

Thord Andersson, Mikael Karlsson

Grafikkortbaserad simulering av radar i syntetiska omgivningar

TOTALFÖRSVARETS FORSKNINGSINSTITUT

Sensorteknik

Box 1165

581 11 Linköping

FOI-R--1634--SE

April 2005

ISSN 1650-1942

Teknisk rapport

Thord Andersson, Mikael Karlsson

Grafikkortbaserad simulering av radar i syntetiska omgivningar

Utgivare Totalförsvarets Forskningsinstitut - FOI Sensorteknik Box 1165 581 11 Linköping	Rapportnummer, ISRN FOI-R--1634--SE	Klassificering Teknisk rapport
	Forskningsområde 4. Ledning, informationsteknik och sensorer	
	Månad, år April 2005	Projektnummer I3508
	Delområde 42 Spaningssensorer	
	Delområde 2	
Författare/redaktör Thord Andersson Mikael Karlsson	Projektledare Thord Andersson	
	Godkänd av Johan Söderström	
	Uppdragsgivare/kundbeteckning FOI Innovationsfond	
	Tekniskt och/eller vetenskapligt ansvarig	
Rapportens titel Grafikkortbaserad simulering av radar i syntetiska omgivningar		
Sammanfattning (högst 200 ord) En tydlig trend inom försvarsmakten är att simuleringar får en allt viktigare roll för metod- och algoritmutveckling, utbildning av personal och utvärdering av ny materiel. I rapporten undersöker vi förutsättningarna för att kunna göra kostnadseffektiva radarsimuleringar i interaktiva hastigheter med hjälp av vanliga konsumentgrafikkort. Rapporten kan delas upp i tre delar där: • utvecklingen av grafikkort diskuteras i termer av prestanda och programmerbarhet. • vanliga beräkningsmetoder för radarmålarea diskuteras och jämförs med renderingstekniker från datorgrafiken. • en experimentimplementation som använder programmerbara grafikkort för radarsimulering redovisas. Den metod vi rekommenderar för framtida interaktiv radarsimulering är att man implementerar högfrekvens-approximationen GO/PO med hjälp av Photon Mapping, som är en modern global belysningsmetod från datorgrafiken. Vidare kan vi rekommendera användningen av grafikkort som en billig och kraftfull beräkningsresurs även för generella, icke-grafiska, vetenskapliga beräkningar.		
Nyckelord datorgrafik, grafikkort, GPU, GPGPU, radarmålarea, radarsimulering, strålföljning		
Övriga bibliografiska uppgifter	Språk Svenska	
ISSN 1650-1942	Antal sidor: 39 s.	
Distribution enligt missiv	Pris: Enligt prislista	

Issuing organization FOI – Swedish Defence Research Agency Sensor Technology P.O. Box 1165 SE-581 11 Linköping	Report number, ISRN FOI-R--1634--SE	Report type Technical report
	Programme Areas 4. C4ISTAR	
	Month year April 2005	Project no. I3508
	Subcategories 42 Above water Surveillance, Target acquisition and Reconnaissance	
	Subcategories 2	
Author/s (editor/s) Thord Andersson Mikael Karlsson	Project manager Thord Andersson	
	Approved by Johan Söderström	
	Sponsoring agency FOI Innovationsfond	
	Scientifically and technically responsible	
Report title (In translation) Graphic card based simulation of radar in synthetic environments		
Abstract (not more than 200 words) In the Armed Forces, simulations are getting an increasingly important role for method- and algorithm development, personnel training and evaluation of new equipment. In the report, we examine the prerequisites for making cost effective radar simulations in interactive frame rates with the help of ordinary off-the-shelf (COTS) consumer graphic cards. The report can be divided in three parts where: - the development of graphic cards is discussed in terms of performance and programmability. - common methods for radar signature estimation, and their relation to rendering techniques from the computer graphics community, are discussed. - an experiment application is presented. The application is using a programmable graphics card for radar simulation in a synthetic environment. The method we recommend for future interactive radar simulations is the high frequency approximation GO/PO implemented with Photon Mapping, which is a modern global illumination method derived in the field of computer graphics. Furthermore, we recommend using graphic cards as an inexpensive and powerful computation resource, even for generic, non graphic, scientific calculations.		
Keywords computer graphics, graphic card, GPU, GPGPU, radar cross section, radar simulation, ray tracing		
Further bibliographic information	Language Swedish	
ISSN 1650-1942	Pages 39 p.	
	Price acc. to pricelist	

Innehåll

1	Bakgrund	6
2	Interaktiv radarsimulering i syntetiska miljöer	7
3	Simuleringsmetoder för Radar.....	9
3.1	Klassiska beräkningsmetoder för radarmålarea	9
3.1.1	Analytiska Metoder.....	9
3.1.2	“Exakta” Numeriska Metoder.....	9
3.1.3	Högfrekvensmetoder.....	10
3.2	Beräkningsmetoder från Datorgrafiken.....	10
3.2.1	Lokala belysningsmodeller	11
3.2.2	Globala belysningsmodeller	11
3.3	Datorgrafik för Radarsimulering.....	14
4	Utvecklingen av konsumentgrafikkort	16
4.1	Historik.....	16
4.2	Allmänna beräkningar på grafikkort	19
4.3	Radarberäkningar på grafikkort	20
5	Simulering av radar med programmerbara konsumentgrafikkort ..	22
5.1	Använda programvaror och programpaket	22
5.1.1	OpenGL	22
5.1.2	OpenSceneGraph, OpenProducer och osgNV	22
5.1.3	Cg från Nvidia	23
5.2	Scengrafer.....	23
5.2.1	Använd scengrafstruktur.....	23
5.3	Fysikalisk modell för simuleringen.....	24
5.3.1	Transformering och sampling av scenen	24
5.3.2	Spridningsmodell.....	25
5.3.3	Hantering av skuggfenomen	27
5.4	Begränsningar & brister i dagens grafikhårdvara	28
6	Simuleringsexempel och resultat.....	29
6.1	Visuell rendering och kommandotolk	29
6.2	Visualisering av framräknat rådata till radarsimulering.....	30
6.2.1	Alternativa utdata från fragmentprogrammet	31
6.3	Estimering av avståndsprofiler och RCS-beräkningar	32
7	Sammanfattning och slutsatser	34
	Tack till.....	35
	Referensförteckning.....	36

1 Bakgrund

En tydlig trend inom försvarsmakten är att simuleringar får en allt viktigare roll för utbildning av personal, utvärdering av ny materiel och metodutveckling. Inom sensorforskningen har simuleringar av sensorernas interna funktion använts länge, men med den växande datorkraften kan nu även sensorsns samspel med dess omgivning simuleras. Detta har givit nya möjligheter att snabbt testa och utvärdera nya sensorkoncept och algoritmer, då framför allt inom multisensorteknik och datafusion, utan att behöva göra tidsödande och kostsamma mätserier.

Idag finns det därför verktyg för många sensortyper som i realtid och med godtagbar kvalitet simulerar sensorn i en syntetisk omgivning. Tyvärr saknas det dock fortfarande sådana verktyg för radarsimulering, vilket innebär att en så pass viktig och kritisk militär sensor som radar idag inte kan, med godtagbar kvalitet, simuleras "on-line" tillsammans med andra sensorer och delsystem. I denna förstudie har vi därför undersökt möjligheterna till att kunna göra radarsimuleringar i interaktiva hastigheter och med en godtagbar kvalitet. Undersökningen har gjorts i två steg:

För det första har vi studerat olika beräkningmetoder för radarsimulering, både traditionella tekniker och intressanta nyutvecklade tekniker från datorgrafiken. Beräkningsmetoderna har studerats med avseende på sin förmåga att klara av komplexa scener, fysikalisk korrekthet, förberedelsekrav på den syntetiska modellen samt hur effektivt de går att implementera.

För det andra har vi studerat en het tekniktrend, den explosiva utvecklingen av ”off the shelf” grafikkort, och undersökt dess potential för att kunna implementera intressanta radarberäkningsmetoder. Vi har också undersökt möjligheterna att utnyttja grafikkorten för andra typer av beräkningar såsom linjär algebra och bildbehandling.

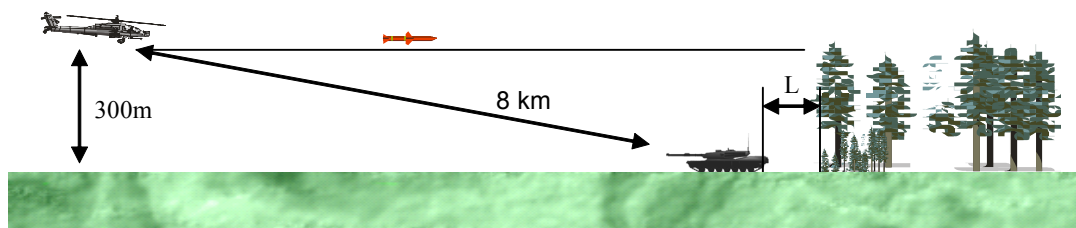
2 Interaktiv radarsimulering i syntetiska miljöer

När man arbetar med algoritmutveckling inom t.ex. detektion, klassificering, datafusion och målföljning så är det en stor fördel om man har tillgång till bra simuleringsverktyg. Man kan då sätta upp testscenarier av olika slag i lämpliga syntetiska omgivningar och på så sätt få tillgång till sensordata som kan användas till att utvärdera olika aspekter och kombinationer av algoritmerna. Kraven på simuleringens prestanda i termer av fysikalisk korrekthet, noggrannhet och hastighet varierar beroende på vad man för tillfället vill testa eller utvärdera. I början av utvecklingen är man vanligtvis intresserad av att kontrollera att programmen är logiskt korrekta, gör ungefär vad de skall och att de fungerar robust för olika typer av indata och situationer. Kraven på simuleringarna är då att de är snabba och flexibla, vilket tillåter täta och effektiva testningar, snarare än full fysikalisk korrekthet och noggrannhet. De senare faktorerna blir dock viktigare då utvecklingen går mot sitt slut och man istället vill utvärdera hur väl systemet skulle fungera i verkliga scenarion. Dessa simuleringar får därför ta den tid som krävs för att kvalitén på utvärderingen skall bli bra. Även i detta fall är det dock en fördel om man har tillgång till en snabb simulator som arbetar i nära realtid då man i denna kan sätta upp sina utvärderingsscenarion interaktivt och få en känsla för vad den riktiga simuleringen kommer att visa. En snabb simulator kan också vara av stor nytta vid t.ex. undervisning och uppdragsplanering, så att man kan testa olika händelseförlopp och ”vad händer om”-scenarier interaktivt med direkt feedback.

I denna förstudie har vi koncentrerat oss på denna snabba och interaktiva typ av simuleringar av radarsensorer i syntetiska miljöer. Det är här viktigt att påpeka att även om simuleringens hastighet och interaktivitet är centrala måste fortfarande simuleringens kvalitet med avseende på fysikalisk korrekthet och noggrannhet vara ”tillräckligt” bra för den givna applikationen för att vara användbar. Därför har vi, vid undersökningen av olika simuleringsmetoder, speciellt studerat hur skalbara de är mellan kvalitet och hastighet, samt vilka fel och artefakter som uppstår då hastighet prioriteras.

Anledningen till att dessa simulatorer hittills har saknats på marknaden¹ är att radarsimuleringar är väldigt beräkningstunga vilket först nu har blivit hanterbart tack vare den snabba datorutvecklingen och framsteg inom simuleringsalgoritmeforskningen.

Figur 1 visar ett typiskt scenario som vi vill kunna simulera.



Figur 1: Ett typiskt simuleringsscenario

En helikopter har skjutit en robot mot en stridsvagn. Roboten innehåller en multisensormålsökare bestående av en radarsensor, en IR-sensor samt mjukvara för att detektera, klassificera och följa mål. Vi vill simulera anflygningsförloppet mot stridsvagnen och undersöka hur målsökarens prestanda varierar med anflygningsgeometri, stridsvagnens beteende, väder, klotter från mark och skog osv.

¹ De produkter som finns, t.ex. RadarWorks gör en väldigt enkel simulering av radar vilket är otillräckligt för många applikationer.

För att kunna simulera radarsensorn i detta scenario krävs att vi korrekt kan:

- propagera den elektromagnetiska strålningen från sensorns antenn ut i scenen
- beräkna strålningens interaktion med och mellan de ytor och objekt som finns i scenen.
(Reflektion, transmission etc.)
- propagera den återvändande strålningen från scenen till sensors antenn

Vi kommer i nästa kapitel undersöka olika metoder för att göra detta.

3 Simuleringsmetoder för Radar

3.1 Klassiska beräkningsmetoder för radarmålarea

Att beräkna hur det elektromagnetiska fältet sprids och interagerar i en scen med objekt med komplicerad geometri och av olika material är ett svårt problem och har studerats av många genom årens lopp. Många olika algoritmer har tagits fram som alla har sina för- och nackdelar. Det som skiljer mellan de olika algoritmerna är främst:

- hur beräkningstunga de är
- hur minneskrävande de är
- hur fysikaliskt korrekta de är; dvs. vilka fysikaliska fenomen klarar algoritmen av att hantera (t.ex. när- eller fjärrfält, multipelreflektioner, spekulär/diffus spridning, diffraktion, krypvågor, koherens, skuggeffekter etc.)
- vilka typer av geometrier de klarar av:
 - komplicerade eller mycket enkla
 - stora eller små relativt de aktuella våglängderna
- hur mycket för- och efterarbete de kräver. En del algoritmer klarar direkt av generella 3D-scener medan andra kräver mycket manuell anpassning av scenen.

Nu följer en uppräknings och en kort beskrivning av de vanligaste metoderna. (Se även [1] av Fagerström som är en utmärkt översikt och på vilket det följande är baserat.)

3.1.1 Analytiska Metoder

Här beskriver man de infallande fälten, de ingående geometrierna och aktuella randvillkor matematiskt och löser sedan Maxwells ekvationer analytiskt för en exakt lösning av den mot mottagaren infallande fältet. Man har tyvärr bara hittat lösningar för väldigt enkla kroppar såsom sfärer och cirkulära plattor. Ibland förenklar man problemet genom att t.ex. anta att det infallande fältet endast har väldigt höga/låga frekvens-komponenter. Man brukar då kalla lösningarna för analytiskt approximativa.

De analytiska metoderna är oanvändbara i praktiken för vanliga scener men intressanta ur teoretisk synpunkt då de kan ge en grundläggande förståelse för ett fenomenets uppkomst och natur.

3.1.2 "Exakta" Numeriska Metoder

Det finns tre vanliga metoder i denna kategori:

- Finite-Difference Time-Domain (FDTD)
- Finite Element Method (FEM)
- Method of Moments (MOM)

Alla dessa metoder utnyttjar numeriska metoder för att lösa Maxwells ekvationer i olika formuleringar. Gemensamt för dessa metoder är också att de klarar av mycket komplicerade objekt och att resultaten generellt är tillförlitliga. Tyvärr är metoderna väldigt beräknings- och minneskrävande vilket i praktiken innebär att stora objekt inte går att hantera. Dessutom krävs en del förbearbetning av scenen och objekten då metoderna kräver olika typer av partitionering av dessa.

3.1.3 Högfrekvensmetoder

Till skillnad mot de föregående metoderna, som mer eller mindre är numeriska lösningar till de grundläggande elektromagnetiska ekvationerna, är dessa metoder anpassade för fall då de aktuella våglängderna är små relativt de ingående objekten och deras detaljer. Vågegenskaper såsom diffraktion kan då ignoreras vilket väsentligt förenklar problemet och gör metoderna beräkningseffektiva.

Grundmetoderna kallas för Geometrical Optics (GO) och Physical Optics (PO). GO baserar sig på klassisk strålgångsoptik där man räknar med speglade kroppar och linser. Detta är också grunden till den enklaste formen av ray-tracing inom datorgrafiken. Metodens snabbhet och möjlighet att klara av multipelreflektioner gör den attraktiv för stora och komplexa objekt. Metoden kräver inte heller någon partitionering av scenen eller objekten. En ren GO-implementering har dock ett antal begränsningar och svagheter vilket ger ofullständiga eller ofysikaliska resultat, vilket gör att man brukar kombinera GO med andra kompletterande metoder.

PO baserar sig på vågekvationerna men gör ett antal antaganden vilket i praktiken gör den till en högfrekvensmetod. Den kan (men behöver inte) implementeras med strålföljning vilket gör den till en bra kompletterande metod till GO. Bl.a. tillåter PO mer komplicerade geometrier och diffusa reflektioner jämfört med en "ren" GO.

Både PO och GO saknar stöd för diffraktion vilket har gjort att man har försökt ta fram kompletterande metoder för detta. Några av dessa är:

- Geometric Diffraction Theory (GTD): Kompletterar GO med grundläggande diffraktionseffekter baserade på geometrisk optik. Den finns en rad förfiningar även för GTD vilka kompenserar för olika svagheter och artefakter:
 - Unified Theory of Diffraction (UTD)
 - Unified Asymptotic Theory (UAT)
 - Method of Equivalent Currents (MEC)
- Physical Theory of Diffraction (PTD): Kompletterar PO med grundläggande diffraktionseffekter baserade på fysikalisk optik. PTD är mer generell än GTD, men också svårare att implementera. En utbyggnad av PDT är ILDC, Incremental Length Diffraction Coefficients, som är en variant av MEC ovan.

Sammanfattningsvis kan man konstatera att alla metoder har sina fördelar och nackdelar och man får välja metod efter de geometri-, kvalitets-, beräknings- och minneskrav som föreligger i en viss problemformulering. Man kan också kombinera olika metoder till en s.k. hybridmetod. För dessa hybrider använder man olika metoder för olika delar/detaljer i scenen och kombinerar resultaten. Detta kräver dock vanligtvis manuell förbearbetning av scenen.

3.2 Beräkningsmetoder från Datorgrafiken

Eftersom vårt mål är att kunna göra radarsimuleringar i syntetiska omgivningar i interaktiva hastigheter, är det bara naturligt att undersöka vilka beräkningsmetoder som används inom datorgrafiken [2], eftersom fotorealism i realtid är ett ständigt mål inom datorspelsbranschen.

Ett vanligt antagande inom datorgrafiken är att geometrisk optik används som den underliggande fysikaliska modellen för hur ljus färdas i och mellan olika medier. Denna modell [3, 4, 5] beskriver ljusfenomen såsom reflektion, skuggning och refraktion men klarar ej, i grundutförandet, av mer avancerade fenomen såsom diffraktion, interferens, polarisation, dispersion, fluorescens och fosforescens. Modellen antar också att ljuset har oändlig hastighet, vilket innebär att om en ljuskälla slås på kommer ljussättningen av den syntetiska omgivningen direkt nå det nya jämviktsläget.

Viktiga begrepp inom datorgrafiken är direkt och indirekt belysning, samt spekulära och diffusa reflektioner.

Direkt belysning har man när en ljuskälla direkt lyser på en yta, t.ex. när en skrivbordslampa lyser på en bok. *Indirekt belysning* har man när en yta belyses av reflektioner från andra ytor. Om man t.ex. tänker sig att skrivbordslampan ovan är det enda som lyser i ett rum utan fönster, så ser man i alla fall mycket av rummets ytor fastän dessa inte är direkt belysta, i "line-of-sight", av lampan. Mycket av det vi uppfattar som en "naturlig belysning", härrör från den indirekta belysningskomponenten.

Reflektionerna från en yta är generellt komplicerade fenomen som beror på ytans materialegenskaper och struktur samt det infallande ljusets egenskaper såsom frekvens och polarisation. Detta innebär att reflektionen av ljus som faller in mot en yta med en viss vinkel, generellt är en icke-trivial funktion av den utgående vinkeln. En funktion som brukar användas för dessa ändamål kallas BRDF [2, 3, 4, 5], *Bidirectional Reflectance Distribution Function*. Två vanliga extremfall av denna funktion är s.k. spekulär och diffus reflektion. Spekulär (spegel) reflektion kallas förhållandet då det infallande ljuset speglas i ytnormalen så att den ingående vinkeln är lika stor som den utgående. Spekulär reflektion ser man i blanka ytor, såsom kastruller, speglar etc. Vid diffus reflektion reflekteras det infallande ljuset isotropt i alla riktningar vilket man kan se på matta ytor, t.ex. väggar, tidningspapper etc. Normalt är dock reflektionerna från ett material en komplicerad blandning av dessa specialfall.

3.2.1 Lokala belysningsmodeller

Den i särklass mest använda beräkningsmetoden idag för 3d-grafik är den s.k. rasteringsmetoden, som alla vanliga grafikkort implementerar mycket effektivt i hårdvara [6, 7, 8]. Anledningen till detta är att en s.k. *lokal belysningsmodell* används, vilket innebär att färgen på en ytpunkt beräknas endast med hjälp av de lokala ytegenskaperna, kamerans position samt positionerna och färgerna hos ett antal punktljuskällor. Detta innebär att ljussättningen för ett ytfragment är helt oberoende av ljussättningen för ett annat, vilket gör att man kan behandla alla fragment parallellt, snabbt och effektivt. Den lokala belysningsmodellen kan dock inte hantera en rad intressanta ljusfenomen såsom skuggning (ett fragment "vet" inte att ett annat är i vägen för ljuskällan), reflektion och transmission. Dessutom klarar modellen endast punktformade ljuskällor. I datorspel använder man sig därför av en uppsättning tricks som kan ge illusionen av t.ex. skuggning och reflektioner, men dessa är ofta kraftigt förenklade, specialanpassade och begränsade till den aktuella syntetiska omgivningen.

Man kan illustrera en belysningsmodells kapacitet med de möjliga ljusvägar modellen tillåter. I den lokala belysningsmodellen färdas fotonerna från ljuskällan till en yta (genom mellanliggande objekt som ignoreras) och vidare till kameran. Om reflektion tillåts, blir antalet möjliga ljusvägar många fler; fotonerna studsar nu mellan objekten innan de träffar kameran. Ju fler uppsättningar av ljusvägar vi tar hänsyn till, desto mer realistisk ljussättning erhålls. Sammanfattningsvis kan man konstatera att den lokala belysningsmodellen inte är tillräcklig för våra behov utan att en *global belysningsmodell* krävs, dvs. en modell där ljussättningen av ett ytfragment tar hänsyn till, och är beroende av, ljussättningen av andra ytfragment.

3.2.2 Globala belysningsmodeller

Det finns idag många olika globala belysningsmodeller men de flesta baserar sig på två grundläggande tekniker, *Ray Tracing* [2, 3, 4, 5, 9, 10, 11] (punktsampling) och *Radiosity* [2, 3, 4, 5, 11] (finita element). Det finns numera även hybridtekniker som försöker förena det bästa av de bägge teknikerna, se t.ex. [3, 5]. 1986 presenterade Kajiya i sin berömda artikel [12] *Renderings-ekvationen* där han visar att både radiosity och Monte Carlo raytracing egentligen är specialfall av en och samma ekvation.

3.2.2.1 Ray Tracing

Ray tracing är en samplingsteknik som följer ljusstrålar genom den syntetiska modellen. I den ursprungliga varianten följer man strålarnas färd baklänges från kameran till ljuskällorna. Denna tidiga variant klarade enbart ljusvägar med rent spekulära reflektioner samt direkt belysning. För att klara av diffus reflektion samt mer generella reflektioner enligt BRDF-funktioner utökades ray tracing med tekniker från Monte Carlo metoderna [13, 14]. Med dessa sprids ljusstrålarna stokastiskt i scenen enligt sannolikhetsfördelningar som bestäms av ytornas BRDF:er, och av en lobfunktion vid ljuskällorna. Monte Carlo ray tracing kan hantera alla typer av ljusvägar inom geometrisk optik inklusive svåra fenomen såsom fokuspunkter (s.k. *caustics*). Den stora nackdelen med denna metod är att den stokastiska ansatsen kräver stora mängder av strålsampel, och därmed processortid, för att signal/brus förhållandet skall vara godtagbart i de resulterande bilderna. Många ansatser har föreslagits för att effektivisera metoden, t.ex. bidirectional Monte Carlo ray tracing, där man samtidigt följer strålar från både kameran och ljuskällorna. Trots detta är dessa, s.k. "unbiased" Monte Carlo metoder väldigt tidsödande. Genom att tillåta andra konvergenskrav ("biased Monte Carlo") kan effektiviteten dock ökas väsentligt.

En stor fördel med strålföljningsmetoderna är att de medger en väldigt flexibel representation av geometrin i den syntetiska omgivningen utan större förändringar; geometrin kan t.ex. beskrivas analytiskt, med en CAD-modell eller procedurrellt. Dessutom behöver inte geometrin förbehandlas på något sätt vilket t.ex. radiositytekniker kräver. Jämfört med rastereringsmetoden är strålföljning också väldigt effektivt m.a.p. geometris komplexitet. Vanliga implementationer av rastereringsmetoden skalar linjärt med antalet polygoner i scenen emedan strålföljningsmetoderna skalar logaritmiskt.

Stora ansträngningar har gjorts för att möjliggöra raytracing i interaktiva hastigheter med vanliga CPU:er, se t.ex. [15, 16, 17, 18, 19]. Även implementationer på specialhårdvara förekommer, t.ex. den aktuella (2005) SaarCOR [20, 21] som är en FPGA-prototyp med en relativt låg klockfrekvens men som klarar att rendera komplexa scener (från t.ex. spel) i realtid. Detta visar den oerhörda effektiviteten och potentialen hos hårdvaruimplementeringar. Vi kommer att titta närmare på detta senare när vi diskuterar utvecklingen av konsumentgrafikkort (GPU:er).

3.2.2.2 Radiosity

Radiosity är en finit element metod som delar upp scenen i en mängd ytpatchar. Dessa patchar har inte ett-till-ett förhållande till polygonerna i den underliggande geometrin utan kan vara både större och mindre beroende på det aktuella belysningsfallet. Detta kräver alltså en förbehandling av scenen vilket är en nackdel jämfört med strålföljningsmetoderna. Ytpatcharna kan absorbera och emittera ljus och radiosity går ut på att man beräknar jämviktsläget hos ljusfördelningen i scenen. Detta görs genom att lösa en uppsättning linjära ekvationer som beskriver ljusutbytet mellan ytpatcharna. De ursprungliga radiositymetoderna hanterade enbart ytor med rent diffus reflektion, dvs. det reflekterade ljuset strålar ut lika mycket i alla riktningar, men senare varianter klarar även mer generella reflektioner även om enkla krökta spekulära ytor fortfarande ger problem. En intressant följd av att radiositymetoderna beräknar ljusutbytet mellan alla ytpatchar är att lösningen är totalt vyoberoende. Detta innebär att då ekvationssystemen väl är lösta kan man lätt beräkna godtycklig vy i scenen utan extra kostnad vilket är en stor fördel jämfört med ray-tracing metoderna. Detta fungerar dock naturligtvis bara så länge scenen är rent statisk. Radiositytekniken är effektiv när modellerna är enkla och ytorna diffusa men beräkningskraven ökar dramatiskt med modellernas komplexitet och mer avancerade reflektionsmodeller; komplexiteten har numeriskt visat sig vara $O(n \log n)$ för de snabbaste implementationerna till skillnad mot $O(\log n)$ för raytracing, där n är antalet scenelement. En anledning till detta är att metoderna beräknar värden för varje patch i hela scenen. Vissa varianter försöker öka effektiviteten genom att utnyttja t.ex. klustring, hierarkiska tekniker och vyberoende beräkningar. En annan nackdel med radiositymetoderna är att kvaliteten på lösningarna är starkt beroende på hur man gjort tesseleringen av geometrin, dvs. hur man har fördelat ytpatcharna över den underliggande

geometrin. Detta är inget lätt problem och kräver normalt manuell förbearbetning av scenen. Dessa egenskaper sammantaget gör radiositymetoderna mindre lämpliga för större scener med komplex geometri.

Även Radiosity har man försökt effektivisera för interaktiva hastigheter, se t.ex. [17, 18, 19, 22].

3.2.2.3 Hybridmetoder

Då raytracing- och radiositymetoder är bra på spekulära respektive diffusa reflektioner, är det naturligt att försöka kombinera de bägge för att utnyttja det bästa av de två angreppssätten. Ofta görs detta i multipassmetoder där den ena varianten, t.ex. raytracing, körs först och resultatet kompletteras med lösningar från den andra, t.ex. radiosity. Detta fungerar rätt bra då ren raytracing är bra på att fånga upp detaljer i scenen men är sämre på långsammare förändringar från t.ex. indirekt belysning, något som är den rena radiositymetodens styrka. Dessa metoder faller dock på att radiosity inte klarar komplexa modeller. Man har försökt lösa detta genom att använda förenklad geometri i radiositypasset då raytracingpasset tar hand om alla detaljer. Denna lösning kräver dock manuell anpassning av scenen då automatiska verktyg saknas för icke-triviala geometrier. Ett annat problem är att felen man inför med geometriförenklingen är svåra att analysera och förutse.

3.2.2.4 Photon Mapping

Photon Mapping [3] är en modern state-of-the-art metod som är lika mycket en insiktsfull renderingsmetod, som en smart datologisk implementationsansats. Den är lika generell som en ren Monte Carlo raytracer men betydligt effektivare, då den utnyttjar en rad optimeringar. Idén bakom Photon Mapping är att man ändrar hur man representerar belysning i scenen. Istället för att lagra belysningsinformationen starkt knutet till den underliggande geometrin, lagrar man informationen i en separat, helt oberoende datastruktur som kallas för en *photon map*. Detta ger möjligheten att beräkna belysningen med hjälp av densitetsestimering som ger mycket effektivare beräkningar än med en traditionell Monte Carlo raytracer. Dessutom blir effekten av densitetsestimeringen att det resulterande bruset i bilden blir spatiellt lågfrekvent istället för det högfrekventa, traditionella, Monte Carlo bruset. Detta är en fördel då en mänsklig betraktare uppfattar högfrekvent brus som mycket mer störande än lågfrekvent brus. Nackdelen med denna metod är att den är *biased*, dvs. väntevärdet av metoden skiljer sig mot det riktiga svaret. Den är däremot *konsistent* vilket innebär att avvikelsen mellan väntevärdet och det sanna resultatet minskar med ökande antal sampel/fotoner vilket gör biased-problemet oviktigt i praktiska sammanhang.

Metoden är uppbyggd i två steg:

- *Photon Tracing*: I detta steg byggs "photon map"-strukturen upp genom att följa fotonernas väg från ljuskällorna ut i scenen. Detta görs på samma sätt som med en effektiv Monte Carlo raytracer; skillnaden är att i en raytracer propageras *radiance* ($\text{Wm}^{-2}\text{sr}^{-1}$) ut i scenen medan en photon tracer propagerar effekt (W). Varje gång en "foton" reflekteras icke-spekulärt lagras träffen i "photon map"-strukturen som är en spatiell datastruktur, ofta ett balanserat kd-träd, oberoende av modellens geometri.
- *Rendering*: I detta steg utnyttjas "photon map"-strukturen av en Monte Carlo raytracer som sänder ut ett antal sampelstrålar genom varje pixel i kameran ut i scenen. Då en stråle träffar en yta estimeras radiansen i punkten genom att estimeras fotondensiteten i omgivningen med hjälp av "photon map"-strukturen. Då strukturen är oberoende av geometrin räcker det att några enstaka fotoner har lagrats i omgivningen av punkten för att ett hyfsat estimat av densiteten skall erhållas.

Metodens uppbyggnad gör att man lätt kan optimera den för speciella belysningseffekter. T.ex. klarar metoden av s.k. *caustics* väldigt effektivt, som är ett erkänt svårt ljusfenomen för raytracers. Caustics uppstår då ljus reflekteras i, eller transmitteras genom, en eller flera spekulära ytor för att till sist träffa en diffus yta. Ett exempel på när caustics uppstår är när en glasboll, liggande på en

grov träyta, belyses. Ljuset fokuseras genom bollen och formar en koncentrerad ljuspunkt, en caustic, på träytan. Vanliga raytracers kräver oerhört många fler sampel än Photon Mapping för att ge samma signal/brus-förhållande i ljuspunkten. Ett mer vardagligt exempel på caustics med Photon Mapping ser vi i Figur 2 där ett glas cognac bryter ljuset ner på en icke Lambertsk yta.

Många ansatser för att använda Photon Mapping i interaktiva hastigheter har gjorts, se t.ex. [15, 17, 18, 23].

3.2.2.5 Diffractionsextensioner

I inledningen av avsnitt 3.2 konstaterade vi att geometrisk optik oftast används som den underliggande fysikaliska modellen för hur ljus färdas i och mellan olika medier. Denna modell klarar inte av att beskriva vågfenomen såsom diffraction, men naturligtvis kan metoderna vi diskuterade i avsnittet 3.1, t.ex. Geometric Diffraction Theory (GTD), användas även i datorgrafik-sammanhang. Se t.ex. [1, 24, 25, 26, 27] för några diffraktionsmetoder.

3.3 Datorgrafik för Radarsimulering

Efter genomgången av de olika beräkningsmetoderna ovan står det klart att det finns ett stort överlapp mellan metoderna som används inom datorgrafiken respektive radarmålareaberi-
beräkning. Speciellt högfrekvensmetoderna GO och PO kan klart dra nytta av framstegen inom Monte Carlo raytracing, t.ex. Photon Mapping, då de ofta utnyttjar strålföljning i sina implementationer. Då metoderna kommer från olika forskningskulturer, radarmålareaforskare är ofta fysiker och datorgrafikforskare är ofta dataloger, måste man vara noggrann med använda begrepp och algoritmer då dessa kan vara bedrägligt lika/olika mellan fälten.

I denna rapport är vi speciellt intresserade av beräkningsmetoder för radarmålarea som kan användas i interaktiva hastigheter med ”hyfsad” kvalitet. Efter att ha gått igenom de olika metoderna i de föregående avsnitten verkar det troligt att exempelvis PO-metoden implementerad med Photon Mapping skulle uppfylla många av våra önskemål:

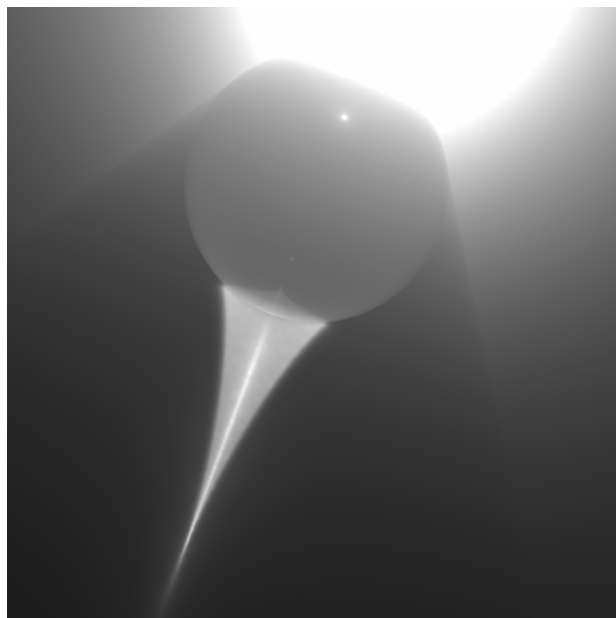
- **Hastigheten:** Photon Mapping är en väldigt effektiv metod för att lösa det globala belysningsproblemet med komplexa modeller. Dessutom är Photon Mapping enkel att parallellisera då den i grunden är en Monte Carlo raytracer vilket gör att varje stråle kan beräknas oberoende av alla andra strålar. Speciellt kan den implementeras i en s.k. *streaming processor* [28, 29] som är en väldigt effektiv beräkningsparadigm för oberoende data som skall behandlas på likartat sätt. Som vi skall se i nästa kapitel är grafikkorten på väg att utvecklas i en rasande fart till fullfjädrade och programmerbara sådana streaming processorer. Faktum är att Photon Mapping, och därmed Raytracing, redan har implementerats på förra(!) generationens grafikkort för en väldigt enkel scen [30, 31]. En framerate på ca. 0.25 Hz uppnåddes då. Med nuvarande utveckling uppskattar vi att grafikkorten och tillhörande programmeringsmetoder utvecklas så att de klarar Photon Mapping i realtid för komplexa scener inom fem år.
- **Kvaliteten:** PO är en beprövad högfrekvensmetod där resultaten lätt kan verifieras. Om vi implementerar PO med hjälp av Photon Mapping kan vi lätt även kontrollera avvägningen mellan prestanda och kvalitet då färre utsända fotoner eller insamlade strålar resulterar i ett lågfrekvent spatiellt brus över bilden utan att för den skull påverka den grundläggande karaktären hos bilden. Då kvaliteten på detta sätt försämras på ett kontinuerligt, långsamt och predikterbart vis, säger man att metoden uppvisar ”gracefull degradation” vilket är en önskad egenskap. Implementationen med Photon Mapping garanterar också att vi kan hantera väldigt generella typer av reflektioner och ytinteraktioner (se nedan). Mer avancerade ljusfenomen såsom interferens, koherens och polarisation är även möjliga.
- **Flexibilitet och Kapacitet:** Om vi implementerar PO med Photon Mapping behöver vi inte förbehandla våra modeller, och vi blir oberoende av hur vi representerar den underliggande

geometrin vilket är en stor fördel. Olika objekt i scenen kan vara representerade på olika sätt till och med; Man kan t.ex. tänka sig polygoner, punktmängder, analytiska och procedurella geometrier. Vidare tillåter Photon Mapping väldigt avancerade reflektioner. T.ex. har man gjort simuleringar av scener med halvgenomskinliga material, såsom marmorstatyer, där ljus går ner genom ytan i en punkt men reflektionen lämnar ytan i en annan punkt, s.k. *subsurface scattering*. Detta är ett specialfall av s.k. *participating media*, som t.ex. partikelfylld luft (rök, moln & dimma) och grumligt vatten (mjölk). Photon Mapping kan hantera även dessa svåra fall och är den första metoden som effektivt har klarat s.k. *volume caustics*. Se Figur 3 nedan.

I nästa kapitel kommer vi att undersöka om vi med hjälp av moderna grafikkort kan implementera Photon Mapping effektivt i hårdvara, eller om endast enklare metoder tillåts idag. Sammanfattningsvis vill vi rekommendera Photon Mapping för övervägande i eventuella framtida implementeringar av t.ex. PO-metoder för radarsimuleringar.



*Figur 2: Ett vardagligt exempel på caustics renderad med Photon Mapping.
Bilden är gjord av Henrik Wann Jensen [3]*



*Figur 3: Ett exempel på "volume caustics in participating media" renderad med Photon Mapping.
Vi ser hur en glassfär i dimma fokuserar ljuset. Bilden är gjord av Henrik Wann Jensen [3]*

4 Utvecklingen av konsumentgrafikkort

4.1 Historik

Utvecklingen av billiga konsumentgrafikkort [6, 32, 33] har under de senaste åren varit explosionsartad. 1987 introducerades VGA-kort ("Video Graphics Array") till persondatorer av IBM. Detta kallar vi idag för en "dum frame-buffer" därför att kortets enda uppgift var att lagra och visa en uppsättning pixlar på en skärm. All hantering och uppdatering av pixlarna sköttes av CPU:n. Gradvis, och till en början relativt långsamt, började konsumentgrafikkorten få fler funktioner och vissa fönsterhanteringsfunktioner i operativsystemen blev hårdvaruaccelererade. Under denna tid, dvs. början av 1990-talet, hanterades all grafik nästan uteslutande i mjukvara. Detta ledde till att grafikapplikationerna, som mestadels var datorspel, såg ganska olika ut med varierande kvalitet, eftersom det var kompetensen och uppfinningsrikedomen hos den enskilde programmeraren som avgjorde slutresultatet.

I slutet av 90-talet (→ 1998) började utvecklingen av hårdvara för datorgrafik i 3D ta fart ordentligt. Detta gick hand i hand med utvecklingen av programmeringsgränssnitten för 3D-grafik där de vanligaste är DirectX [34], skapad av Microsoft, och OpenGL [35, 36] som är en öppen standard. Programmeringsgränssnitten specificerar klart hur den s.k. renderingskedjan ser ut, dvs. hur skapandet av en bild går till och vilket ansvar programmeraren respektive grafikhårdvaran har. Uppdateringen och den s.k. rasteringen av pixlarna flyttades till grafikkorten och CPU:n avlastades. Detta var den första generationen av GPU:er (Graphic Processors). Se även Figur 4.



Figur 4: Exempelbilder från två olika GPU generationer. Gen 1 & 2 till vänster respektive höger.

Fortfarande var dock CPU:n ansvarig för geometrihanteringen, s.k. vertexhantering, och de första GPU:erna hade väldigt begränsad och fix texturhantering. Detta ändrades i och med introduktionen av andra generationens GPU:er (1999-2000) som inkluderade s.k. T&L (Transformation and Lighting) där CPU:n avlastades nästan totalt från all vertexhantering. Med denna generation försvann därmed en av de stora skillnaderna mellan konsument- och professionell grafikhårdvara. Textur- och pixelhanteringen i denna generation var också mer flexibel och kapabel än den första. Se Figur 4.

I detta skede hade grafikprogrammerarna tillgång till en renderingskedja effektivt implementerad i hårdvara och där CPU:n var avlastad och kunde utnyttjas till andra uppgifter. Detta gav upphov till stora grafiksprång i 3D-applikationerna, främst spel, med möjligheter till en helt ny grad av detaljrikedom. Nackdelen med de nya GPU:erna var att möjligheten att förändra och konfigurera renderingskedjan var starkt begränsad. Detta gav en väldigt likriktning av applikationerna och spelen, allt såg "likadant" ut. Den tredje generationen av GPU:er (2001) införde därför programmerbarhet, där framför allt vertexhanteringen kunde programmeras. Även flexibiliteten hos pixelhanteringen utökades men kunde ännu inte kallas för programmerbar. Den fjärde, nuvarande, generationen av GPU:er (2002→?) erbjuder programmerbarhet för både vertex- och pixelhantering, där allt kraftfullare språkkonstruktioner tillåts för varje ny modell. Se även Figur 5.



Figur 5: Exempelbilder från två olika GPU generationer. Gen 3 & 4 till vänster respektive höger. Bilderna är skapade av Nvidia.

Utvecklingen av grafikkortens möjligheter och programmerbarhet är imponerande, men än mer imponerande är prestandaökningen i varje generation. Ett sätt att mäta prestanda hos grafikkort är att se hur många trianglar (vanlig grafikprimitiv) kortet kan rita per sekund. 1998 klarade de bästa konsumentgrafikkorten att rita ca. 6 miljoner trianglar per sekund. 2004 låg den siffran på 600 miljoner trianglar per sekund. Detta är en faktor 100 i prestandaökning på bara 6 år vilket innebär en faktor på 2.15 per år, en dryg fördubbling per år! Ett annat vanligt mått på ett grafikkorts prestanda är hur många pixlar av en viss kvalitet kortet klarar av att rita per sekund. 1998 var den siffran ca. 50 miljoner pixlar/s och 2004 var siffran uppe i 6400 miljoner pixlar per sekund. Detta innebär en faktor 128 i prestandaökning på bara 6 år! Se Tabell 1 nedan för utvecklingen av Nvidia grafikkort mellan 1998 och 2004. Man skall lägga märke till att priset för dessa kort har legat någorlunda konstant under denna utveckling.

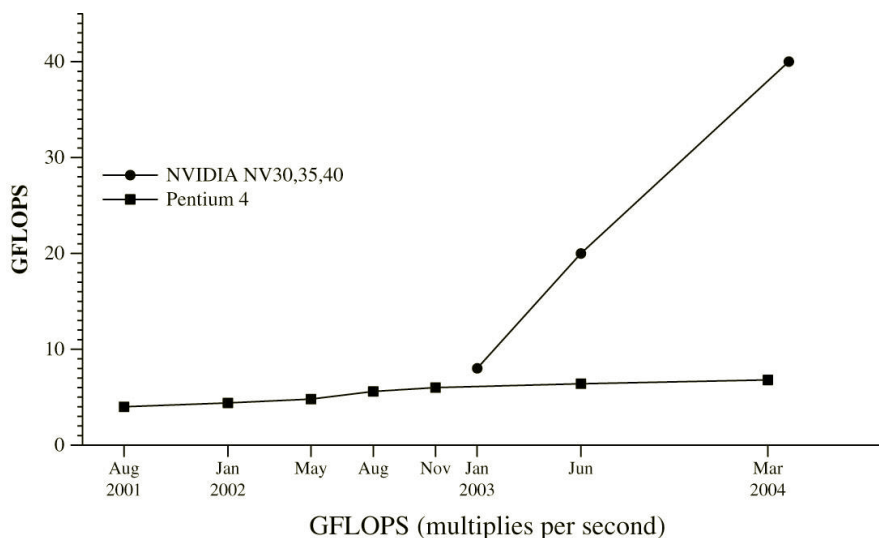
Generation	År	Produkt-exempel	Process	Transistorer	Fill Rate	Polygon Rate
1	Sent 1998	Riva TNT	0.25 μ	7 M	50 M	6 M
1	Tidigt 1999	Riva TNT2	0.22 μ	9 M	75 M	9 M
2	Sent 1999	GeForce 256	0.22 μ	23 M	120 M	15 M
2	Tidigt 2000	GeForce2	0.18 μ	25 M	200 M	25 M
3	Tidigt 2001	GeForce3	0.15 μ	57 M	800 M	30 M
3	Tidigt 2002	GeForce4 Ti	0.15 μ	63 M	1200 M	60 M
4	Tidigt 2003	GeForce FX	0.13 μ	125 M	2000 M	200 M
4	Sent 2004	GeForce 6800U	0.13/0.11 μ	222 M	6400 M	600 M

Tabell 1: Exempel på utvecklingen av grafikkort 1998-2004 [6].

Vilka drivkrafter ligger då bakom denna otroliga utveckling? Man brukar peka på följande tre faktorer för att förklara "grafikrevolutionen":

- Moore's lag: Under en lång tid har halvledarindustrin var artonde månad lyckats fördubbla antalet transistorer som får plats på en given yta. Detta har givit oss ständigt snabbare och/eller billigare CPUer samt givit hårdvarukonstruktörerna en till synes osinlig källa av nya resurser att använda i sina arkitekturer.

- Datorgrafik är väldigt enkel att parallellisera: Man brukar säga att datorgrafik är "embarrassingly parallel" eftersom man väldigt lätt kan dela upp problemet i oberoende delproblem. Detta är speciellt sant för den lokala belysningsmodellen där varje ytfragment kan behandlas oberoende och parallellt, men gäller även för raytracing där varje stråle kan behandlas separat. Varje delproblem skall dessutom genomgå samma beräkningssteg vilket kraftigt minskar behovet av "overhead"-hårdvara för kontroll och övervakning. Detta innebär att man kan utnyttja Moore's lag effektivt och använda en stor andel av transistorerna till "nyttigt" beräkningsarbete. För CPUer går detta inte då dess arbetsuppgifter är av mycket mer heterogen och seriell natur. Resultatet är att prestanda ökar mycket fortare för GPUer än för CPUer. Detta är illustrerat i Figur 6. Svårigheterna att utnyttja transistorerna i CPUer kan också illustreras med att en Pentium 4 2004 har ca. 125 miljoner transistorer (beroende på modell) medan den senaste generationen GPUer har ca. 222 miljoner transistorer.



Figur 6: Utvecklingen av beräkningskapacitet för en vanlig processor och konsumentgrafikkort (Bilden är gjord av Ian Buck, Stanford University)

- Människans behov av underhållning: Datorspelsindustrin har utvecklats enormt de senaste åren och är den ekonomiska kraft som sammanbinder de ovanstående faktorerna och ger incitament för den ständiga vidareutvecklingen. De moderna stora spelen har budgetar och omsättningar som är i samma klass som Hollywoods största filmer.

Om Moore's lag fortsätter att hålla i 10 år till och om vi kan fortsätta att utnyttja alla nya transistorer lika effektivt som hittills, kan vi 2014 förvänta oss ett billigt konsumentgrafikkort vars prestanda ligger runt 10 TFlops. Detta är i samma prestandaklass som världens snabbaste superdator (2004) "Earth Simulator Center", som upptar samma yta som fyra tennisplaner över 3 våningsplan! [37] Denna extrapolation är naturligtvis högst osäker men ger perspektiv över utvecklingen de senaste tio åren, och en uppfattning om den oerhörda framtida potentialen.

Vad kan vi då vänta oss av den femte GPU generationen? Om man tittar på vad som presenteras i nyutkomna böcker, journaler och på konferenser, så går trenden mot streaming processors och en global belysningsmodell [29, 30, 31, 37, 53, 54, 38, 39, 40, 41, 42, 43]. Just nu (februari 2005) är vi i en övergångsperiod då globala belysningsfenomen simuleras med multipla pass med den nuvarande programmerbara rasteringskedjan. Under 2005 - 2007 kommer vi att få se en övergång till en allt mer komplett streaming processor som stödjer generell strömprogrammering fullt ut. På en sådan kan mycket mer än bara grafik beräknas, t.ex. ljud och fysik, vilket vi kommer att diskutera i nästa sektion. Vidare kommer vi troligtvis också få se starten av en helt ny grafikrenderingskedja som bygger på raytracing. Den gamla renderingskedjan kommer dock av bakåtkompatibilitetsskäl stödjas i många år framöver.

Framgången för GPU:er i grafiksammanhang har lett till att man försöker göra motsvarande hårdvaruimplementationer av andra krävande tillämpningar, t.ex. fysiksimulering [44]. Troligtvis kommer dock nästa generations GPU:er vara så flexibla och kraftfulla att dessa nya produkter kommer att få det svårt att klara sig i konkurrensen.

4.2 Allmänna beräkningar på grafikkort

Som vi har sett har vi haft en stor utveckling av både programmerbarhet och prestanda de senaste åren. Detta har haft till följd att allt fler forskare och ingenjörer har börjat använda dessa billiga grafikkort för problemlösning och simulering inom vitt skilda fält, en del inte ens grafikrelaterade. Detta angreppssätt har fått namnet *GPGPU*, ”*General-Purpose computation on GPUs*”. Många internetsidor, artiklar och böcker har börjat publiceras, speciella workshops har organiserats och universitetskurser har startats [45, 46, 47, 48, 49, 50]. En bra www-sida som innehåller mycket GPGPU-information är [45]. Exempel på dessa nya tillämpningar är:

- Avancerad rendering med globala ljusmodeller och bildbaserad modellering
- Cellulära automater och flödesdynamik
- Beräkningsgeometri
- Akustikberäkningar och allmän ljudbehandling
- Signal- och bildbehandling. Datorseende
- Artificiell Intelligens
- Vetenskapliga beräkningar av olika slag, t.ex. lösning av PDEs, linjära ekvationssystem, olika former av fysikalisk simulering och modellering
- Studier av s.k. *Stream Processors*
- Databassökningar
- Monte Carlo metoder

Idag är det ganska krångligt att använda grafikkorten till annat än just grafik därför att programmeringsgränssnitten (API) förutsätter detta. Man måste därför representera sitt problem i datorgrafiska termer samt så att det passar renderingskedjan. Detta är inte alltid så enkelt och utgör därför en hög tröskel för de flesta. Därför förväntar sig de flesta att framtidens APIer kommer att abstrahera grafikkorten som en s.k. *streaming processor* (SP) [28, 29], och erbjuda stöd för programmering mot en sådan. I en SP representeras data som strömmar av oberoende element av samma datatyp. På dessa strömmar opererar s.k. *kernels*, beräkningskärnor, som gör samma beräkningar på alla de inflödande elementen och representerar resultaten som en eller flera utgående strömmar. Beräkningskärnorna kan sedan kopplas i kaskad på ett naturligt sätt. Denna beräkningsmodell passar grafikkorten väldigt bra då de i princip redan är uppbyggda på detta sätt. Jämför man t.ex. mot en CPU som måste kunna hantera en seriell ström av ej oberoende instruktioner på heterogena data så inser man att en SP kan göras betydligt effektivare. I detta ligger också en av de stora framtida utmaningarna. Man vill göra grafikkorten mer programmerbara och generella så att de klarar fler typer av problemområden, men samtidigt måste man vara försiktig så att man inte förlorar anledningen till att utvecklingen går så fort som den har gjort, nämligen parallelliserbarheten i grafikproblemet. Håller man sig till SP-modellen är detta dock inget problem eftersom den förutsätter parallelliserbarhet.

Ett exempel på ett programmeringshjälpmedel för allmänna beräkningar på en GPU är *BrookGPU*[47]. Detta är ett C-liknande språk med extensioner för att stödja dataparallellism och s.k. *aritmetisk intensitet*, vilket innebär att man i sina beräkningskärnor försöker minimera global kommunikation samt maximera lokaliserade beräkningar för att göra kärnan så effektiv som

möjligt. Ett annat alternativ är *Sh* [48], som tillåter användaren att skriva program för GPU:er i vanlig C++. En snabb utveckling för BrookGPU, *Sh* och liknande verktyg förväntas de närmaste åren, vilket kommer att göra beräkningskapaciteten i GPU:er tillgängliga för många fler.

4.3 Radarberäkningar på grafikkort

I kapitel 3 kom vi fram till att simuleringar med photon mapping har en rad önskvärda egenskaper, och vi vill därför kunna implementera algoritmen på en GPU för att kunna dra nytta av GPU:ernas oerhörda utveckling. Purcell har i sin doktorsavhandling "Ray Tracing on a Stream Processor" [29] undersökt just detta och han har visat följande:

- Hur man effektivt kan implementera ray tracing och photon mapping på en allmän streaming processor (SP).
- Hur en modern (2003-2004) GPU kan användas som en SP. I försöksimplementationen av en strålföljare visar han att prestanda på en GPU motsvarar de snabbast kända algoritmerna på en CPU. Han redovisar också varför han tror att prestanda hos GPU implementeringar kommer att öka snabbare än motsvarande CPU implementeringar.
- Vilka begränsningar som finns hos moderna (2003-2004) GPU:er och hur begränsningarna hindrar GPU:erna från att vara fullfjädrade och effektiva SP. Han ger också förslag på hur man kan undvika begränsningarna i framtida GPU:er. Begränsningarna är/var:
 1. Svårigheter att göra databeroende beräkningar: Kontrollinstruktioner såsom "if" och "for" för villkorad respektive repeterad exekvering saknas eller är bristfälligt implementerade
 2. Support för "Scatter"-operationer saknas, dvs. möjligheten att skriva på valfri plats i ett minne.
 3. Support för multipla utvärden per pixel saknas
 4. Support för heltalsaritmetik saknas.
 5. Support för adressering av stora 1D arrayer saknas.
 6. Support för effektiv mellanlagring av delresultat saknas.
 7. Utläsning av data från GPU:n till CPU:n för långsam.
 8. Verktyg såsom debuggers och profiler, för generell programmering på GPU:er saknas nästan helt.

Sammanfattningsvis menar Purcell att vi är mycket nära ett genombrott för GPU:n som en parallell co-processor och att interaktiv ray tracing i högsta grad är möjlig inom en snar framtid. Det är här intressant att notera att de senaste (2005) modellerna av GPU:er redan har överkommit många av begränsningarna (t.ex. 1,3,6,7 ovan) som Purcell nämner, vilket han även förutspår i sin avhandling. Detta har resulterat i att implementering av strålföljning på GPU:er har lämnat state-of-the-art forskningsfronten och kan nu göras på t.ex. teknologinivå av examensarbetare. Två exempel på sådana projekt är [51, 52].

Vi kan alltså konstatera att det redan idag är fullt möjligt att implementera photon mapping på GPU:er, men att riktigt effektiva implementationer, som t.ex. tillåter komplexa scener i interaktiva hastigheter, blir möjliga först om ett par år (2006-2007). Vi kan också konstatera att det finns många som arbetar och strävar mot detta mål vilket gör att vi tror att avsaknaden av debuggers och andra nödvändiga utvecklingsverktyg kommer vara avhjälpt inom kort. Slutsatsen blir därför att vi rekommenderar en satsning på radarsimulering på GPU:er med start under 2006-2007 då vi tror att hård- och mjukvara kommer att vara tillräckligt mogen.

Ett tecken på att övergången från lokala till globala belysningsmodeller redan har startat, är att facklitteraturen för interaktiv grafikprogrammering allt mer har börjat ta upp fenomen såsom

indirekt belysning [33]. Denna nya trend och de nya algoritmer som har börjat komma [53, 54] är verkligen högintressanta för radarsimulering då de i övergångsperioden till en fullödig global belysningsmodell erbjuder en semilokal variant som kan vara fullt tillräcklig för vissa applikationer.

Eftersom radarsimuleringsexperimenten i detta projekt utfördes under våren 2004 då grafikhårdvaran saknade en del viktiga funktioner, beslöt vi oss för att inte försöka implementera någon global belysningsmodell utan istället undersöka vad vi kunde åstadkomma med en avancerad, men lokal, belysningsmodell. Detta kommer att avhandlas i nästa kapitel.

5 Simulering av radar med programmerbara konsumentgrafikkort

För att påvisa de nya programmerbara grafikkortens användbarhet för mer generella beräkningar och simuleringar än den datorgrafik de primärt är konstruerade för, valde vi att undersöka hur avancerad radarsimuleringsmetod som är möjlig att implementera med dem. Vi kommer i detta avsnitt beskriva den implementering vi gjort och försöka belysa vad grafikhårdvaran har för funktionalitet som kan användas för att erhålla snabba och effektiva beräkningar. Vi kommer vidare att belysa vilka brister som finns i dagens kort och vad det ger för begränsningar vid implementationen av de simuleringar vi vill göra. Vi vill dock redan nu påpeka att den generation kort vi använt, redan har ersatts med en ny, där flera av dessa begränsningar avlägsnats.

5.1 Använda programvaror och programpaket

En mycket ansenlig del av den programmering som krävs för att genomföra simuleringar på grafiska 3D-objekt, oavsett om det är radar eller vanlig datorgrafik som avses, kräver en omfattande hantering av de geometriska beskrivningar av scenen som skall beräknas. Förutom det krävs programvara som förenklar hanteringen och programmeringen av grafikkorten. De följande avsnitten ger en kort beskrivning av de hjälpprogramvaror som använts i detta arbete.

5.1.1 OpenGL

OpenGL [35, 36] är ett programmerings API för hantering av tredimensionell datorgrafik. OpenGL har dock en mycket låg abstraktionsgrad vad gäller de geometriska beskrivningar den arbetar med. Den arbetar med enskilda ytsegment, såsom polygoner, fyrhörningar och liknande geometriska primitiver. Programpaketet är dock från början framtaget att hantera den mest grundläggande grafikhanteringen, dvs. hur dessa primitiver ska hanteras för att de ska kunna ritas upp på skärmen som bildpunkter. På senare tid har dock större delen av detta arbete tagits över av hårdvara på grafikkorten. OpenGL har därför blivit mer av ett standardiserat kommunikationsgränssnitt mellan applikationen och grafikkort från olika tillverkare. Värt att nämnas är att Microsoft har tagit fram en konkurrerande standard till OpenGL, som kallas DirectX3D [34]. Vi ska inte här diskutera skillnaderna mellan dessa standarder, utan bara konstaterar att de för detta ändamål är i stort sett likvärdiga. Valet av OpenGL har styrts av valet av OpenSceneGraph [56] som scenhanteringsprogramvara, se avsnitt 5.1.2.

5.1.2 OpenSceneGraph, OpenProducer och osgNV

Den scen som ska användas i simuleringen måste hanteras i form av en datastruktur som kallas en scenograf [55], se avsnitt 5.2. De olika objekt som ingår i scenen måste laddas från filer med varierande format, olika material och texturer måste hanteras och slutligen konverteras till kommandon och primitiver som kan hanteras av OpenGL. Vidare måste fönsterhantering, mus och tangentbord, m.m. hanteras mot operativsystemet. För dessa ändamål har programvarorna OpenSceneGraph [56] och OpenProducer [57] använts. Dessa är s.k. "open source"-projekt och kan användas utan kostnad för icke kommersiellt bruk.

OpenProducer hanterar fönster i Windows och den virtuella kamera som används som modell för att konfigurera OpenGLs projektion av scenen till en tvådimensionell bild av scenen. OpenProducer används i kombination med OpenSceneGraph för att hantera scenen.

OpenSceneGraph är ett stort programvarupaket med många funktioner och komponenter för att hantera den geometriska beskrivningen av scenen. Förutom det görs hela konfigurationen av

OpenGL med hjälp av OpenSceneGraph. Detta görs genom att i scengrafens olika noder påverka vad som kallas OpenGLs tillstånd (eng. "state").

osgNV [58] är ett tilläggs paket till OpenSceneGraph, som möjliggör användandet av Nvidias programmeringsspråk till grafikkorten vilket kallas Cg, se avsnitt 5.1.3.

5.1.3 Cg från Nvidia

Cg [6] ("C for Graphics") är ett programmeringsspråk som grafikkretstillverkaren Nvidia tagit fram för att programmera moderna grafikkort. Det är detta programmeringsspråk, med tillhörande verktyg, som möjliggjort det arbete som denna rapport beskriver. Cg ger en abstraktionsnivå på programmeringen av grafikkortens primära processeringssteg, som gör det möjligt att konstruera komplicerade funktioner på ett enklare sätt än tidigare. Programmeringsspråket påminner mycket i sin struktur om C, men med t.ex. fyrdimensionella vektorer och matriser tillgängliga som datatyper.

5.2 Scengrafer

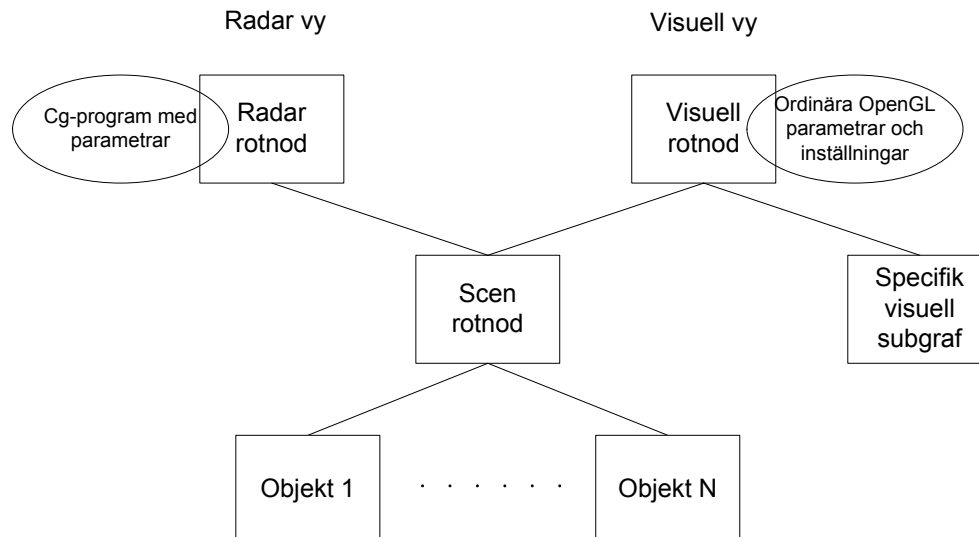
Beskrivningen av den geometri som ligger till grund för datorgrafiksimuleringar, brukar organiseras i en datastruktur som kallas scengrafer [55]. Dessa är en datalogisk struktur där alla objekt kopplas till varandra i en hierarkisk icke-cyklisk graf. Grafen innehåller de olika grafiska primitiverna i grafens lövnoder (polygoner, linjer etc.), medan övriga noder innehåller transformationer, switchar eller tillståndsförändringar av olika slag. En betydande del av den programkod som den framtagna applikationen består av, manipulerar scengrafen så att önskad funktion erhålls.

Scengrafen används till att placera ut objekt i relation till varandra. Detta görs med hjälp av homogena transformationer av typen skalning rotationer och translationer. Dessa kodas som matrisoperationer i homogena koordinater. Fördelen med detta är att konsekutiva operationer enkelt kan appliceras med en serie matrismultiplikationer, vilka sammanställs genom traversering av scengrafen.

5.2.1 Använd scengrafstruktur

I den applikation som implementerats i detta arbete, har den använda toppstrukturen på scengrafen getts en något speciell struktur. Syftet med denna struktur är att kunna ha olika rotnoder med olika OpenGL-tillstånd beroende på om scenen ska renderas visuellt eller om det experimentella Cg programmet ska användas. Observera att ett Cg-program, med dess parametrar och variabelbindningar, betraktas som ett OpenGL-tillstånd, på samma sätt som t.ex. en projektionsmatris eller en konfiguration av en stensilbuffert.

Normalt används en rotnod i en scengraf och olika objekt läggs in under denna i en hierarki. Det vanliga är att Cg-program används lokalt i grafen, dvs. på ett enskilt objekts subgraf, för att implementera en speciell effekt, ytstruktur, material, etc. I detta fall måste grafens struktur garantera att endast det Cg-program vi implementerat används på alla objekt i grafen, eftersom vi vill ha med alla objekt i vår radarsimulering, se Figur 7. Därför måste programmet läggas i rotnoden, och konfigureras, så att detta tillstånd inte får ersättas utan bara kompletteras med andra tillstånd längre ner i grafen. Vidare finns subgrafer som endast ska visas i den visuella simuleringen, så som en kommandorad där användaren kan skriva in kommandon för att kontrollera simuleringen. Denna måste således hållas avskild från radarrenderingen.



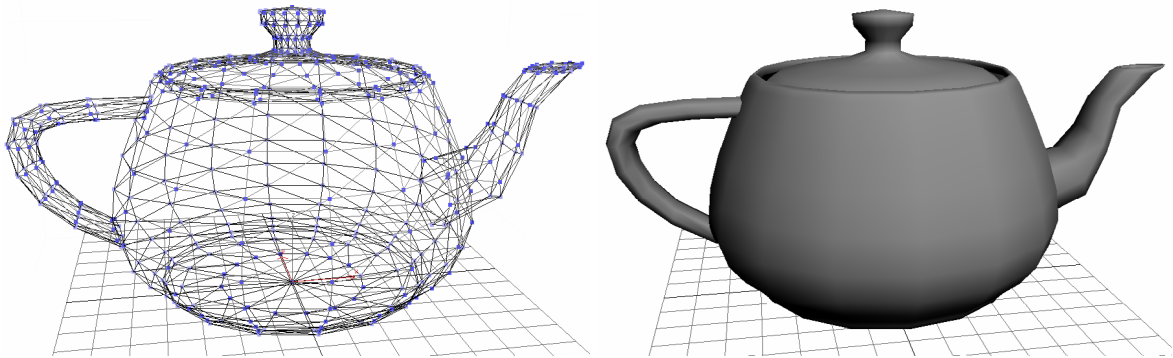
Figur 7: Använd grundstruktur för radarsimuleringsimplementeringen.

5.3 Fysikalisk modell för simuleringen

Den fysikalisk modell som ligger till grund för den implementering som gjorts i denna studie, baseras på geometrisk (GO) och fysikalisk optik (PO), se avsnitt 3.1. Den geometriska behandling av scenen, som är en mycket beräkningskrävande del i GO, hanteras i denna implementering av hårdvaran på grafikkortet. Den begränsning man då får med dagens hårdvara är att man bara kan hantera enkelstudsar, dvs. strålgången sändare – objekt – mottagare. Det är dock möjligt att hantera bistatiska och multistatiska beräkningar, dvs. man kan ha fler än en sändare och de kan vara placerade var som helst. Eventuella direktbidrag måste dock hanteras separat, vilket i praktiken inte är något problem. Man kan med hjälp av scengrafien hantera ett fåtal ”line of sight” beräkningar och på så vis även få med effekten av skymda sändare om så önskas. I denna implementering hanteras ej skuggfenomen mellan objekt i scenen, varför programmet i nuvarande version endast är lämpad för monostatiska beräkningar. I avsnitt 5.3.3 beskrivs dock en enkel metod för att lägga till hantering av skuggor om man önskar utveckla programmet ytterligare.

5.3.1 Transformer och sampling av scenen

I de moderna grafikkorten hanteras scengeometrin i huvudsak i två behandlingssteg. Det första steget hanterar geometrins fundamentala byggstenar, dvs. vertexpunkterna. Vertexarna definierar hörnen eller brytpunkterna i geometrins polygoner och är i egentlig mening de enda punkterna man kan vara säker på att geometrins stämmer överens med den geometri man vill att modellen skall efterlikna, se Figur 8. I vertexpunkterna definieras även ytans normalriktning, material, texturkoordinater m.m.



Figur 8: Geometrin i datormodellerna byggs upp av sampelpunkter, kallade vertexpunkter. Dessa binds sedan samman till polygoner och ytor för att bilda en modell av objektet.

5.3.1.1 Vertexshader

Grafikkorten har en särskild programmerbar beräkningsenhet, "vertexshader", som endast hanterar beräkningar på vertexpunkterna. Huvudsyftet med dessa beräkningar är att transformera objektens vertexpunkter till deras korrekta plats i scenen sett från sensorn, samt att eventuellt beräkna belysningen på ytan i punkterna. Transformationen till vykoordinater sker med hjälp av den totala transformationsmatris som definierats av scengrafens struktur, se avsnitt 5.2.

För de syften som varit primära i detta projekt, har dock denna del av hårdvaran begränsad användbarhet. Därför utförs endast de nödvändiga geometritransformationerna av position, normal och texturkoordinater, samt vidarebefordring av färg m.m. i denna del av processkedjan.

5.3.1.2 Fragmentshader

En av de mest betydelsefulla delarna i beräkningen är den del i hårdvaran som utför själva samplingen av scenen och därigenom skapar det som kallas fragment. Ett fragment skall inte förväxlas med en pixel, dvs. en bildpunkt på skärmen, utan är en av många träffpunkter i scenen, längs den stråle som emanerar från fokuspunkten, går genom pixelns mittpunkt på skärmen och vidare ut genom scenen. Alla fragment som skapas kommer inte att skickas vidare i processkedjan. Beroende på inställningar i OpenGLs processkedja, kan vissa fragment termineras omedelbart, t.ex. genom användning av "stencilbuffers" vilka kan ses som ett digitalt maskeringsmönster.

Längs flertalet av strålarna/samplingsriktningarna, kommer dock flera fragment att behållas och skickas vidare för processning. Denna processning sker då i den programmerbara delen av hårdvaran som kallas "fragmentshader" och det är i denna del som huvuddelen av processningen i denna implementation sker, se avsnitt 5.3.2.

Efter "fragmentshadern" sorteras utdata för varje fragment i en djupbuffer. Beroende på hur hårdvaran konfigurerats, så kan fragmenten för respektive pixel hanteras lite olika. Det som är målet för denna implementering är att det fragment som är närmats mottagaren skall användas oförvanskat som värde för pixeln. Det innebär att ingen "alpha-blending" (används för transparenta ytor) eller "anti-aliasing" (används för att minska taggiga linjer och kanter i bilderna) skall användas.

5.3.2 Spridningsmodell

Under studiens gång har olika varianter på spridningsmodeller implementerats och provats. Spridningsmodellens uppgift är att givet den infallande strålens riktning, ytans orientering, yt- och materialegenskaper, beräkna den reflektion som erhålls i en viss betraktningsriktning. I datorgrafik-sammanhang kallas denna överföringsfunktion normalt för ytans BRDF, se avsnitt 3.2. Denna benämning har ej adopterats fullt ut i radarsammanhang, men termen används av och till med något

varierande formuleringar. Det som framförallt skiljer en BRDF för radar från en optisk, är behovet av att hantera polarisations- och fasvridningar.

De flesta av de spridningsmodeller som implementerats har varit "ofysikaliska" i den meningen att de baserats på den energispridningsmetodik som normalt används inom datorgrafiken och inte på de formuleringar som normalt används i radarsammanhang, t.ex. fysikalisk optik (PO). Anledningen till att olika modeller provats är att ambitionen med implementeringen varit att hantera även markspridning, vilket inte låter sig hanteras på ett bra sätt med fysikalisk optik. För detta ändamål experimenterades med andra sätt att på ett enkelt sätt beskriva returen från en yta med ett fåtal parametrar, på motsvarande sätt som exempelvis de modifierade formuleringar av Lamberts lag används i datorgrafiken. Det uppenbara problemet med detta är att ge parametrarna vettiga värden och avsaknaden av mätdata har medfört att dessa modeller ej kunnat verifieras på ett tillförlitligt sätt.

Vi fokuserar därför den fortsatta beskrivningen på PO. Ekvation (1) beskriver den formulering som normalt används för fysikalisk optik [59].

$$\sqrt{\sigma} = -i \frac{k |\bar{L}| |\bar{W}|}{\sqrt{\pi}} \hat{n} \cdot (\hat{e}_r \times \hat{h}_i) e^{ik\bar{r}_0(\hat{i}-\hat{s})} \frac{\sin\left(\frac{1}{2}k\bar{L} \cdot (\hat{i}-\hat{s})\right)}{\frac{1}{2}k\bar{L} \cdot (\hat{i}-\hat{s})} \frac{\sin\left(\frac{1}{2}k\bar{W} \cdot (\hat{i}-\hat{s})\right)}{\frac{1}{2}k\bar{W} \cdot (\hat{i}-\hat{s})} \quad (1)$$

Ekvationen beskriver det fält som erhålls i en observationsriktning, $\hat{s}$, då man integrerar de strömmar som induceras i en "oändligt stor" platta, givet det infallande fältet definierat av $\hat{h}_i$ och $\hat{i}$, se Figur 9a. Integralen beräknas dock sedan över en begränsad yta vars kanter, $\bar{L}$ och $\bar{W}$, definieras av avståndet mellan den aktuella strålen och dess grannar, se Figur 9b. Då samplingsmönstret på en datorskärm är rektangulärt och eftersom beräkningarna i fragmentprogrammet görs i vyrummet, dvs. med mottagaren i origo, blir uttrycken för $\bar{L}$ och $\bar{W}$

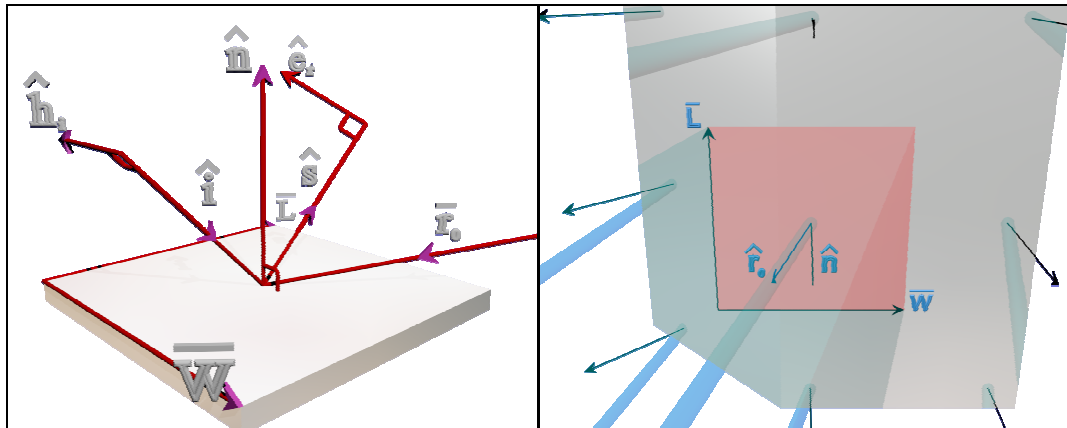
$$\bar{W} = \Delta x \begin{pmatrix} 1 \\ 0 \\ -\frac{n_x}{n_z} \end{pmatrix}, \quad \bar{L} = \Delta y \begin{pmatrix} 0 \\ 1 \\ -\frac{n_y}{n_z} \end{pmatrix} \quad (2)$$

där n_x , n_y , n_z är komponenterna i ytnormalen $\hat{n}$, Δx och Δy är avståndet mellan två pixlar vid plattan. Nu måste man dock vara lite försiktig. I grafiksammanhang kan man använda olika projektioner av scenen, beroende på vad man vill åstadkomma. De som är aktuella här är perspektivprojektion eller ortografisk projektion. Om ortografisk projektion används är det inga problem, den satta storleken på projektytan L_x , L_y , dividerat med det antal pixlar man har i sid- respektive höjdlid, N_x , respektive N_y , ger de korrekta värdena på Δx och Δy , dvs.

$$\Delta x = L_x / N_x, \quad \Delta y = L_y / N_y \quad (3)$$

Om istället perspektivprojektion används, måste divergensen mellan strålarna tas med i beräkningen. Det enklaste sättet att definiera detta är att ange en "field of view", FOV , för sensorn och dividera dessa vinklar med N_x , respektive N_y . Uttrycken för Δx och Δy blir då

$$\begin{aligned}\Delta x &= 2|\bar{r}_0| \tan\left(\frac{FOV_x}{N_x}\right) = 2r_z \tan\left(\frac{FOV_x}{N_x}\right) \\ \Delta y &= 2|\bar{r}_0| \tan\left(\frac{FOV_y}{N_y}\right) = 2r_z \tan\left(\frac{FOV_y}{N_y}\right)\end{aligned}\quad (4)$$



Figur 9: a) T.v. definieras de i ekvation (1) ingående storheterna. b) T.h. illustreras hur $\bar{L}$ och $\bar{W}$ definieras givet strålnippets infall mot ett objekt.

Beräkningen i fragmentprogrammet tar hänsyn till sändarens och mottagarens positioner i förhållande till fragmentet och beräknar det reflekterade fältet. Som nämnts ovan, hanteras ej skuggning av sändare i den nuvarande versionen, varför sändaren och mottagaren är samlokaliserade. Vid beräkningen har dock inga antaganden om detta gjorts, varför det är fullt möjligt att beräkna bistatiskt, förutsatt att man accepterar felen som kan uppstå pga. utelämnade skuggfenomen.

Indata till "fragmentshadern" kan till viss del styras av programmeraren. De mest väsentliga parametrarna är naturligtvis fragmentets position i förhållande till mottagaren, samt sändarens position. I detta program tas även hänsyn till sändarens lobmönster, vilket skickas in till fragmentprogrammet i form av en textur. I den nuvarande implementeringen hanteras dock endast en sändare per pass. Man kan dock utöka antalet sändarpositioner, texturer m.m. och på så vis minska denna begränsning en aning. Programmet hanterar dock multipla mottagare och multipla sändare genom upprepade beräkningspass.

Utdata från fragmentprogrammet är i den normala tillämpningen, datorgrafik, en fyrdimensionell vektor med ett färgvärde och en transparens (RGBA, röd, grön, blå och alfa), samt ett djupvärde, dvs. avståndet från sensorns projektiionsplan till fragmentet. I den generation hårdvara som använts i denna studie har detta varit den enda datamängd som varit möjlig att få ut från fragmentprogrammet per beräkningspass. Av dessa värden är djupet nödvändigt att ange på ett korrekt sätt och kan därför inte disponeras till andra data. Således har man fyra flyttal (plus djupet) till förfogande för att överföra framberäknad information från grafikhårdvaran tillbaka till huvudprogrammet.

5.3.3 Hantering av skuggfenomen

Då OpenGL egentligen endast hanterar lokal belysning, får man inte direkt med effekter av t.ex. skuggning. För radartillämpningen spelar detta mindre roll, så länge man bara är intresserad av monostatisk radar. Dock är det av intresse att hantera skuggfenomen så fort man är intresserad av bi- eller multistatisk radar. Det finns då flera olika metoder att använda för att med OpenGL implementera skuggfenomen. Den enklaste och mest rättframman metoden för denna tillämpning är att göra en multipassimplementering, där man för varje sändare renderar scenen med det synfält som sändarens lobmönster definierar och sparar på sig den resulterande djupbufferten. Varje värde

i djupbufferten representerar avståndet till det närmaste fragmentet i den specifika riktningen från sändaren.

Djupbufferten för varje sändare skickas sedan som en textur, tillsammans med data på sändaren, till fragmentprogrammet i huvudrenderingspasset. I fragmentprogrammet jämförs sedan varje nyskapat fragments avstånd till sändaren, med det värde som sparats i djuptexturen. Om avståndet till sändaren är större (med lite marginal) än värdet i djupbufferten, så innebär det att fragmentet är skuggat av någon geometridetalj som ligger närmare sändaren.

Detta kan låta som en enkel och självklar metod, men den har en del problem inbyggd som gör att den kan ge konstiga resultat i vissa scener och sändar-mottagargeometrier. Det antal pixlar som kan användas i genereringen av djupbuffertarna avgör tillsammans med sändarnas lobvidder, samt dess avstånd till de aktuella objekten, hur stora geometrier som tillåts skugga varandra. Man erhåller ett aliasingfenomen i kanterna på skuggorna.

5.4 Begränsningar & brister i dagens grafikhårdvara

Under arbetets gång har vi identifierat några begränsningar i den hårdvara vi använt, Nvidias GeForce FX 5950 Ultra, vilka har medfört att vi varit tvungna att kompromissa och införa ”smarta” lösningar. Hårdvaran var ”state of the art” under slutet av 2003 och början av 2004, men har nu ersatts av GeForce 6800 serien. I denna nya generation grafikkort har de flesta av dessa begränsningar arbetats bort, vilket är symptomatiskt för den utveckling som just nu sker på grafikkortsområdet. Följande begränsningar har identifierats:

- Begränsade möjligheter att leverera utdata från fragmentprogrammet. I den version av hårdvara vi använt, kan endast en vektor med fyra 32-bitars flyttal (RGBA) plus ett 32-bitars djupvärde (flyttal) per pixel och renderingspass skickas ut. Denna begränsning har dock kraftigt reducerats i och med introduktionen av ”Multiple Render Targets” (MRTs) i ”Pixel Shader 3.0”-specifikationen och den senaste versionen av hårdvaran, se avsnitt 4.6 i [60].
- Grafikbussen (AGP) har en asymmetrisk överföringshastighet. Med denna busslösning går det snabbt att skicka data till grafikkortet, men långsamt att läsa stora datamängder tillbaka till primärminnet. Denna begränsning är dock till stor del borta i och med introduktionen av PCI-Express, som introducerats i PC-datorer under hösten 2004. De moderna grafikkorten stöder PCI-Express.
- Djupvärdet kan ej läsas samtidigt med de fyra ”färgvärdena”, vilket innebär att två bildöverföringar från grafikkortet till datorns primärminne måste göras. Med MRTs kommer detta problem att accentueras, om man inte kan reducera datamängden genom multipassberäkningar på grafikkortet. PCI-Express lindrar dock detta problem till stor del.
- Ej full support för dynamisk branching, dvs. ”korrekt” hantering av loopar och villkorssatser. I den använda versionen av hårdvaran evaluerades alla instruktioner i programkoden, oavsett om de utgör en gren i en villkorssats som är aktuell eller ej. Detta medför många onödiga beräkningar i ett program med många villkorssatser. I och med ”Pixel Shader 3.0”-specifikationen är även denna begränsning eliminerad.

Förutom dessa begränsningar är självfallet en avsaknad av ray-tracing stöd, såsom strålhantering och möjligheten att introducera nya strålar i scenen, begränsande och gör det i praktiken omöjligt att hantera multipelstudsar. Denna begränsning kvarstår ännu. Om man dock analyserar de artiklar och dokument som beskriver vart grafikkortstillverkarna vill komma, ser man att denna funktionaliteten verkar finnas ”runt hörnet”. Detta skulle medföra ett revolutionerande tekniksprång för simulering av detta slag.

6 Simuleringsexempel och resultat

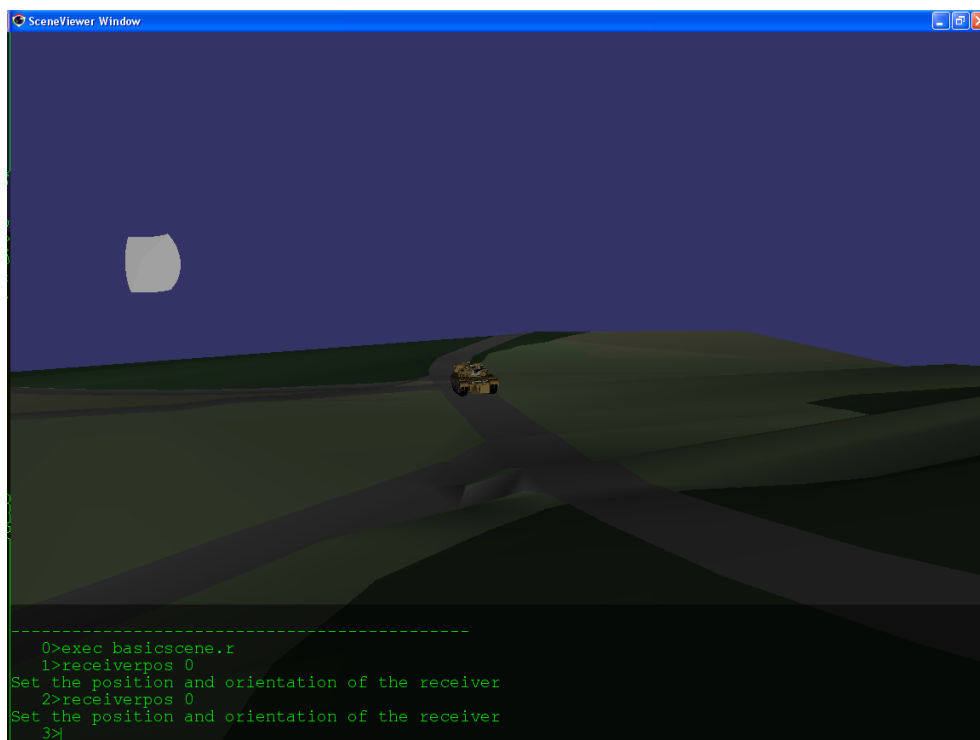
I detta avsnitt kommer några exempel att visas för att illustrera de möjligheter som erbjuds med den teknik som projektet studerat. Det bör påpekas att de metoder och materialparametrar som använts ej är kalibrerade, varför det inte är meningsfullt att jämföra absoluta värden på de radarmålyttestimat som framräknats med den framtagna programvaran. Viktigare är att studera och jämföra form och strukturer på exempelvis avståndsprofiler för ett mål eller en scen.

6.1 Visuell rendering och kommandotolk

Den första bilden, Figur 10, visar den visuella rendering av scenen som finns för att underlätta scenkomposition och verifiering av att den utförda simuleringen görs som tänkt. Uppe till vänster i Figur 10 syns den ikon som markerar radarsensorns placering i scenen och längst ner det kommandointerface som implementerats för att kontrollera och konfigurera simuleringen. Färgerna på marken representerar olika material, såsom gräsfält, vägar, etc. I detta exempel har en liten och ganska grov markmodell använts.

Gränssnittet tillåter att användaren ”åker omkring” i scenen och betraktar den från godtyckligt håll. Radarsensorn, eller radarsensorerna, kan placeras ut oberoende av observatörens placering. En radarsensor kan även flyttas till samma observationspunkt som den visuella med ett enkelt kommando.

Kommandotolken tillåter att kommandon samlas i en fil och körs i sekvens, vilket möjliggör en effektiv hantering av sammansatta scener. Animeringar och komplexa beräkningssekvenser kan också hanteras med denna funktion.



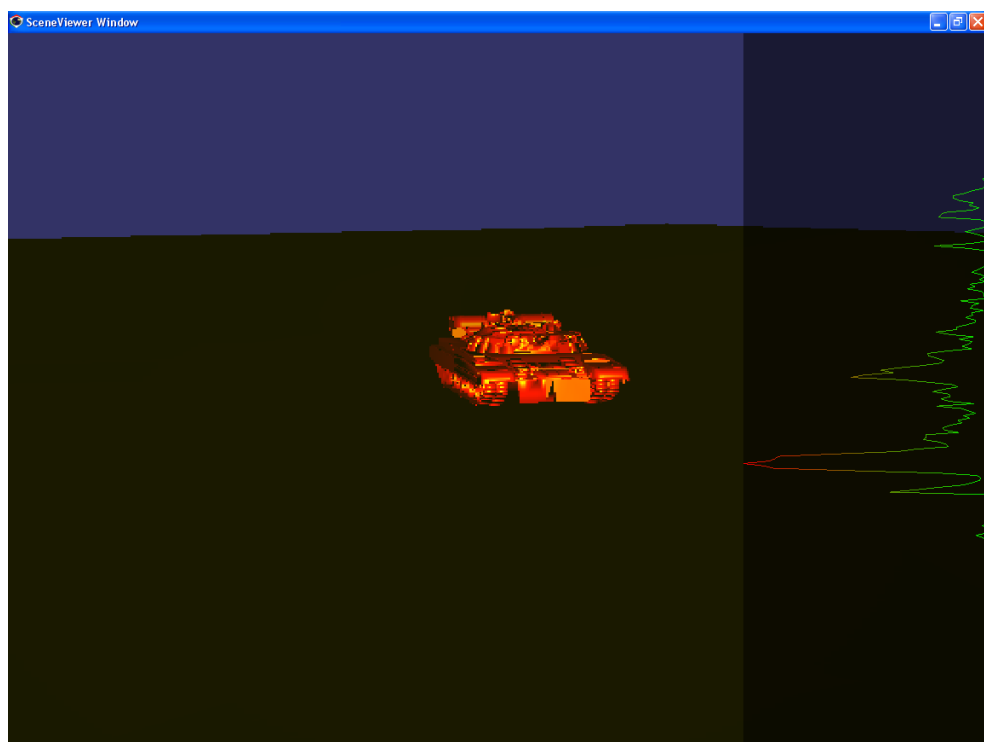
Figur 10: Bilden visar en visuell rendering av en scen. Kommandointerface och radarsensorns placering syns längst ner respektive uppe till vänster i bilden.

6.2 Visualisering av framräknat rådata till radarsimulering

I Figur 11 visas en bild av det utdata som radarberäkningen i fragmentprogrammet på grafikkortet resulterar i. Observera dock att denna bild bara är en skärmdump från den visualisering av resultatet som görs i programmet. Utdata från fragmentprogrammet skrivs egentligen till en databuffert som inte är synlig på skärmen (en s.k. pBuffer, eller pixel buffer) och hämtas därifrån tillbaka till programmet, vilket sedan visualiserar data i ett fönster på skärmen. Tillsammans med "rådata" visas även en avståndsprofil överlagrad till höger i bilden. Avståndsprofilen plottas nerifrån och upp, dvs. de nedre delarna är närmast mottagaren.

Radarreturen från en scen innehåller en mycket stor dynamik, dvs. variation i signalvärden i de olika pixlarna. I Figur 11 visas data i linjär skala, vilket gör att endast vissa strukturer i scenen syns. I Figur 12 visas samma scen plottad i logaritmisk skala. En bildskärm har ett mycket begränsat dynamikintervall och normala grafikkort kan endast visa bilder med en intensitetsupplösning på 8 bitar, eller 256 nivåer. De modernaste grafikkorten har möjlighet att visa 10 bitar eller 1024 nivåer, men få bildskärmar stödjer detta ännu.

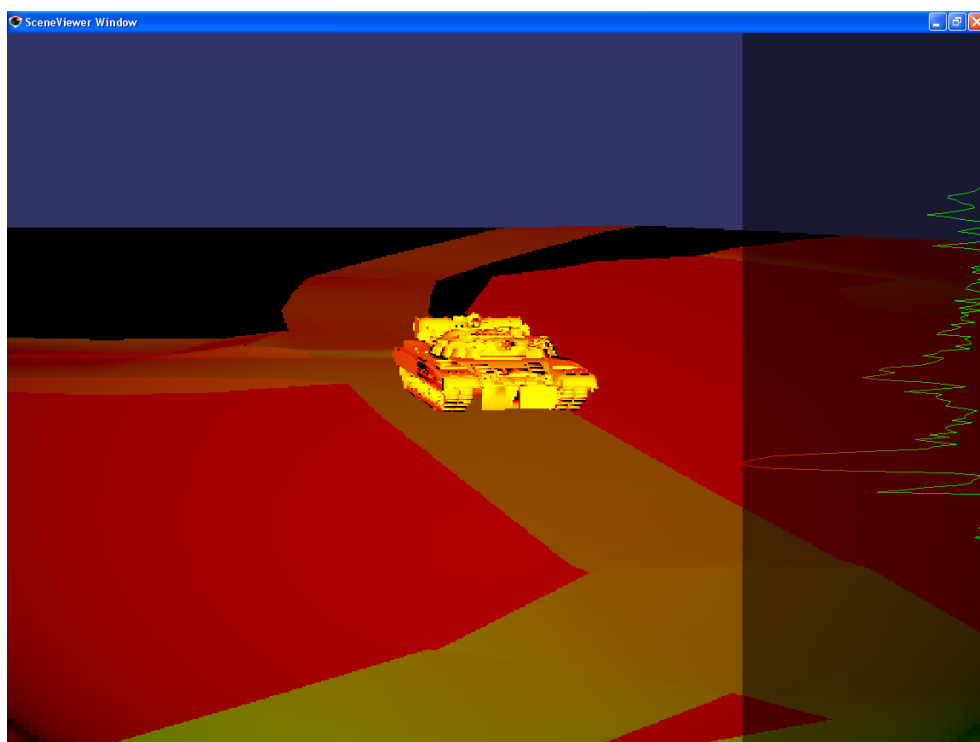
För att kunna tolka vad bilden visar måste man känna till hur utdata från fragmentprogrammet kodas i färgkanalerna. I den *röda* kanalen läggs energin från diffus spridning, dvs. energi som sprids utan fasinformation bevaras. Exempel på ytor med stor diffus andel spridd energi är skrovlig mark. I den *gröna* färgkanalen sparas den energi som sprids spekulärt, dvs. med fasinformationen intakt. Ytor som sprider energin på detta vis är typiskt metalliska och jämna. I den *blå* kanalen sparas fasvridningsinformation, dvs. den fasvridning som orsakas av ytans dielektriska egenskaper plus den fasvridning som kan finnas i sändarantennens lob. Detta data har en mycket begränsad variation, vilket gör att det ytterst sällan är synligt i bilderna. Den totala fasvridning som uppkommer pga. ytans avstånd från sändaren och mottagaren återfinns i *alfa* kanalen, vilken inte alls är synlig i den bild som visas på skärmen.



Figur 11: Bilden visar hur "rådata" ut från fragmentprogrammet kan se ut då det visas i linjär skala. Till höger ses också den avståndsprofil som kan räknas fram med detta data. Observera att färgerna i bilden är ett resultat av hur utdata från grafikkortet kodats i de tillgängliga färgkanalerna (RGBA). Rött är diffus och grönt är spekulär spridning.

Då data visas i logaritmisk skala erhålls en bild där man kan se returerna från scenen i sin helhet, se Figur 12. Man kan se de svaga returerna från marken, samtidigt som de kraftiga spekulära returerna från målobjekten framträder. Då returen från marken är i huvudsak diffus, blir dessa pixlar röda till färgen. Vägarna har getts en något mer spekulär materialegenskap och framträder därför med mer grönt i färgblandningen. Returen från den metalliska stridsvagnen är i huvudsak spekulär, vilket ger färgtoner med ännu mer grönt i. Observera också att programmet utför en autoskalning av den röda och den gröna färgkanalen var för sig. Det medför att man inte utan vidare kan jämföra nivåerna i de båda kanalerna inbördes.

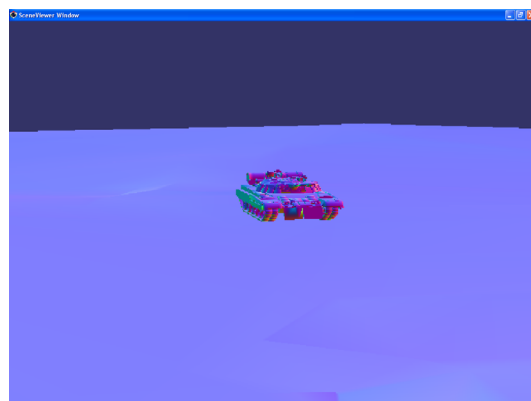
Man kan även lägga märke till att den diffusa spridningen är mindre vinkelberoende än den spekulära. Detta framträder t.ex. på sidan av stridsvagnen, där den spekulära energin sprids bort från mottagaren, men där den diffusa delen ger ett svagt bidrag.



Figur 12: Utdata från fragmentprogrammet visat i logaritmisk skala. De gröna delarna har i huvudsak spekulära materialegenskaper, medan de röda främst ger diffus energiretur.

6.2.1 Alternativa utdata från fragmentprogrammet

Det som beskrivits ovan om utdata från fragmentprogrammet, används vid den normala radarsimuleringen. Det finns dock möjlighet att skicka ut andra data från grafikkortet, vilket kan vara av stort intresse om man vill förbättra beräkningsresultatet ytterligare. Genom att ha kännedom om ytornas lokala orientering, ytnormal, i varje pixel kan man göra en approximativ uppskattning av dubbelstudsar. Denna projektiva metod att beräkna dubbelbidrag används exempelvis i programmet FOPOL. I Figur 13 visas en bild där färgerna (RGB) representerar ytnormalens xyz-komponenter och alfa kanalen avståndet till varje yta.



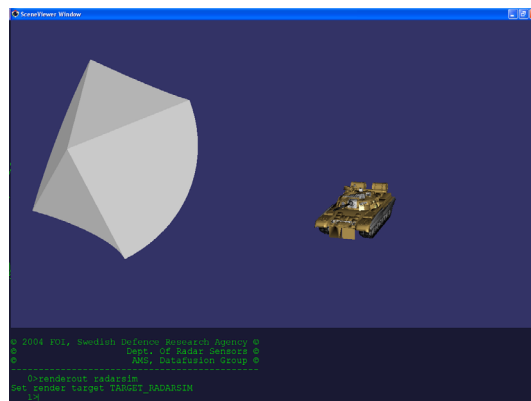
Figur 13: Ytans normalriktning relativt mottagaren (rgb=xyz).

6.3 Estimering av avståndsprofiler och RCS-beräkningar

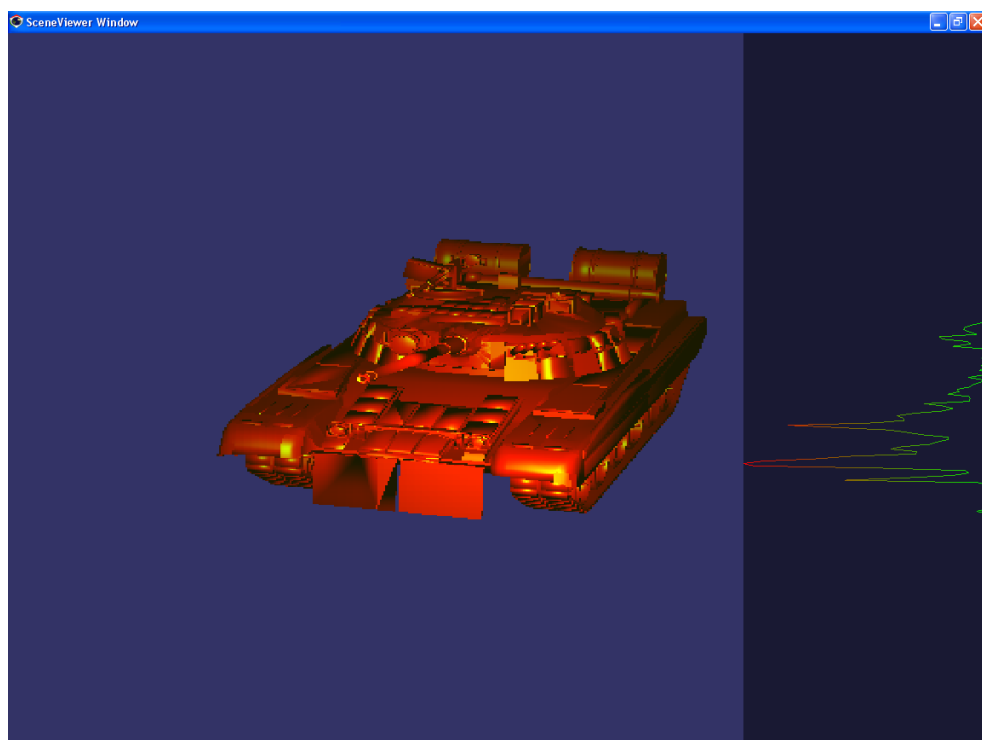
En av de viktigare användningarna av radarberäkningsprogram är att använda dem för studier av t.ex. måligenkänning. Man använder i dessa sammanhang vanligen antingen högupplösta avståndsprofiler eller SAR bilder. Då det är önskvärt att kunna räkna fram stora mängder träningsdata för algoritmer är accelererade beräkningar av det slaget som detta projekt intressant.

Figur 14 visar den mätuppställning som använts i beräkningarna i detta avsnitt. Radarantennens position visas med sfärsegmentet till vänster i bilden. I normala fall görs beräkning av radarmålyta m.m. på mycket större avstånd än i detta fall, eftersom man vill ha en plan vågfront in mot objektet. I illustrationssyfte gör vi dock beräkningen på ett kortare avstånd, dvs. i närfält.

I Figur 15 ser man data från radarsimuleringen av stridsvagnen av modell T80, plottad i linjär färgskala. Den tillgängliga dynamiken ligger vid de större intensitetsnivåerna i utdata, för att visa de små men kraftiga spekulära (grönt) reflektionscentra som finns på stridsvagnen. Dessa reflektionscentra bidrar mycket till objektets totala radarmålyta och är av intresse att lokalisera då man vill reducera radarmålytan från objektet. Man ser även att de flesta delarna är huvudsakligen röda, vilket innebär att dessa ytor reflekterar bort energin (kom ihåg att den röda kanalen och den gröna skalas oberoende av varandra vid visningen).



Figur 14: Visuell bild av mätuppställningen i detta exempel.



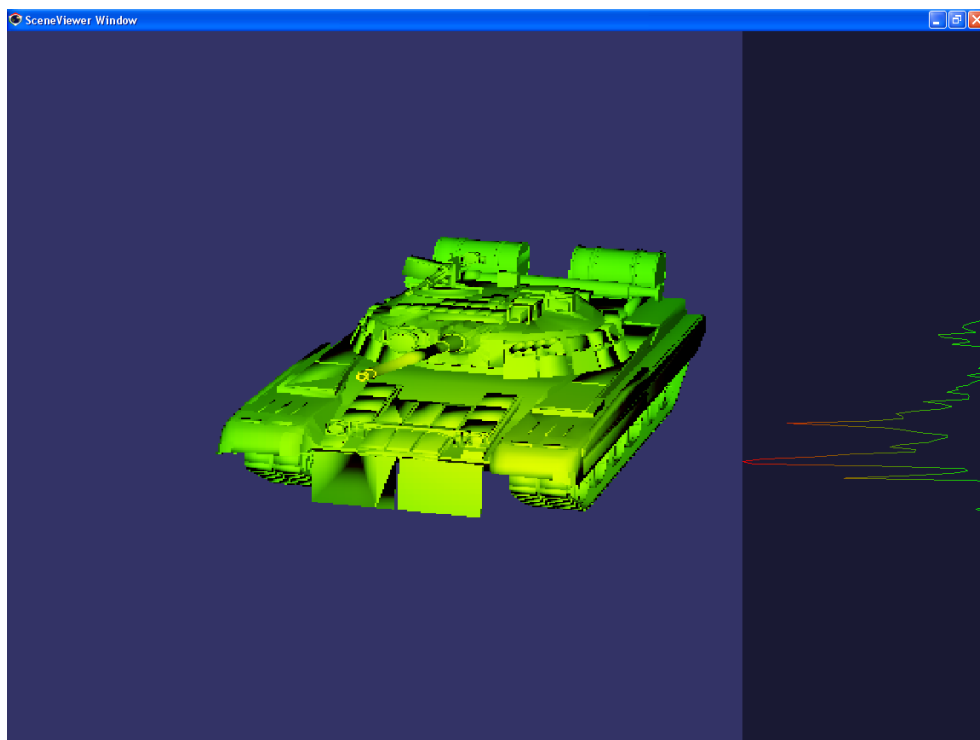
Figur 15: Resultat från beräkning av T80 med linjär plottning av utdata.

Man kan i denna bild dock se en del av begränsningen i den beräkning som görs med denna metod. I och med att multipelstudsar inte hanteras kommer de bidrag som skulle tillkomma på grund av exempelvis studs i vagnen - tornet och tillbaks, inte att finnas med i beräkningen. Det medför att

avståndsprofilen kommer att sakna några ”toppar” eller att dessa har fel amplitud. Erfarenhetsmässigt vet man dock att de mest betydande egenskaperna i avståndsprofilen härrör från enkelstudsar, varför metoden ändå är användbar för många syften.

Om man studerar den avståndprofil (till höger i bilden) som beräknats fram, återfinner man de kraftiga spekulära bidragen i de högre topparna. Den första lilla toppen härrör från spetsen av kanonröret, de efterföljande från vagnfronten och sedan tornet. Upplösningen i avståndsprofilen är ca. 0,3 m i detta fall.

Som jämförelse kan man plotta den kraftigt varierande spekulära energin i logaritmisk skala men behålla den långsammare varierande diffusa energin i linjär skala. Man erhåller då bilden i Figur 16. I denna är färgerna mer jämt fördelade mellan komponenterna och man kan se hur belysningsenergin fördelas över objektet. De områden som fortfarande har en röd övervikt i denna bild har en ytriktning som i det närmaste fullständigt reflekterar bort den infallande energin.



Figur 16: Bild med den spekulära komponenten plottad med logaritmisk skala och den diffusa plottad med linjär.

7 Sammanfattning och slutsatser

Vi började denna rapport med att diskutera när och varför man vill ha interaktiva radarsimuleringar i syntetiska miljöer. Användningsområdena som framkom är algoritmutveckling, försöks- och uppdragsplanering samt undervisning. Vi fortsatte sedan med att diskutera olika simuleringsmetoder för radar och deras respektive för- och nackdelar. Det visade sig att det finns stort överlapp mellan radarberäkningsmetoder och metoder från datorgrafiken, men också att man måste vara försiktig med de använda begreppen då de är bedrägligt lika/olika mellan de två olika fälten. Den metod vi kan rekommendera för framtida interaktiv radarsimulering är att man implementerar högfrekvensapproximationen GO/PO med hjälp av Photon Mapping, som är en modern global belysningsmetod från datorgrafiken med många attraktiva egenskaper såsom låg beräkningskomplexitet, hög flexibilitet och med inbyggd skalbarhet mellan kvalitet och prestanda, s.k. ”graceful degradation”.

För att uppnå hög kostnadseffektivitet och höga simuleringsprestanda vill vi utnyttja den explosionsartade utvecklingen av billiga konsumentgrafikkort. Denna utveckling diskuterade vi härnäst i rapporten och vi kunde konstatera att grafikkort numera, med fördel, kan användas till andra beräkningar än bara grafik. Utvecklingspotentialen är enorm och vi rekommenderar starkt dessa nya programmerbara grafikkort som en billig och kraftfull beräkningsresurs. Framför allt framkom att radarberäkningar baserade på GO/PO implementerat med Photon Mapping redan idag är möjliga att implementera på ett vanligt grafikkort, men att interaktiva hastigheter för komplexa scener först är möjliga om några år, runt 2007.

För att visa potentialen med radarsimulering på moderna programmerbara grafikkort presenterade vi sedan en simuleringsapplikation vi implementerat i projektet. Implementationen baseras på den enklare belysningsmodell som används i OpenGL, men är utökad med vertex- och fragmentshaderprogram som utför de radarspecifika reflektionsberäkningarna. Använda hjälpprogramvaror beskrevs och den fysikaliska modellen gick igenom. Till sist visade vi simuleringsexempel och resultaten från dessa presenterades.

Sammanfattningsvis vill vi rekommendera Photon Mapping för framtida interaktiva radarsimuleringar och framhålla att man med stor fördel kan göra implementationen på vanliga konsumentgrafikkort.

Tack till

FOI Innovationsfond för att vi har fått möjligheten att genomföra denna studie. Vi vill även tacka projektet Avancerade MålSökare, AMS, för att ha stöttat studien, och speciellt Sara Molin och Fredrik Näsström som starkt har bidragit till vår försöksimplementation.

Referensförteckning

- [1] "Beräkningsmetoder för radarmålära. En översikt.", Jan Fagerström, Swedish Defence Agency, FOI, Vetenskaplig Rapport, FOA-R--97-00553-615, ISSN 1104-9154, Augusti 1997
- [2] "Fundamentals of Computer Graphics", Peter Shirley, ISBN 1-56881-124-1, 2002, AK Peters
- [3] "Realistic Image Synthesis Using Photon Mapping", Henrik Wann Jensen, ISBN 1-56881-147-0, 2001, AK Peters (See also <http://graphics.ucsd.edu/~henrik/>)
- [4] "Real-Time Rendering", Tomas Akenine-Möller, Eric Haines, ISBN 1-56881-182-9, 2002, AK Peters (See also <http://www.realtimerendering.com/> for a lot of real-time rendering resources)
- [5] "Advanced Global Illumination", Philip Dutré, Philippe Bekaert, Kavita Bala, ISBN 1-56881-177-2, 2003, AK Peters, (See also <http://www.advancedglobalillumination.com/>)
- [6] "The Cg Tutorial", R. Fernando, M.J. Kalgard, ISBN 0-321-19496-9, 2003, Addison-Wesley Professional (See also http://developer.nvidia.com/object/cg_tutorial_home.html)
- [7] Nvidia Corporation, Products, Developer Information and Presentations, <http://www.nvidia.com/page/desktop.html>, <http://developer.nvidia.com/page/home.html> & http://developer.nvidia.com/page/event_calendar.html
- [8] ATI Technologies Inc., Products and Developer Information, <http://www.atitech.com/products/>, <http://www.atitech.com/developer/index.html>
- [9] "Realistic Ray Tracing", Peter Shirely, R. Keith Morley, ISBN 1-56881-198-5, 2003, AK Peters
- [10] "An Introduction to Ray Tracing", Edited by Andrew S. Glassner, ISBN 0-12-286160-4, 1989, Morgan Kaufmann
- [11] "Global Illumination Compendium. The Concise Guide to Global Illumination Algorithms", Philip Dutré, Katholieke Universiteit Leuven 2003, <http://www.cs.kuleuven.ac.be/~phil/GI/>
- [12] "The Rendering Equation", James T. Kajiya, ACM SIGGRAPH Computer Graphics, Volume 20, Issue 4 (August 1986), ISSN:0097-8930, <http://doi.acm.org/10.1145/15886.15902>
- [13] "Robust Monte Carlo Methods For Light Transportation Simulation", Eric Veach, Stanford University PhD Dissertation 1997, http://graphics.stanford.edu/papers/veach_thesis/
- [14] "On Robust Monte Carlo Algorithms For Multi-Pass Global Illumination", Frank Suykens, Katholieke Universiteit Leuven PhD Dissertation 2002, <http://www.cs.kuleuven.ac.be/cwis/research/graphics/CGRG.PUBLICATIONS/FRANKPHD/>
- [15] "Realtime Ray Tracing and Interactive Global Illumination", Ingo Wald, Saarland University PhD Dissertation 2004, <http://www.mpi-sb.mpg.de/~wald/PhD/>
- [16] "inTrace - Realtime Ray-Tracing Technologies", <http://www.intrace.com>

-
- [17] "State of the Art in Global Illumination for Interactive Applications and High-quality Animations", Cyrille Domez, Kirill Dmitriev and Karol Myszkowski, published in Computer Graphics Forum issue 22:1 2003, http://www.mpi-sb.mpg.de/resources/anim/EG_STAR/EGCGF02.pdf
 - [18] "Global Illumination for Interactive Applications and High Quality Animations", Organizers: Cyrille Domez, Philipp Slusallek, Course at Siggraph 2003, <http://www.mpi-sb.mpg.de/resources/anim/Course27/index.html>
 - [19] "Industrial-Strength Global Illumination", Per H. Christensen, Part of course at Siggraph 2003, http://www.mpi-sb.mpg.de/resources/anim/Course27/Part_5.pdf
 - [20] "SaarCOR - A Hardware Architecture for Real Time Ray Tracing", <http://www.saarcor.de/>
 - [21] "Realtime Ray Tracing of Dynamic Scenes on an FPGA Chip", Jörg Schmittler, Sven Woop, Daniel Wagner, Wolfgang J. Paul, and Philipp Slusallek, Graphics Hardware 2004, <http://graphics.cs.uni-sb.de/~jofis/SaarCOR/DynRT/DynRT.html>
 - [22] "Radiosity on Graphics Hardware", Greg Coombe, Mark Harris, and Anselmo Lastra, Graphics Interfaces 2004, <http://www.cs.unc.edu/~coombe/research/radiosity/> or <http://www.cs.unc.edu/~coombe/research/radiosity/coombe04radiosity.pdf>
 - [23] "Real Time Photon Mapping", Timothy R. Jozwowski, Michigan Technological University Master Thesis 2002, www.cs.mtu.edu/~shene/NSF-2/Tim-Jozwowski.pdf
 - [24] "Modelling Diffraction in the Monte Carlo Ray Trace Environment", Katherine L. Coffey, Chapter 3&4 of Master Thesis Virginia Tech 1998, <http://scholar.lib.vt.edu/theses/available/etd-110398-161755/>
 - [25] "Edge Diffraction in Monte Carlo Ray Tracing", Edward R. Freniere, G. Groot Gregory and Richard A. Hassler, Published In Optical Design and Analysis Software, Proceedings of SPIE, Volume 3780, Denver, 1999, www.lambdares.com/data/Ttechpapers/edge-diffraction-1999.pdf
 - [26] "A Uniform Theory of Diffraction for an Edge in a Perfectly Conductive Surface", R. G. Kouyoumjian, P.H. Pathak, Proc. IEEE **62**:1448, 1974
 - [27] "An analytical time-domain model of edge diffraction", Svensson U. P., Andersson R. I., Vanderkooy J., in Proceedings of NAM98, Nordic Acoustical Meeting, Stockholm, Sweden, Sept. 7-9, pp. 269-272, 1998, <http://www.tele.ntnu.no/users/svensson/ED.html>
 - [28] "Computer Graphics On A Stream Architecture", John Douglas Owens, Stanford University PhD Dissertation 2002, http://graphics.stanford.edu/papers/jowens_thesis/
 - [29] "Ray Tracing on a Stream Processor", Timothy John Purcell, Stanford University PhD Dissertation 2004, http://graphics.stanford.edu/papers/tpurcell_thesis/
 - [30] "Ray Tracing on Programmable Graphics Hardware", Timothy J. Purcell, Ian Buck, William R. Mark and Pat Hanrahan, ACM Transactions on Graphics. *21* (3), pp. 703-712, 2002. (Proceedings of ACM SIGGRAPH 2002), <http://graphics.stanford.edu/papers/rtongfx/>
 - [31] "Photon Mapping on Programmable Graphics Hardware", Timothy J. Purcell, Craig Donner, Mike Cammarano, Henrik Wann Jensen, and Pat Hanrahan, Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, pp. 41-50, 2003, <http://graphics.stanford.edu/papers/photongfx/>
 - [32] "GPU Gems", Edited by R. Fernando, ISBN 0-321-22832-4, 2004, Addison-Wesley Professional (See also http://developer.nvidia.com/object/gpu_gems_home.html)
-

-
- [33] "GPU Gems 2", Edited by Matt Pharr, Series Editor R. Fernando, ISBN 0-321-33559-7, 2005, Addison-Wesley Professional (See also http://developer.nvidia.com/object/gpu_gems_2_home.html)
 - [34] DirectX Graphics API, <http://msdn.microsoft.com/directx/>
 - [35] OpenGL Graphics API, <http://www.opengl.org/>
 - [36] "OpenGL Programming Guide, Fourth Edition", Dave Shreiner, Mason Woo, Jackie Neider, Tom Davis, ISBN 0-321-17348-1, 2004, Addison-Wesley Professional
 - [37] "Trends in GPU Evolution", Industrial Seminar Eurographics 2004, http://eg04.inrialpes.fr/Programme/IndustrialSeminar/PPT/Trends_in_GPU_Evolution.pdf
 - [38] "GDC 2005 Nvidia Presentations", http://developer.nvidia.com/object/gdc_2005_presentations.html
 - [39] "GDC 2005 ATI Presentations", <http://www.atitech.com/developer/techpapers.html>
 - [40] "When Will Ray-Tracing Replace Rasterization?", Kurt Akeley, David Kirk, Larry Seiler, Philipp Slusallek, <http://www.siggraph.org/s2002/conference/panels/panels1.html>
 - [41] "Ray Tracing As The Future Of Computer Games", Juri A. Oudshoorn, University Of Utrecht 1999, <http://give-lab.cs.uu.nl/ve&gd/raytracing.pdf>
 - [42] "The Ray Engine", N. Carr, J.D. Hall. and J.C. Hart , Proc. Graphics Hardware, Sep. 2002, <http://graphics.cs.uiuc.edu/~nacarr/papers/RayEngine-gh02.pdf>
 - [43] "GPU Algorithms for Radiosity and Subsurface Scattering", N. Carr, J.D. Hall. and J.C. Hart, Proc. Graphics Hardware, July 2003, <http://graphics.cs.uiuc.edu/~nacarr/papers/gpuradsub.pdf>
 - [44] "PhysX: The world's first physics processing unit", <http://www.ageia.com>
 - [45] "GPGPU, General Purpose Computation using Graphics Hardware", <http://www.gpgpu.org/>
 - [46] "Shader-Tech, A focus on modern GPU programming...", <http://www.shader-tech.com/>
 - [47] BrookGPU, <http://graphics.stanford.edu/projects/brookgpu/>
 - [48] "Sh, Embedded Metaprogramming Language", <http://libsh.org/>
 - [49] "SIGGRAPH 2004 GPGPU COURSE", Organizers: Mark Harris, David Luebke, Course at Siggraph 2004, <http://www.gpgpu.org/s2004/>
 - [50] "IEEE Visualization 2004 TUTORIAL", Organizer: Aaron Lefohn, Course at IEEE Visualization 2004, <http://www.gpgpu.org/vis2004/>
 - [51] "Raytracing on Programmable Graphics Hardware", Filip Karlsson, Carl Johan Ljungstedt, 2005, <http://www.ce.chalmers.se/edu/proj/raygpu/>
 - [52] "Implementing Ray Tracing on GPU", Martin Christen, 2005, <http://www.clockworkcoders.com/ogls/rt/>
 - [53] "Dynamic Ambient Occlusion and Indirect Lighting", Michael Bunnell, Chapter in "GPU Gems 2", http://download.nvidia.com/developer/GPU_Gems_2/GPU_Gems2_ch14.pdf
 - [54] "Dynamic Ambient Occlusion and Indirect Lighting", Michael Bunnell, Nvidia, GameDevelopers Conference 2005, http://developer.nvidia.com/object/gdc_2005_presentations.html
 - [55] "Scene Graph Rendering", Dirk Reiners, OpenSG Forum 2002, <http://graphics.cs.ucdavis.edu/~staadt/ECS275/scenegraph-rendering.ieeevr2002.pdf>
-

- [56] "OpenSceneGraph", <http://www.openscenegraph.org/>
- [57] "Open Producer", <http://www.andesengineering.com/Producer/>
- [58] "OsgNV", <http://osgnv.sourceforge.net/>
- [59] "RADAR Cross Section 2nd Ed", E.F. Knott, J.F. Shaeffer, M.T. Tuley, ISBN 0-89006-174-2, Artech House, Boston London, 1993
- [60] "NVIDIA GPU Programming Guide, Version 2.2.0", NVIDIA Corporation, 2004, http://developer.nvidia.com/object/gpu_programming_guide.html