggning, är Phong skuggning mera krävande och producerar mer realistiskta reflektioner.

Själva renderingen görs med en algoritm som bestämmer om bilden eller objekten ska processeras först, fördelen med den här algoritem är att voxlerna kan läsas och processeras som tvådimensionella skivor som är vinkelräta mot kameraplanet. Idén med volymrendering är att rendera genomskinliga tredimensionella modeller och ett av ändamålen för systemet är att skapa kirurgiska simulationer eller ultraljud. Dessutom kan simulationen genast reagera på olika typer av ändringar av parametrar.

4. Strålföljning (Ray Tracing)

Strålföljning kan ses som en utveckling av strålkastning, på samma sätt som i strålkastning skickas strålar från en kamera eller ett öga, strålarna färdas genom ett kameraplan som är vänt rakt mot kameran och träffar sedan någon yta. Skillnaden i de grundläggande idéerna är att strålarna nu fortsätter rekursivt en eller flera gånger från en träff med en yta och flera nya strålar kan uppstå med uppgiften att simulera reflektion, refraktion eller skuggning. Resultatet är en mera fotorealistisk bild för kostnaden av en större belastning på grafikprocessorn, eller i icke-realtidsrendering oftast processorn. Den höga kvaliteten på bilder rendererade med strålföljning gör tekniken ideal för icke-realtidsrendering, men även ett framtida mål för realtidsrendering.

4.1 Hur strålföljning fungerar

I boken *Fundamentals of Computer Graphics* [7] finns en lång redögörelse för hur strålföljning fungerar, som handlar om allt mellan den bakomliggande idén till

specialeffekter. Inom renderingen finns det två tillvägagångssätt för i vilken ordning en scen renderas, strålföljning behandlar hela bilder (eng. image-order), varje pixel behandlas en efter en. Den andra metoden behandlar varje objekt och tar reda på dess inverkan på andra objekt, båda metoder kan producera samma bild men på olika sätt.

Från samma bok påstås följande, uppgiften att visa en tredimensionell scen på en platt tvådimensionell skärm har redan studerats av målarkonstnärer långt före datorns tid. Vanligtvis använder både artisten och datorgrafiken ett linjärt perspektiv, konstnärens metoder för att få rätt perspektiv kan repliceras med enkla matematiska regler. Dessutom behöver kamerastrålarna en matematisk representation, vektorer passar utmärkt för jobbet då det som behövs är en utgångspunkt, riktning och avstånd. Dessutom behövs en matematisk representation av en triangel eller andra polygon för att beräkna en stråles träffpunkt.

4.2 Strålföljning i realtid

I en artikel från Intels egna nätsidor [8], beskrivs ett forskningsprojekt där datorspelet Quake Wars renderas med en strålföljningsmotor. Tidiagre försök av strålföljning i datorspelet Quake 3 har genomförts år 2004, det krävde en samling av 20 datorer med dubbla processorer i varje dator för att rendera i 512x512 upplösning med en bildhastighet på 20 bilder per sekund. En lyckad demonstration för utvecklarna, men givetvis var det inte vettigt för konsumenter. Artikeln skrevs 2009, nu behövde utvecklarna endast en dator med fyra av den tidens senaste 2,66 GHz Intel Dunnington (Xeon) processorer för att uppnå 1280x720 upplösning med 20 till 35 bilder per sekund.

Figur 4: En scen från Quake Wars renderad helt och hållet med strålföljning (från Intels nätsidor [8]).

Strålföljningsrenderingen är enligt artikeln lättare att jobba med för programmerare, på grund av att specialeffekter och skuggor i den konventionella skanlinjerenderingen behöver räknas ut som uppskattningar över flera renderingsomgångar. Dessutom sparas belysningsinformationen i texturer med begränsad upplösning. Med strålföljning behövs endast en kontroll för att ta reda på om ytan kan nås av någon ljuskälla. På samma sätt som i det första exemplet på strålkastning i avsnitt 3.1 skickas en skuggningsstråle från träffen med en yta mot varje ljuskälla, om det finns någon yta före en ljukälla, då är punkten i skugga.

4.3 Nvidias strålföljningsmotor

Nvidia har utvecklat en strålföljningsmotor, NVIDIA® OptiX™, som får fritt användas till och med för kommersiellt bruk [9]. I artikeln *Incremental Instant Radiosity for Real-Time Indirect Illumination* från 2007 [10] berättar författarna om hur de har använt Nvidias strålföljningsmotor för realtidsrendering och hur de har undersökt prestandan med ett grafikkort och kombinationen av flera likadanna grafikkort. De visar att renderingsmotorn är kapabel till att uttnyttja kombinationen av två grafikkort, men ger mest uppsnabbning av renderingen bara där det krävs mycket beräkningskraft. De förklarar att OptiX-motorn fungerar på Quadro och Tesla seriernas grafikkort, men enligt Nvidia även nya GeForce-kort [9]. Motorn förser utvecklaren med ett applikationsprogrammeringsgränssnitt (API), som ger utvecklaren ett smidigt sätt att använda färdigt utvecklade stabila funktioner.

I samma artikel beskrivs hur strålföljning har testats för olika typer av scener, några fordon gjorda av en miljon trianglar, en ko med olika typer av reflektiv yta och genomskinliga snökristaller som faller från himlen. Renderingen av snökristallerna presterade sämst på grund av det stora antalet refraktions- och reflektionsstrålar som förgrenar sig vid första och andra träffen med en genomskinlig yta. Bäst presterade den Phong skuggade kon, med tre gånger så hög bildfrekvens som snökristallerna. Från det här experimentet drogs slutsatsen att en viktig utmaning inom strålkastningen är den stora variationen i prestanda för olika scener.

4.4 Strålföljningens framtid

Strålföljningens största användningsområde i realtid är för att välja objekt i 3D-rymd. För program som behöver bestämma vilket objekt som är synligt i en pixel är strålföljning den ideala metoden [7]. Det finns några exempel där realtidsstrålföljning

gör den huvudsakliga renderingen, men inga stora kommersiella produkter. I en artikel av Jacco Bikker diskuteras några problem med strålföljning [11]. De områden som han anser behöver mera forskining är, för primära strålar och skuggningsstrålar behövs ett sätt att knyta ihop sammahängange strålar. En metod för att minska på antalet strålar som används för mjukare skuggor, global belysning och kantutjämning (eng anti-aliasing). Utöver det, behövs hårdvarustöd för strålföljning och sätt att introducera strålföljning för utveckalre.

En liknande renderingsteknik som kalls vägföljning (eng. path tracing) har också använts för icke-realtidsrendering i animerade filmer redan 1998 och har nu gjorts som en realtidsmotor. Artiklarna *The Brigade Renderer: A Path Tracer for Real-Time Games* (2012) och *The Path to Path-Traced Movies* (2014) av Bikker och Schijndel respektive Christensen och Jarosz förklarar allt om vägföljning [12, 13]. Vägföljning är väldigt likt strålföljning, en stråle scickas igen till en yta och en skuggningsstråle till ljuskällan, skillnaden är att ytterligare en stråle skickas vidare några gånger från en träff för att bedöma intensiteten på det indirekta ljust som når ytan. I en scen kan den här beräkningen av indirket ljus innebärara att längre in i en oupplyst tunnel är mörkare än nära utgången, den här effekten är inbyggd i vägföljning men inte i strålföljning.

Distribuerad strålföljning är ett annat sätt att rendera diffusa skuggor och kantutjämning, tekniken föreslogs redan 1984 av Robert L. Cook et al. [14], idén är att först ökas antalet strålar per pixel och beroende på hur de används produceras olika effekter. T.ex. utspridda sampel av strålar som färdats genom ett material producerar halvgenomskinlighet eller sampling på olika tider producerar rörelseoskärpa. År 2006 publicerades en forskningsrapport om interaktiv distribuerad strålföljning, med hjlälp av nya opimeringar lyckades de rendera en scen med upplösningen 512x512 med 16 sampel per pixel i 1-20 bilder per sekund [15]. Sedan 2006 har det gjorts stora framsteg inom processor- och grafikkortsindustrin, med den senaste teknologin borde distribuerad strålföljning vara möjlig i realtid, men något sådant genomförande har inte dokumenterats.

5. Radiosity

Stråföljningens idé om hur ljus beter sig är begränsat av testet om en yta kan nås direkt av en ljuskälla, några varianter av strålföljningen har förbättrat idén men modellerna baserar sig inte helt och hållet på den korrekta fysiken. En mera fysiskt korrekt metod utvecklades från forskning inom värmestrålning och belysningsteknik och började kallas radiosity. Cohen och Wallace förklarar i boken *Radiosity and Realistic Image Synthesis* [16] hur och varför tekniken fungerar. Radiosity beräknar hur ljus beter sig med en ekvation som baserar sig på lagen om energins bevarande. De första versionerna fungerade endast i utrymmen där alla ytor kunde ses från varandra, några år senare utvecklades metoder som klarade av alla utrymmen på grund av den hierarkiska indelningen av omgivningen.

Från samma bok framgår att tekniken gör alla beräkningar helt oberoende av åskådarens position, den slutgiltiga bilden kan givetvis ändå påverkas av åskådarens synvinkel. Själva radiosity renderingen sker alltså oftast som förhandsrendering och kan användas i realtidsprogram. En viktig sidoeffekt av den indirekta belysningen och reflektioner är att färgerna också överförs. Tekniskt sett så kan den ursprunglinga radiosityteknikenses som ett sätt att lösa renderingsekvationen med antagandet att all reflektion av ljus sker likformigt i alla riktningar, ytan är i det fallet isotropisk och reflektionen är diffus.

Grunden för hur radiosity fungerar kommer från formlerna för radiometri som är en del av optik och mäter elektromagnetisk strålning. I samma bok defineras radiositet, B, enligt fysiken eller mera specifikt inom radiometrin, som energin per ytenhet (W/m2) som lämnar en yta, uträkningen görs med följande formel:

$$B = \int_{\Omega} L_o \cos\theta \, d\boldsymbol{\omega}$$

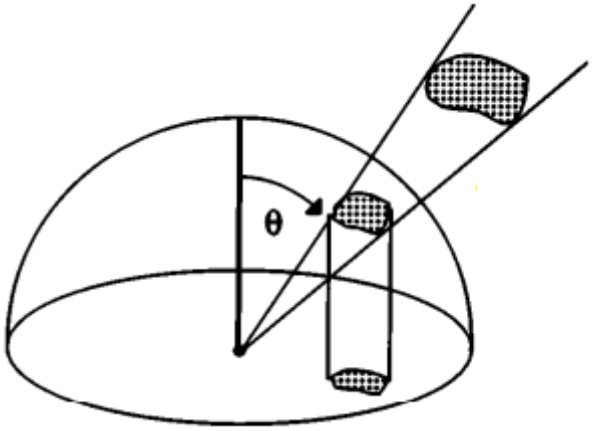
där L_o är utgående radians ($W/sr/m^2$) och $\cos\theta \, d\boldsymbol{\omega}$ är den projicerade rymdvinkeln (se figur 5). Motsvarande energi som infaller på en yta, även känt som irradians, E, beräknas genom att integrera över den infallande radiansen över ett halvklot enligt följande formel:

$$E = \int_{\Omega} L \cos\theta \, d\boldsymbol{\omega}$$

Där L är den infallande radiansen. I specialfallet då alla ljusstrålar är parallella, som sker då en enda avlägsen ljuskälla lyser upp en yta blir integralen förenklad till:

$$E = E_0 \cos \theta$$

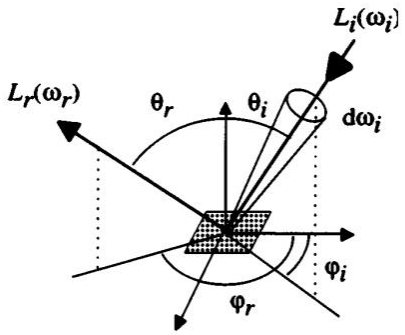
Där E_0 är energi per ytenhet av vinkelrät area från den avlägsna ljuskällan.



Figur 5: θ och en rymdvinkel (från *Radiosity and Realistic Image Synthesis* [16])

En viktig observation är att mängden reflekterat ljus är proportionell mot irradiansen på en icke-genomskinlig yta, den här observationen leder till den dubbelriktade reflektans fördelningsfunktionen (eng. Bidirectional reflectance distribution function). BRDF defineras som:

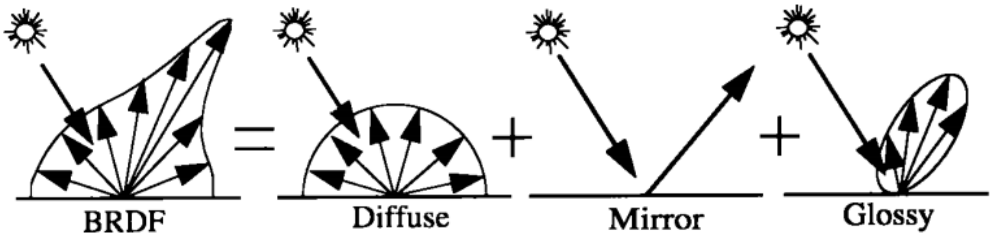
$$f_r(\vec{\omega}_i \rightarrow \vec{\omega}_r) \equiv \frac{L_r(\vec{\omega}_r)}{L_i(\vec{\omega}_i) \cos \theta_i \, d\omega_i}$$



Figur 5: Infallande och projicerad radians (från *Radiosity and Realistic Image Synthesis* [16])

Den här funktionen har använts i många olika modeller för att beskriva reflektion inom datorgrafiken, förutom i radiosity används en aningen enklare version av BRDF i t.ex. Phong reflektionsmodellen. Cohen et al. föreslår att i praktiken är det

lämpligt att se BRDF som summan av spegellik reflektion, ideal diffus reflektion och glansig reflektion.



Figur 7: En praktisk modell av BRDF (från *Radiosity and Realistic Image Synthesis* [16]).

Det finns mera beskrivningar av hur ljus beter sig i andra förhållanden, men det som nu behövs för att en dator ska kunna effektivt beräkna en komplicerad funktion är en numerisk metod. Den finita elementmetoden kan användas för att approximera partiella differentialekvationer, i det här fallet handlar det om att summera basfunktioner efersom linjära kombinationer av basfunktioner kan representera kontinuerliga funktioner. Uträkningen som datorn gör blir alltså:

$$B(\mathbf{x}) \approx \hat{B}(\mathbf{x}) = \sum_{i=1}^{n} B_i N_i(\mathbf{x})$$

Där B(**x**) är radiosity funktionen, B_i och N_i är basfunktioner och **x** är en punkt i en liten del av en yta, oftast hörnen i en triangel. Det här är grunden till hur och varför radiosity fungerar i datorgrafik.

5.1 Stegvis ökande radiosity

I artikeln *Incremental Instant Radiosity for Real-Time Indirect Illumination* av Saumli Laine et al. som publicerades 2007 [17] presenterades en implementering av radiosity som uppdaterar en bråkdel av alla virtuella ljuspunkter i varje bild i ett realtidsprogram. Virtuella ljuspunkter är alltså punkter som nås av en riktig ljuskälla och som i sin tur fungerar som mindre effektiva ljuskällor. De som skrev artikeln har inte behövt använda sig av några förhandsberäkningar av ljuset och de använder ett vanligt grafikkort för att rendera några olika scener.

Istället för att förhandsberäkna det indirekta ljuset och gå miste om möjligheten att göra förändringar i realtid eller tvärtom att uppdatera hela ljusberäkningen varje bild och riskera väldigt låg bildfrekvens så har de gjort en kompromiss mellan dem. För att få en bild av hur beräkningen av det indirekta ljuset presterar så påstår de att en uppdatering av 4-8 virtuella ljuspunkter per bild av de totala 256 virtuella ljuspunkterna i en scen uppbyggd av ca. 80 000 trianglar tar ungefär 71% av renderingstiden.

I deras version av radiosity används en del strålföljning, strålar från en ljuskälla används för att konstruera de virtuella ljuspunkterna och från de virtuella ljuspunkterna skickas också strålarna vidare en gång till någon yta. Deras egna insats som gör att metoden fungerar bra är återanvändningen av virtuella ljuspunkter, även för rörliga ljuskällor. För mycket rörelse av ljuskällor leder ändå till att programmet tvingas kassera fler virtuella ljuspunkter än vad som kan renderas till nästa bild och leder till en tillfällig försämring av kvaliteten på skuggor.

5.2 Radiosity i realtid

I magisteravhandlingen *Real-time Radiosity* av Magnus Burenius som publicerades 2009 [18] presenteras några sätt att anpassa radiosityalgoritmen för att fungera i realtid. Den första versionen gör antagandet att hela scenen är statisk, med det antagandet behöver synfaktorerna endast beräknas en gång. En synfaktor innebär inom radiometri, proportionen mellan strålning som lämnar en yta och träffar en annan yta. Uträkningen av dessa synfaktorer är enligt Burenius det steget som tar längst i den klassiska radiosityalgoritmen. De enda variationerna av ljusförhållanden som är möjlig blir då justering av ljusstyrkan från ljuskällorna, och flyttande av ljuskällorna men förflyttningsrörelsen fungerar inte i realtid. Den här anpassningen av radiosity visas med en C++ implementation som renderar en scen med 915 olika områden som kan påverka varandra om de är synliga från varandra. I den här scenen skeddde växelverkan mellan områden totalt 390 000 gånger och en bild kunde genereras varje 10 ms.

Figur 8: till vänster indelningen av områden, till höger resultatet efter interpolering över områdena. (från Burenius M., *Real-time Radiosity* [18])

I samma avhandling förbättras den tidigare idén genom att gruppera ihop områden till områden som ökar i storlek med avstånd till området de påverkar i en bräkning. De större områden används för beräkningar som inte behöver vara lika noga på grund av att de är långt ifrån varandra och intensiteten på ljuset avtar på grund av att det sprids i olika riktningar. Områden som är nära varandra och vinklade så att de kan påverka varandra grupperas inte ihop för att behålla precision. Det här är den grundläggande idén bakom det som kallas hierarkisk indelning i radiosity. En implementation gjord på det här sättet klarade av att beräkna nästan lika många växelverkan men av områden som är mindre och därför producerades bättre bildkvalitet.

6. Diskussion

-

KÄLLFÖRTECKNING

[1]Shilov Anton, *Discrete Desktop GPU Market Trends Q2 2016: AMD Grabs Market Share, But NVIDIA Remains on Top* , AnandTech. (hämtat 5.3.2017) http://www.anandtech.com/show/10613/discrete-desktop-gpu-market-trends-q2-2016-amd-grabs-market-share-but-nvidia-remains-on-top

[2] Hughes John F., Van Dam Andries, *Computer Graphics Principles and Practice 3rd ed.*, Ohio, Addison-Wesley, 2013

[3]Vandevenne Lode, *Lode's Computer Graphics Tutorial – Raycasting* (hämtat 25.3.2017)
http://lodev.org/cgtutor/raycasting.html

[4] UC Davis, *Interactive Volume Visualization*,
http://idav.ucdavis.edu/~okreylos/ResDev/VolVis/FootHighQuality.png

[5] Hanspeter Pfister, Hardenbergh Jan, *The VolumePro Real-Time Ray-Casting System*, Los Angeles, California; New York, N.Y, Harvard, SIGGRAPH '99, 1999
https://dash.harvard.edu/bitstream/handle/1/4141472/Pfister_VolumePro.pdf?sequence=2

[6] Overveld Van, Wyvill B., *Phong Normal Interpolation Revisited*, The Netherlands, Canada, Eindhoven University of Technology, University of Calgary, ACM Transactions on Graphics, Vol. 16. No. 4, p. 397-419, 1997
http://webhome.cs.uvic.ca/~blob/publications/p397-van_overveld.pdf

[7] Shirley P. Marschner S., *Fundamentals of Computer Graphics 3rd ed.*, Florida, CRC, 2011

[8] Pohl Daniel, *Quake Wars* Gets Ray Traced*, Intel, (hämtat 26.3.2017)
https://software.intel.com/en-us/articles/quake-wars-gets-ray-traced/

[9] *NVIDIA® OptiX™ Ray Tracing Engine*, Nvidia, (hämtat 27.3.2017)
https://developer.nvidia.com/optix

[10] Ludvigsen H., Elster A. C., *Real-Time Ray Tracing Using Nvidia OptiX*, Trondheim, Norway, Dept. of Computer and Info, Norwegian University of Science and Technology, EUROGRAPHICS 2010/ H. P. A. Lensch and S. Seipel, 2010
http://www.idi.ntnu.no/~elster/pubs/eurographics-ludvigsen-elster.pdf

[11] Bikker J., *Real-time Ray Tracing through the Eyes of a Game Developer*, Breda, The Netherlands, IGAD / NHTV University of Applied Sciences, IEEE/EG Symposium on Interactive Ray Tracing 2007
http://ieeexplore.ieee.org/document/4342584/

[12] Bikker J, van Schijndel Jeroen, *The Brigade Renderer: A Path Tracer for Real-Time Games*, Breda, The Netherlands, ADE/IGAD, NHTV Breda University of

Applied Sciences, 2012
https://jbikker.github.io/literature/The%20Brigade%20Renderer%20-%20A%20Path%20Tracer%20for%20Real-Time%20Games%20-%202012.pdf

[13]Christensen Per H., Jarosz Wojciech, *The Path to Path-Traced Movies*, Pixar Animation Studios; Dartmouth College, Foundations and Trends in Computer Graphics and Vision Vol. 10, No. 2 (2014) p. 103–175
http://graphics.pixar.com/library/PathTracedMovies/paper.pdf

[14] Cook Robert L., Porter Thomas, *Distributed Ray Tracing*, Computer Division Lucasfilm Ltd. SIGGRAPH'84, Computer Graphics Volume 18, Number 3 July 1984
http://artis.inrialpes.fr/Enseignement/TRSA/CookDistributed84.pdf

[15] Boulos Solomon, Edwards Dave, *Interactive Distribution Ray Tracing*, SCI Institute, University of Utah, Technical Report UUSCI-2006-022
https://graphics.stanford.edu/~boulos/papers/cook_tr.pdf

[16] Cohen Michael F., Wallace John R., *Radiosity and Realistic Image Synthesis*, USA, Academic Press Professional, 1993

[17] Laine S., Saransaari H., *Incremental Instant Radiosity for Real-Time Indirect Illumination*, NVIDIA Research, PDI/DreamWorks, Helsinki University of Technology, Remedy Entertainment, Eurographics Symposium on Rendering (2007), http://www.nvidia.com/docs/IO/67074/laine2007egsr_paper.pdf

[18] Burenius M., *Real-time Radiosity*, KTH, Master's thesis, 2009
http://www.nada.kth.se/~burenius/BureniusExjobb2009.pdf

[19] Akenine-Möller Tomas, Haines Eric, Hoffman Naty, *Real-Time Rendering Third Edition*, A K Peters, Ltd., Wellesley, Massachusetts, 2008