HÖGSKOLAN SKÖVDE
Institutionen för kommunikation och information

Mikrovågssimulering med realtidsljus

Realtids-ray tracing i CUDA

Simon Haggren

Examensarbete i datalogi med inriktning mot dataspelsutveckling

Mikrovågssimulering med realtidsljus

Examensrapport inlämnad av Simon Haggren till Högskolan i Skövde, för Kandidatexamen (B.Sc.) vid Institutionen för kommunikation och information. Arbetet har handletts av Peter Sjöberg.

2010-09-03

Härmed intygas att allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och att inget material är inkluderat som tidigare använts för erhållande av annan examen.

Signerat: _____

Mikrovågssimulering med realtidsljus

Simon Haggren

Sammanfattning

Detta arbete undersöker möjligheterna med att simulera mikrovågor i ett slutet system. Systemet implementeras med en redan befintlig teknik kallad ray tracing. Ray tracing är en ljussättningsteknik som går ut på att simulera fotoners rörelse mellan ljuskälla och betraktare i en miljö man önskar ljussätta, och sedan belysa de områden som blir träffade för att på detta vis rendera en bild. Fotoner och mikrovågor har egenskaper som liknar varandra då de båda är elektromagnetism med olika våglängder. Ray tracing är en krävande algoritm då många uträkningar för varje foton måste utföras varje uppdatering. Därför har algoritmen implementerats med CUDA, ett bibliotek från Nvidia som gör det möjligt att använda GPU:n som ett generellt beräkningssystem. Detta är lämpligt för just den här typen av problem då GPU:ns arkitektur är ämnad för multipla, parallella uträkningar.

Nyckelord: Realtids ray tracer, Mikrovågor, Simulering, CUDA, GPGPU

Innehållsförteckning

1 Introduktion

Företaget Gisip tillverkar produkter för uppvärmning och torkning inom industri med hjälp av mikrovågstekniker. Gisip har vid framställning av deras produkter skapat reella fysiska prototyper, vilket kostar dem tid och pengar att framställa. Därför söker de alternativ för att modellera och simulera deras produkter med datorsimuleringar.

Då hela modelleringen är alldeles för stor och möjligen ogenomförbar så ligger fokus på själva mikrovågssimuleringen och huruvida en lösning skulle kunna se ut med hjälp av realtidsljussättning. Vid realtidsljussättning finns det framförallt två tekniker som man kan använda sig av, rasterisering och ray tracing.

Rasterisering är det som vanligast används i dagens spel då det är en snabb teknik som man kan få relativt bra resultat ifrån. Tekniken fungerar genom att man kollar varendra triangel i scenen och räknar ut ljusets direkta påverkan från ljuskällan, baserat på ytnormalen. Vid ray tracing så kalkyleras ljusberäkningarna genom att följa alla strålar som genereras och applicera ljus på trianglarna som korsar strålen (Popov, 2006).

Fördelarna med ray tracing är att man på ett enkelt och realistiskt sätt kan simulera reflektioner och refraktioner, vilket är nödvändigt för detta projekt. Dock är ray tracing mycket krävande och det kan vara svårt att uppfylla kriteriet om realtid. Som ett mål för att lösa detta så kommer CUDA att användas.

CUDA är en instruktionsuppsättningsarkitektur ifrån Nvidia som är till för att exploatera kraften i GPU (**G**raphics **P**rocessing **U**nit) för att använda den som ett generellt beräkningssystem. Detta gör så att man kan använda GPU för att göra de många kalkyleringar som behövs för en ray tracer och avlasta dessa ifrån CPU(**C**entral **P**rocessing **U**nit).

Mikrovågssimuleringen sker i ett slutet system där simuleringen går ut på att kartlägga energidistribueringen av de projekterade mikrovågorna. Som validering kommer Gisip att tillhandahålla data av energidistribuering i en kub. Denna kub kommer att modelleras i applikationen. Det kommer även finnas en annan part där en jämförelse med en tyngre ljusberäkningsteknik kommer att ske.

2 Bakgrund

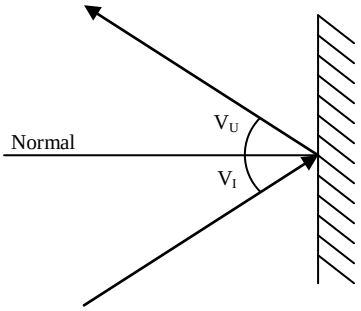
Företaget Gisip producerar produkter vars uppgift är att torka olika former av varor med hjälp av framförallt mikrovågor. Vid varje beställning av en ny produkt utvecklar de fysiska prototyper och kör tester på dessa för att se om varan går att torka på det sätt som kunden vill. Om någonting inte stämmer så kan Gisip i värsta fall vara tvungna att börja om från början och utveckla en ny prototyp. Detta är en väldigt kostsam process i både tid och pengar.

Detta arbete är ett delsteg mot att lösa problemet som Gisip har. Arbetet kommer att ske ihop med en kollega, Björn Karlsson. Vi skall tillsammans simulera mikrovågor men kommer att fokusera på två olika tekniker för att göra detta. Karlsson kommer att fokusera på en teknik som förhoppningsvis kommer att återspegla verkligheten så nära som möjligt, dock så är den långsam och det tar lång tid att få utdata. Den teknik detta arbete fokuserar på är inte lika ackurat, men den är mer effektiv i fråga om tid. Vilket är ett måste då krav på realtid finns. Båda lösningarna handlar om att simulera mikrovågor med redan befintliga ljussättningstekniker. När man pratar om realtidsljussättning så finns det framförallt två metoder man kan kolla på med dagens teknik, rasterisering och ray tracing (Ploeg, 2006).

2.1 Mikrovågor

Mikrovågor är en elektromagnetisk vågrörelse som har ett frekvensspann på cirka 10^7 – 10^{11} Hz vilket ger en våglängd på mellan några millimeter till ett par meter. Mikrovågor har applicerats på ett antal olika tillämpningar som t.ex. radar, kommunikation och mikrovågsugnar. Mikrovågorna i en mikrovågsugn har den specialla egenskapen att de reflekteras mot väggarna medans nästan alla ämnen inuti mikrovågsugnen blir i stort opåverkade, förutom vattenmolekyler som blir starkt uppvärmda (Alphonce, et al 1998). Enligt Schmitz och Kobbelt (2006) så uppför sig mikrovågor ungefär på samma sätt som ljus med avseende på reflektion, refraktion, interferens och diffraktion. De största skillnaderna uppstår på grund av att mikrovågor har längre våglängd så diffraktion uppstår och måste tas i åtanke, vid ljus är diffraktion nästan försumbart.

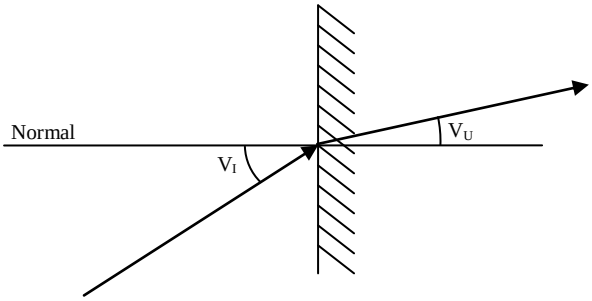
2.1.1 Reflektion



Figur 1 Reflektion vid total reflektion

I ett totalt återspeglande medium blir alltid strålen helt reflekterad. Reflektionen sker då så att infallsvinkeln (V_I) alltid är densamma som utfallsvinkeln (V_U) sett ifrån normalen. Detta fenomen där alla strålar reflekteras oberoende på vinkel kallas total reflektion (Alonso, 1992).

2.1.2 Refraktion



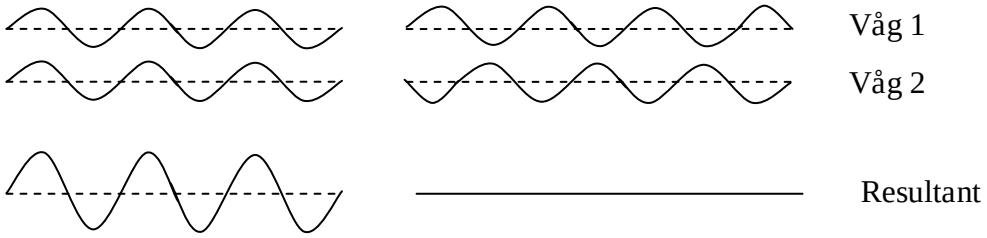
Figur 2 Refraktionsmodell

Refraktion är det fenomen som uppstår när strålen träffar mediet men inte reflekteras bort från det. Strålen bryts då in mot mediet med en ny vinkel. Den nya utfallsvinkeln på strålen kan beräknas genom att använda Snells lag:

$N_1 \sin V_I = N_2 \sin V_U$

Där N_1 respektive N_2 är brytningsindex. Brytningsindexet på olika material är bland annat baserat på strålens frekvens (Alonso, 1992).

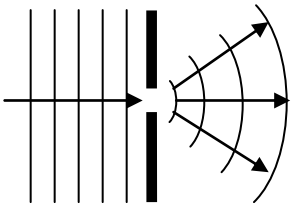
2.1.3 Interferens



Figur 3 Till vänster påvisas konstruktiv interferens och till höger destruktiv interferens

Interferens uppstår då en eller flera mikrovågor korsar varandra. När vågorna korsar varandra så uppstår ett fenomen där amplituderna på mikrovågorna interfererar med varandra. Att vågorna är i fas betyder att vågornas förskjutning är i jämn takt. Detta ger en summerad amplitud av de båda korsande vågorna. Detta kallas för konstruktiv interferens, om de båda vågorna är av samma amplitud och de ger ifrån sig ett interferensmaximum så blir den gemensamma amplituden lika mycket som amplituden multiplicerat med två (se vänster i figur 3). Om vågorna hamnar i ofas, det vill säga när vågorna är förskjutna med en halv våglängd. Då tar vågorna ut varandra och det blir en destruktiv interferens(se höger i figur 3) (Alonso, 1992).

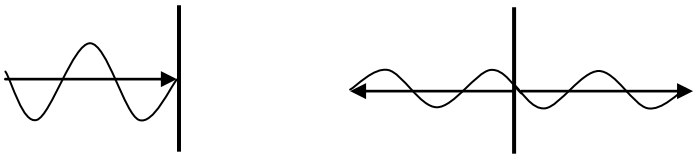
2.1.4 Diffraktion



Figur 4 Vågorna sprider sig efter att ha passerat en smal öppning

Diffraktion uppstår då flera vågor möter ett hinder. Man kan beskriva diffraktion som det synbara fenomenet av att vågorna tenderar böja sig runt hörn eller sprida sig när de passerar små öppningar (Alonso, 1992).

2.1.5 Absorption



Figur 5 Till vänster kolliderar en mikrovåg med ett objekt, till höger så uppstår det reflektion och refraktion där den refrakterande mikrovågen absorberas.

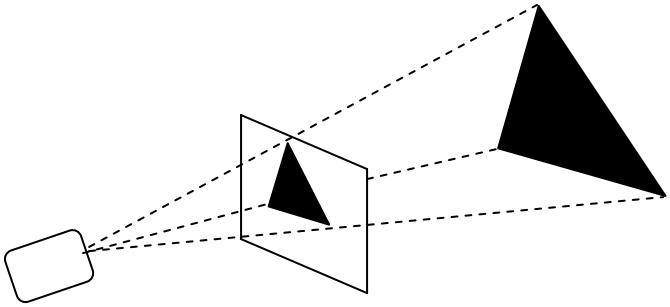
När en mikrovåg kolliderar med ett objekt så kan mediet i objektet reagera på den refrakterande mikrovågen. Detta får molekylerna i mediet att vibrera med vågrörelsen och värmeutveckling uppstår (Alphonce, et al 1998). Mikrovågorna som inte absorberas helt kommer antingen reflekteras eller refrakteras. Dessa mikrovågor får lägre effekt och kommer slutligen att dö ut på grund av senare absorptioner.

2.2 Ljussättningstekniker

Denna sektion kommer att inledas med rasteriseringstekniken, som är den vanligaste och snabbaste att använda idag. Sedan fortsätter det med ray tracing och hur denna teknik fungerar. Ray tracing är dock en krävande teknik så till slut kommer en diskussion om hur man kan gå tillväga för att optimera ray tracing.

2.2.1 Rasterisering

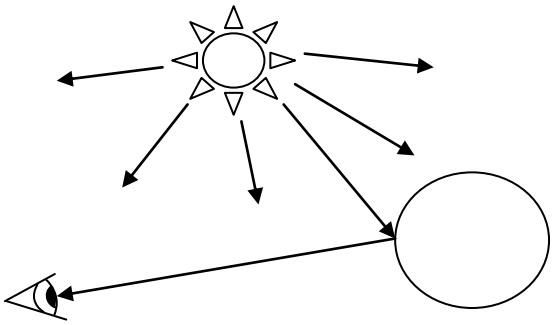
Rasterisering är en teknik som vanligast används i dagens realtidsbaserade applikationer som kräver ljussättning. Rasterisering är en effektiv teknik i tidsaspekt och det ger ett relativt godtyckligt resultat i de flesta fall. Vid rasterisering beskriver man primitiver med ett antal olika punkter. För att få vinst i prestanda så beskrivs dessa punkter oftast som trianglar (Mattson, 2003). Tekniken fungerar genom att man projicerar trianglarna från scenen genom en hålkamera och mappar dem på ett tvådimensionellt rutnät som kallas för färgbuffern (se figur 6). Varje element i rutnätet refereras som pixlar. För att resultatet ska kunna bli korrekt, så att inte bakomliggande trianglar hamnar framför närmre trianglar, använder man sig av en till buffer som kallas för Z-buffern. Z-buffern lagrar distansen från varje pixel till motsvarande del av triangeln. Den ser till så att endast de trianglar som ligger närmare kameran än de pixlar som redan finns på Z-buffern blir utskrivna på färgbuffern. I detta fall så skrivs även den nya informationen ut på Z-buffern (Popov, 2006).



Figur 6 Projektion av en triangel på färgbuffern

Rasterisering använder sig oftast av en ljussättningsteknik kallad för Phong shading som kan delas upp i tre kategorier, *Ambient-*, *diffus- och spekulärbelysning*. Den första kategorin, ambient belysning, motsvarar det approximerade ljuset som uppstår av reflektioner från multipla ljuskällor och manifesteras oftast som ett konstant ljus som appliceras uniformt på objekten. Den andra kategorin, diffus belysning, är det ljus som kommer direkt från ljuskällan och oberoende av betraktarens position appliceras över objektet. Spekulär belysning är den sista kategorin och uppstår i glansiga objekt där ljuset direkt reflekteras tillbaka till ögat från ljuskällan. Därför är spekulärbelysningen beroende på vart ljuskällan är positionerad, samt vart betraktarens öga befinner sig (St-Laurent, 2004).

2.2.2 Ray tracing



Figur 7 Principen bakom ray tracing, man följer ljusstrålarna och registrerar de som träffar kameran.

Ray tracing har länge använts inom underhållningsindustrin för att få fram de mest fotorealistiska bilderna, det har till exempel använts inom film och delvis inom vissa spel (Sherrod, 2008). Ray tracing är en teknik som till stor grad återspeglar hur ljuset uppför sig i verkligheten då tekniken fungerar genom att följa alla ljusstrålar som skapats från ljuskällan och se vart i världen som dessa kolliderar. När en kollision mot en triangel har detekteras så ljussätts området kring kollisionen, samt att en ny ljusstråle skapas och skickas iväg (Popov, 2006). Det finns i huvudsak två olika tekniker som används vid ray tracing, *forward ray tracing* och *backward ray tracing* (Sherrod, 2008).

Forward ray tracing är den naiva lösningen som bygger på att man från ljuskällan skickar ut ett antal ljusstrålar och spårar dessa till kameran (se figur 7). När man skickar iväg ljusstrålarna med ljuskällans spridning så krävs det att man skickar ut ett stort antal ljusstrålar för att öka chansen att några av ljusstrålarna skall pricka kameran. De ljusstrålar som inte prickar kameran kommer inte att hanteras. Detta gör forward ray tracing till en ineffektiv lösning då det sker beräkningar på många ljusstrålar som endast kommer att kastas bort (Sherrod, 2008).

Backward ray tracing löser det föregående problemet med överflödiga ljusstrålar som aldrig fångas upp. I backward ray tracing skickar man ut ljusstrålarna från kameras alla pixlar in emot scenen för att se om de slutligen träffar en ljuskälla. Med denna lösning ser man till att alla strålar som skickas kommer att ha träffat kameran (Sherrod, 2008).

2.2.3 Ray tracing i realtid

Ray tracing är en väldigt krävande teknik, då flera hundra tusen strålar måste skickas ut och testas mot geometrin vid varje uppdatering. En naiv ray tracer till ett modernt spel skulle behöva göra en biljon beskärningstester varje bilduppdatering (Allgyer, 2008). För att bygga en ray tracing applikation så kan man använda sig av *stackless KD-tree traversal* (Popov, 2006) och CUDA (Nvidia, 2009). Denna applikation är geometrifattig och en stackless KD-tree traversal har då inte implementeras.

2.3 CUDA

CUDA (**C**ompute **U**nified **D**evice **A**rchitecture) är en instruktionsuppsättnings-arkitektur ifrån Nvidia. CUDA:s uppgift är att utnyttja den multitrådade kapaciteten i GPU:n. Detta görs möjligt genom att använda C som högnivåspråk för att låta GPU:n fungera som ett generellt beräkningssystem, även kallat GPGPU (**G**eneral **P**urpose **G**raphics **P**rocessing **U**nit). Detta gör att man kan använda GPU:n för att lösa många olika komplexa beräkningsproblem som kan uppstå (Nvidia, 2009).

Ryoo et al. (2008) beskriver CUDA:s programmeringsmodell som ett system där det finns en *host*, som CPU:n, och en eller flera *devices* som motsvarar de olika GPU:erna som finns tillgängliga. Varje device består av ett antal processorkärnor och man kan med en liten kostnad skapa ett stort antal *trådar* som jobbar i ett massivt parallellt system. En tråd är en uppstyckning av en process och innehåller en sekvens av instruktioner. Alla trådar hanteras som ’delprocesser’ och kan köras parallellt med varandra. CUDA bygger på SPMD modellen (**S**ingle-**P**rogram **M**ultiple **D**ata) med vilket menas att alla trådar är baserad på samma kod men med olika indata för att få fram olika resultat. CUDA-programmering görs med en nyckelordsextension av språket C. Utökningen tillåter programmeraren att hantera minne mellan varje host och device samt att skapa funktioner kallade *kernels*, vilket är den kod som skall köras på GPU:n. När en kernel-funktion kallas så specificeras antalet trådar som skall köras parallellt. Varje tråd exekverar kerneln var för sig och de blir tilldelade en unik identifierare för att de skall kunna ge olika resultat. Flera tusentals trådar kan då skapas samtidigt från samma kernel för att lösa ett komplext beräkningssystem.

Enligt Parker et al. (1999) så lämpar sig en ray tracer utmärkt till att implementeras i ett parallellt system då varje pixel kan beräknas oberoende av varandra samt att datastrukturen för varje stråle oftast är *read-only*.

2.4 Ljussättningstekniker för att simulera mikrovågor

Som tidigare nämnt är rasterisering en teknik som är jämförelsevis billig i beräkningskraft gentemot ray tracing. Den stora nackdelen med rasterisering till detta projekt är att rasterisering endast avhandlar lokal illumination. Lokal illumination menas med att man endast kan bevisa det direkta ljusets påverkan lokalt i scenen. Man kan alltså inte påpeka för beteende så som reflektion, refraktion och dylikt. Ray tracing är en så kallad global illuminationsteknik. Med detta så påverkas hela scenen globalt från resten av scenen och man kan påvisa mikrovågors beteenden som reflektion och refraktion.

3 Problemformulering

Gisip är ett företag som tillverkar stora mikrovågsugnsliknande produkter. Dessa produkter måste testas som prototyper innan de kan tas i bruk för att kunna få ut en bra effekt på ugnen. I nuvarande läge är det kostsamt och dyrt för Gisip att testa dessa prototyper eftersom de produceras i fysisk form och de har inte några direkta möjligheter att genomföra några tester i förtid.

Detta arbete skall tillsammans med Björn Karlsson vara ett delsteg mot att hjälpa Gisip med detta problem. Under detta arbete så kommer en mikrovågssimulator att tillverkas, vars uppgift är att graflägga energidistribueringen av mikrovågorna i ett slutet system.

Enligt Schmitz och Wenig (2006) uppvisar mikrovågor och ljusstrålar samma egenskaper som till exempel reflektion och refraktion. De har även påvisat att det går att använda en befintlig ljussättningsteknik för att simulera olika former av mikrovågor. I deras fall var det ett ad hoc nätverk som bygger på mikrovågor vid kommunikation (Schmitz och Kobbelt, 2006).

Eftersom mikrovågor kan approximeras som ljusstrålar (Schmitz och Wenig, 2006) så kommer detta arbete, som tidigare nämt, tillverka en mikrovågssimulator som bygger på en realtids-ray tracer för simuleringen av mikrovågor. Stark fokus ligger på realtid och ett flertal approximationer och hjälpmedel tas i bruk för att realisera detta. Mikrovågssimulatorn kommer till exempel inte att hantera interferens och simulatorn kommer vara byggd med Nvidias CUDA. Realtid kommer att avhandlas som en bilduppdateringshastighet på femton bilder per sekund (Mattson, 2003).

3.1 Metodbeskrivning

3.1.1 Plattform

Då krav på realtid skall tas i åtanke så måste man specifiera systemet som applikationen kommer att köras på.

Processorn är en Intel® Core™ i3 Processor 330M (2,13 GHz, 3 MB). RAM 4 GB DDR3. Grafikkortet är Nvidia Geforce GT 330M 1 GB GDDR3. Operativsystemet består av Windows 7 Professional 64-bit.

3.1.2 Validering

Företaget Gisip har en grundläggande version av deras produkt som de använder för att sätta upp en testmiljö. Denna miljö består i stort av en kub med en eller flera magnetroner. En magnetron är en elektrisk komponent som skickar iväg mikrovågor, dessa mikrovågor har en frekvens på 2.45 GHz. I miljön kan de kartlägga mikrovågsdistribueringen genom att sätta ut vattenflaskor på olika ställen och jämföra temperaturökningen gentemot den utskickade potentiella energin från magnetronen. På detta sätt får man reda på effekten vid olika positioner i kuben.

Schmitz och Kobbelt (2006) gjorde en applikation för att undersöka vågpropageringen i ett trådlöst nätverk med hjälp av fotonmappning. Denna applikation kommer att valideras på ett liknande sätt som Schmitz och Kobbelt gjorde. De satte upp en testmodell som modellerade en verklig plats. I modellen simulerades nätverkets spridning, där de fick ut en textur vars färgvärden motsvarade styrkan på vågorna vid olika positioner. Detta system validerades genom att ställa ut mottagare och sändare vid utvalda punkter i den fysiska miljön. Därefter lästes styrkorna av vid varje

mottagare och man kunde således se hur signalstyrkorna stämde överens med den genererade texturen. Resultatet användes för att beskriva hur korrekt approximationen simulerade verkligheten.

En stor nackdel med den metoden är att man inte har tillgång till direkt rådata från verkligheten. Datan måste först simuleras för att sedan testas mot verkligheten och se om den stämmer vid specifika punkter. Det finns en stor chans att kritiska punkter i systemet som inte har blivit testade kan gå förlorat, vilket kraftigt kan påverka systemets pålitlighet.

För att jämföra datan från Gisip med applikationen så har en uppvisningsmiljö skapats. Denna uppvisningsmiljö modellerar insidan av Gisips kub. För var sida av kuben skapas det en textur varje uppdatering. Alla strålar som träffar en sida färglägger motsvarande pixel i texturen, samt att den även förlorar lite energi och reflekteras bort från ytan. Texturerna motsvarar då mikrovågsspridningen i kuben och man kan med det visa vart ansamlingar av strålarna kommer att befinna sig. Till det här så har även flaskor implementerats till uppvisningsmiljön. Med dessa flaskor kan man bedöma den procentuella spridningen mellan flaskorna som strålarna kolliderar med. Detta ger en högre precision på valideringen av systemet.

Förutom jämförelsen med den verkliga kuben från Gisip så görs även en jämförelse med Karlsson. Karlsson har gjort en applikation med samma problem, men kravet på realtid finns inte i Karlssons arbete. Jämförelsen fokuserar på skillnaden i korrektheten till verkligheten samt en avvägning mot tidsaspekten i detta projekt. Båda arbeten bygger på en gemensam uppvisningsmiljö för att underlätta jämförelsen.

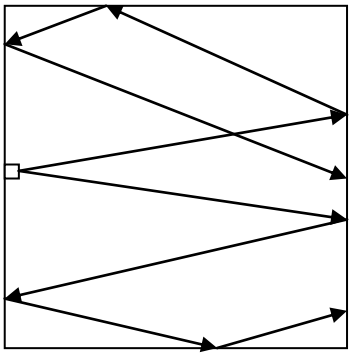
4 Genomförande

4.1 Det producerade systemet

4.1.1 Applikation

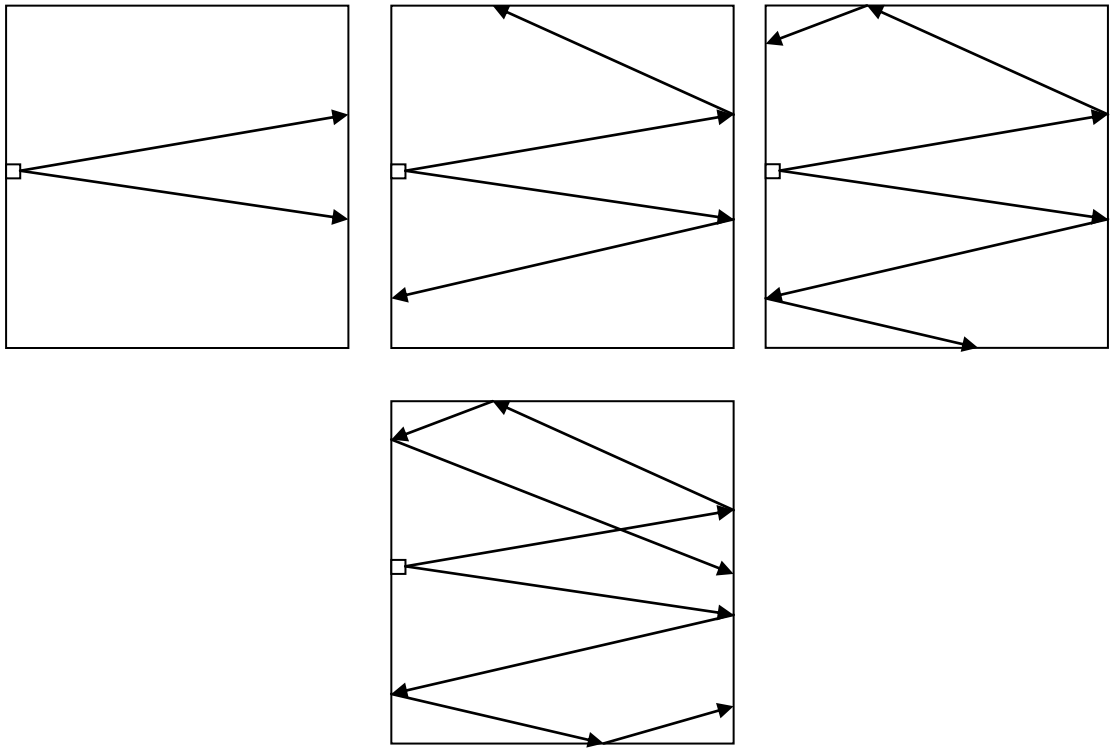
Applikationens slutgiltiga funktion är att i realtid generera sex stycken texturer som mappas till de olika planen på insidan av den simulerade kuben. Dessa texturer genereras som ett resultat av en ray tracer-algoritm. Varje pixel i texturen motsvarar kollisioner mellan planet och strålarna, och intensiteten på pixlarna motsvarar antalet kollisioner. Det vill säga om färgvärdet är noll så har inga kollisioner skett där.

Algoritmen har implementerats i CUDA, vilket leder till att koden inte kan vara objektorienterad och måste skrivas i C. I en traditionell ray tracer skapas alla strålar som behövs varje uppdatering för att sedan följas tills antal kollisioner har uppnått en viss kvot. När alla strålar har följts klart ritar man ut texturen och börjar om vid nästa iteration. Denna applikation skapar ett satt antal strålar från magnetronen varje runda. Varje stråle som skapas får initiellt ett bestämt energivärde. Energivärdet motsvarar hur många kollisioner strålen kan genomgå innan den dör ut. Varje iteration utgör en kollision av varje stråle. Detta ger att det vid varje given tidpunkt finns fler strålar i världen än vad magnetronen skickar ut vid varje iteration. Figur 8 visar hur en normal raytracer hade fungerat med två strålar och tre kollisioner per stråle. Varje pil mot en yta motsvara en kollision och det är där som texturen mot planet färglägges. Alla dessa steg sker i en iteration vilket ger långa iterationer.



Figur 8 En enkel översikt av strålarnas spridning i en kub

Figur 9 visar hur denna applikation följer strålarna. Varje ruta motsvarar en ny iteration så det tar totalt fyra iterationer innan den första strålen gör sin sista kollision. Detta ger kortare iterationer men fler strålar i systemet. Efter de första fyra iterationer så kommer systemet stabiliseras då det vid nästa iteration skickas ut en ny stråle exakt likadant som den föregående.



Figur 9 En progressiv översikt av strålarnas spridning i en kub

4.1.2 Struktur

Applikationen är uppbyggd med datastrukturerna Ray, Plane, Magnetron och Container. Datastrukturen Ray representerar varje stråle i världen och den innehåller värden för position, riktning och energi. Plane är varje sida på kuben och innehåller normalen på planet, positionen, storleken och texturen. Magnetron är den enhet som skapar och skickar iväg strålarna. Den innehåller position och riktning för magnetronen samt spridningen som strålarna skickas ut i och antalet strålar som skall skapas varje uppdatering. Strålarna ifrån magnetronen skickas ut i en kon från magnetronens position längsmed magnetronens riktning. Basen på konen är specifierad med spridningsvariabeln som man kan se i koden nedan.

```
void Emit(){
	srand(100);
	for(int i = 0; i < mag->nrOfEmittingRays; i++){
		float spr;
		if(mag->spread == 0)
			spr = 0;
		else{
			spr = rand() % (int)mag->spread;
			spr /= 10000;
		}
		AddRay(mag->position, make_float3(mag->direction.x + sinf(i + 1)*spr,
			mag->direction.y + cosf(i + 1)*spr,
			mag->direction.z));
	}
}
```

Seed-värdet på slumpgeneratorn är satt till en konstant eftersom strålarna skulle ge ett uniformt resultat varje uppdatering. Den sista datastrukturen, Container, beskriver en sfärisk kollisionsyta för att avgöra hur stor andel av strålarna som har kolliderat med den. Till dessa datastrukturer finns det diverse hjälpfunktioner för att räkna ut till exempel strålarnas reflektion samt längden och skalärprodukten på en vektor.

När man programmerar till CUDA så är det ett antal saker man måste ha i åtanke. För att programmera till den så skapar man olika funktioner som kallas för *kernels*. Dessa körs endast på GPU och istället för att kalla på en funktion i taget som man gör på CPU så kallar men funktionerna i ett massivt parrallellt trådat system. För att kunna använda variabler på GPU:n så måste först minne allokeras på GPU:ns ramminne och sedan kopiera över från CPU till GPU. Detta har en stor overhead och rekommenderas att göras så lite som möjligt. Här nedan är ett exempel på hur en kernel kan se ut. Denna kernel nollställer texturernas färgvärden.

```
__global__ void ResetImages(Plane* planes, int i){
	int a = blockIdx.x * blockDim.x + threadIdx.x;
	planes[i].image[a] = 0;

}
```

Nyckelordet `__global__` deklarerar funktionen som en kernel-funktion, vilket även ger tillgång till variablerna `blockIdx, blockDim` och `threadIdx` Dessa använder man för att räkna ut vilken av trådarna man skall indexera med. När kerneln kallas på så kallas den lika många gånger som det finns trådar. Antalet trådar i detta fall är satt till lika många som bildens storlek i bytes. Bildens storlek är 512x512 med 3 bytes per pixel. Det finns 6 stycken plan, vilket totalt ger 4718592 skapade trådar för att köra denna kernel. För att kalla på kerneln så används följade kod:

```
ResetImages<<<blocksPerGrid, threadsPerBlock>>>(cuPlane, i);
```

För att koden skall fungera så måste först datan för varje plan allokeras och kopieras till GPU:n, det görs på detta sätt:

```
size_t size = sizeof(Plane)*6;
CUDA_SAFE_CALL(cudaMalloc((void**) &cuPlane, size));
CUDA_SAFE_CALL(cudaMemcpy(cuPlane, planes, size, cudaMemcpyHostToDevice));
```

4.1.3 Algoritm

Här nedan kommer den huvudsakliga algoritmen för hur applikationen fungerar.

```
#Allokera magnetronen och de sex planen på CPU:n
Malloc magnetron
Malloc plane[6]
#Allokera planen på GPU
cudaMalloc plane[6]

While(true)
	ResetTextures() #återställ texturerna i varje plan
	EmitRays() #Kasta ut strålarna från magnetronen

	cudaMalloc(ray) #Allokera och kopiera strålarna till GPU
	#kod som körs på GPU

	For each ray

		For each plane
			vd = Dotproduct(plane.normal, ray.direction)
			if(vd = 0) #om strålarna är parallela med planet
				continue #hoppa över
			v0 = -(Dotproduct(plane.normal, ray.position) +
			    length(plane.position))

			t = v0/vd
			if(t <= 0) #om planet är bakom strålen
			    continue #hoppa över

			pi = ray.direction * t + ray.position
```

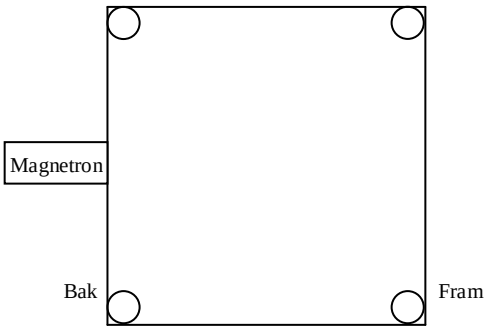
```
#hitta positionen i texturen på planet
colpos = findlengthfromcenter(pi, plane.position)

ray.position = pi
reflectRay(ray, plane.normal) #reflektera strålen
ray.energy = ray.energy  - 1 #sänk energin på strålen

#öka intensiteten på pixeln
colorpixel(plane.texture, colpos)

if(ray.energy <= 0) #ta bort strålen om energin är minre än 0

        delete(ray)

copytoCPU()#Kopiera strålarna och planen tillbaka till CPU
```

4.2 Genomförda mätningar

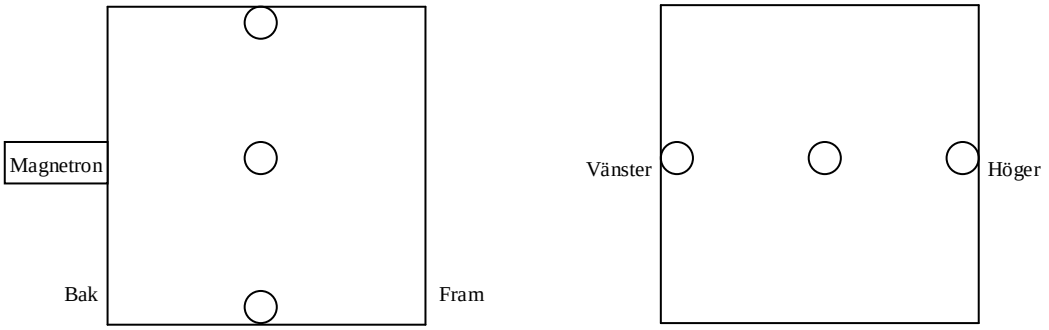
4.2.1 Testmiljö

På företaget Gisip gjordes ett antal olika mätningar vilket skall användas som underlag för att testa systemet. Mätningarna gjordes i en aluminiumklädd kub med ett inlopp för en magnetron. Kubens dimensioner mäter upp cirka 4 våglängder, det vill säga ungefär 49cm. På denna kub utfördes fyra olika testfall och varje testfall utfördes tre gånger för att få ett statistiskt underlag. Testfallen bestod i att fylla glasflaskor med 100 ±1 gram vatten för att sedan sätta ut dem vid olika positioner i kuben. Innan testet genomfördes mättes temperaturen i vattnet och därefter sattes magnetronen igång i två minuter. Efteråt mättes temperaturen återigen i de olika glasflaskorna. Genom att räkna ut den procentuella skillnaden på temperaturökningen mellan flaskorna så kan man mäta distriuberingen av strålarna inne i kuben.

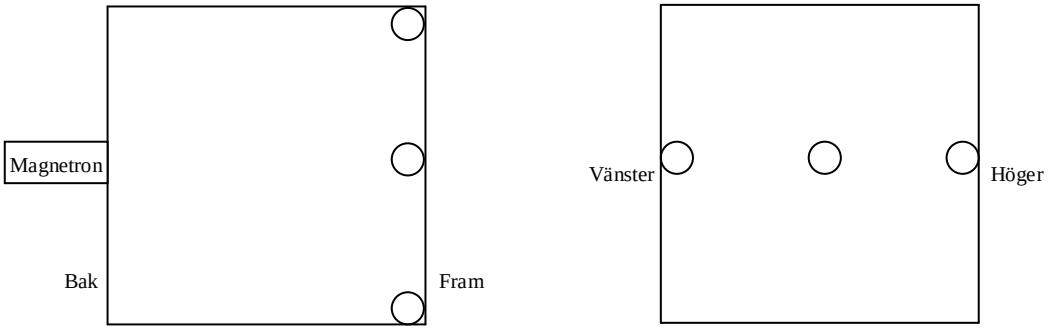
I första testet placerades fyra glasflaskori varsitt hörn (se figur 10). I det andra testet användes tre glasflaskor, dessa placerades längs mittlinjen och höjdes upp till halva höjden av kuben (se figur 11). I det tredje testet sattes tre stycken glasflaskor ut på rad längst med halva höjden av kuben (se figur 12). I det fjärde och sista testet sattes glasflaskorna likadant som i tredje testet fast högst upp i kuben (se figur 13).



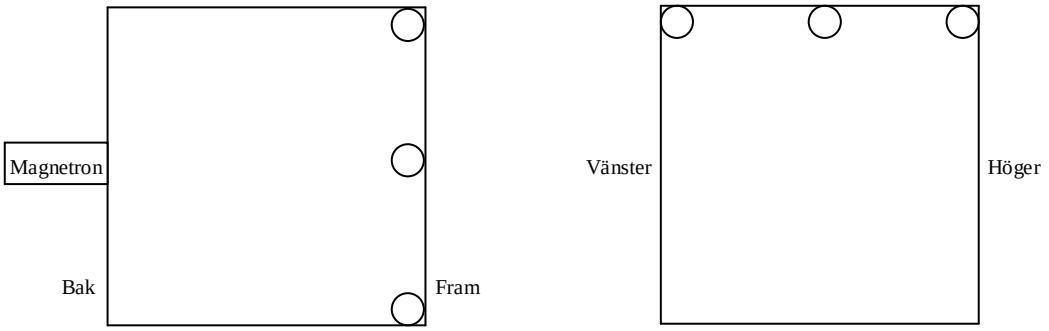
Figur 10 En visualisering av första testet sett ovanifrån.



Figur 11 Andra testet sett ifrån ovan till vänster och framifrån(sett ifrån magnetronens perspektiv) till höger.



Figur 12 Tredje testet sett ifrån ovan till vänster och framifrån(sett ifrån magnetronens perspektiv) till höger.



Figur 13 Fjärde testet sett ifrån ovan till vänster och framifrån(sett ifrån magnetronens perspektiv) till höger.

4.2.2 Samlad data

TEST	Temp Före	Temp Efter	Medelresultat i %
1-1	Alla = 18	BH = 38, BV = 40.5, FH = 53.5, FV = 42	
1-2	Alla = 18	BH = 38, BV = 41.5, FH = 54.5, FV = 41	
1-3	Alla = 18	BH = 35.5, BV = 41, FH = 56, FV = 41	BH = 18.76, BV = 22.51, FH = 35.89, FV = 22.83
2-1	Alla = 19	HM = 52, VM = 52, MM = 75	
2-2	Alla = 18	HM = 52, VM = 56.5, MM = 79.5	
2-3	Alla = 18,5	HM = 51, VM = 53, MM = 76	HM = 26.15, VM = 27.86, MM = 45.99
3-1	Alla = 20	FM = 81, FV = 60.5, FH = 68	
3-2	Alla = 20	FM = 80, FV = 68 , FH = 66	
3-3	Alla = 21	FM = 79, FV = 70 , FH = 68.5	FM = 39.04, FV = 30.04, FH = 30.91
4-1	Alla = 22	FM = 98, FV = 86 , FH = 71	
4-2	Alla = 14	FM = 98.5, FV = 83 , FH = 66	FM = 40.68, FV = 33.71, FH = 25.60

BH = Bakre högra, BV = Bakre vänstra, FH = Främre högra, FV = Främre vänstra, HM = Höger Mitten, VM = Vänster Mitten, MM = Mitten mitten.

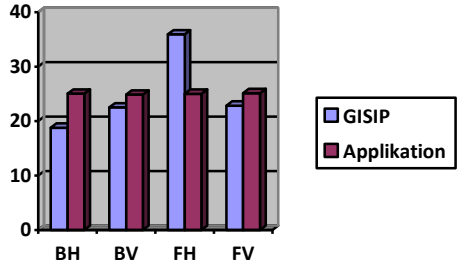
Figur 14 Den samlade datan från Gisip, spalten längst till höger beskriver den genomsnittliga procentuella temperaturhöjningen i vattnet.

TEST	Antal skärningar	% fördelning mellan vattenflaskorna	% fördelning av alla strålar som kolliderat
1-BH	37068	25.04	37.07
1-BV	36807	24.86	36.81
1-FV	37174	25.11	37.17
1-FH	37003	24.99	37.00
2-HM	39922	31.46	39.92
2-MM	47183	37.18	47.18
2-VM	39789	31.36	39.79
3-FM	44637	37.00	44.64
3-FV	37891	31.41	37.89
3-FH	38108	31.59	38.11
4-FM	38134	33.86	38.13
4-FV	37236	33.06	37.24
4-FH	37266	33.09	37.27

BH = Bakre högra, BV = Bakre vänstra, FH = Främre högra, FV = Främre vänstra, HM = Höger Mitten, VM = Vänster Mitten, MM = Mitten mitten.

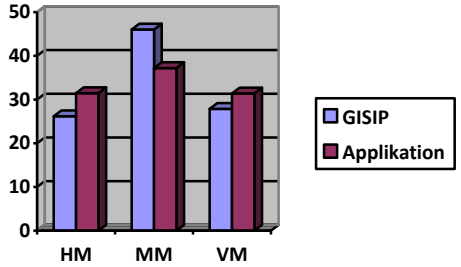
Figur 15 Den samlade datan från det producerade systemet, den procenturella fördelningen är mellan vattenflaskorna i systemet.

Alla tester utfördes tre gånger, förutom det fjärde som bara utfördes två gånger. Anledningen till detta är att under testkörningarna så blev flaskorna så upphettade att de började koka över när locket togs av. Här nedan kommer stapeldiagram som jämför den procentuella fördelningen mellan Gisips resultat och applikationens resultat.



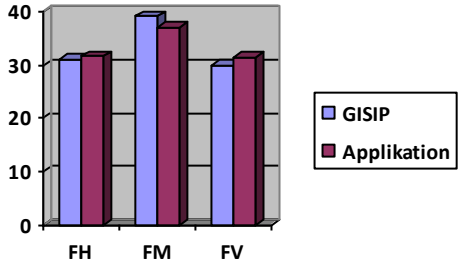
BH = Bakre högra, BV = Bakre vänstra, FH = Främre högra, FV = Främre vänstra

Figur 16 Test 1. Procentuell fördelning av strålarna vid de olika flaskorna.



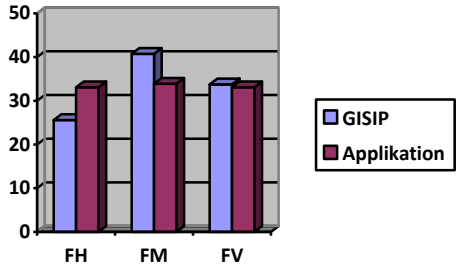
HM = Höger Mitten, VM = Vänster Mitten, MM = Mitten mitten.

Figur 17 Test 2. Procentuell fördelning av strålarna vid de olika flaskorna.



FH = Främre högra, FV = Främre vänstra, FM = Främre mitten.

Figur 18 Test 3. Procentuell fördelning av strålarna vid de olika flaskorna.



FH = Främre högra, FV = Främre vänstra, FM = Främre mitten.

Figur 19 Test 4. Procentuell fördelning av strålarna vid de olika flaskorna.

4.3 Analys av mätningar

4.3.1 Analys av datan

I det första testet är flaskorna utplacerade i varje hörn av kuben. Man kan där tänka sig att det kommer bli en helt jämn spridning över alla flaskor men att framsidan som magnetronen är riktad mot, har en något större fördelning av strålarna då de lättare kommer att falla in på dem. Applikationens resultat gav en nästan helt jämn fördelning på 25% men med en ytterst marginell ökning på den bakre högra och främre vänstra flaskan. Dessa siffror skulle antagligen närma sig en totalt jämn fördelning ju flera strålar det finns i systemet. Resultatet av den verkliga testdatan från Gisip visar en signifikant högre fördelning på den främre högra flaskan, medans den bakre högra har en något lägre och de andra två flaskorna har en jämn fördelning. Anledning till denna snea fördelning beror framförallt på konstruktionen av kuben. Kuben är inte tillverkad med precisionsverktyg och därför kan kubens dimensioner vara något annorlunda än vad applikationens kub är. Gisips kub har även svetsfogar, popnitar och andra liknande ojämnheter som kan störa resultatet.

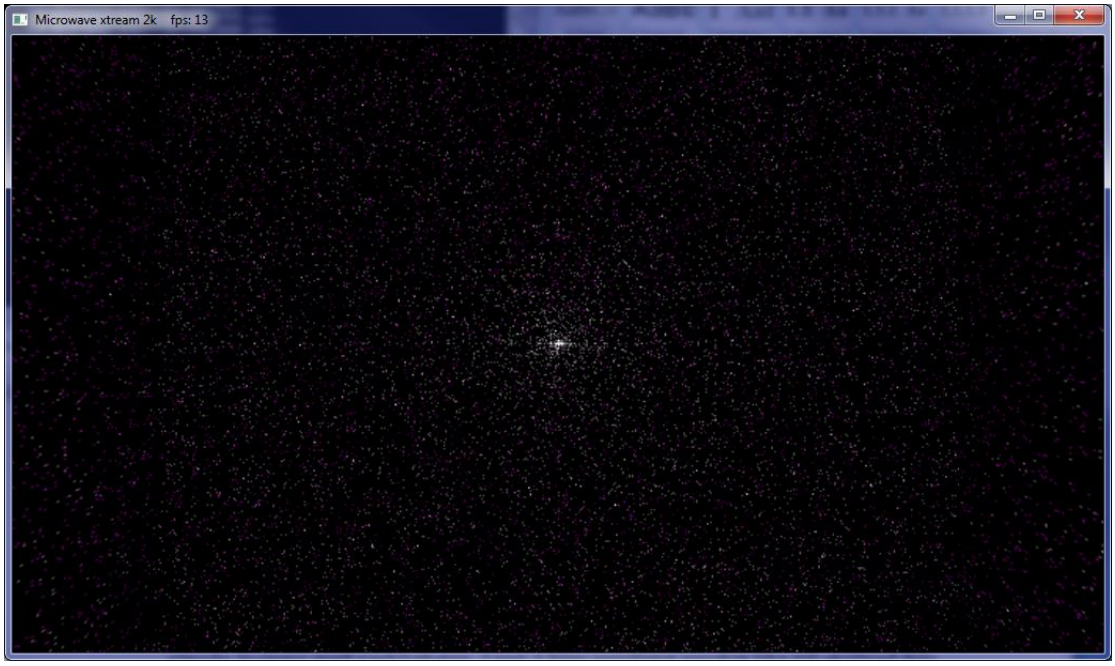
I det andra testet är tre flaskor utplacerade längst mittlinjen. Här kan man tänka sig att mittenflaskan skall ha större fördelning mot sig då den är rakt framför magnetronens mynning. Applikationen demonsterar ganska tydligt detta där mittenflaskan har en hög procentuell andel medans de yttre flaskorna har en nästan exakt likadan, fast lägre, andel. Gisips resultat visade liknande egenskaper, framförallt det första testfallet. I andra testfallet så ökade temperaturen i den vänstra flaskan till en mycket högre grad än flaskan till höger. Detta orsakades antagligen av att själva flaskan blivit mindre nedkyld än vad den högra flaskan var, och därför hjälpte det till att värma flaskan vid upphettningen. Detta problem löstes genom att mellan varje test sänka ner alla flaskor i en hink med vatten för att låta dem alla få en likvärdig temperatur. När tredje testfallet kördes höll sig temperaturen jämnare igen. I jämförelse med applikationens resultat så var det en mycket större distriubering mot mittenflaskan men i övrigt var resultatet likadant. Detta kan bero på att Gisips utlopp för magnetronen är mera riktad mot mitten än vad applikationens spridning är.

Tredje testets utplacering av flaskorna var på motstående vägg av magnetronen, samt att de var på halva höjden av kuben. Detta test borde ge samma fördelning av strålarna men effekten borde vara högre vilket ger högre temperaturer. Applikationen gav nästan exakt samma resultat som det tidigare testet, det vill säga en totalt jämn distriubering förutom mittenflaskan då den har en högre koncentration. Testen från Gisip gav ett väldigt fint resultat och det genomsnittliga resultatet visar till stor grad samma resultat som applikationens. Resultatet visade även att temperaturen steg betydligt mer än det tidigare testet.

I det fjärde och sista testet var flaskorna utplacerade likadant som i det tredje testet med den enda skillnaden att de låg längst upp i kuben. Anledningen till detta test var att inloppet till magnetronen verkade tyngas ner av sin egen vikt, vilket i så fall skulle betyda att magnetronen förskjuter strålarna något uppåt. Applikationen gav resultatet att det var en näst intill jämn fördelning men med en något högre fördelning mot mittenflaskan. När första testet i Gisips kub kördes så blev mittenflaskan så varm att när korken togs av från flaskan så kokade den över. När vattnet har stigit till dessa temperaturer så kan man inte längre få ut korrekt data ur dem. Därför är vattnet i andra testet initialt kallare. Samma sak hände då igen och därefter beslöts att inte göra flera tester då det fanns risk för skador.

4.3.2 Analys av systemet

Applikationen gav en totalt jämn fördelning av alla strålarna i systemet utan avvikelser. I en skärmdump från applikationen så kan man se var i kuben strålarna kolliderar.



Figur 20 Skärmdump från applikationen

De lila prickarna indikerar en stråle som precis har passerat en flaska för att sedan kollidera med kuben. De vita prickarna är kollisioner mot kuben med strålar som inte har kolliderat med en flaska sen senaste kollision mot kuben. Denna bild visar framsidan av kuben, alltså den sida som magnetronen pekar mot. Man kan här se att det är en starkare ansamling av strålar direkt motstående till magnetronen, och att runt omkring jämnar fördelningen ut sig. En anledning till varför det är så jämnt fördelat är att kuben, till skillnad från Gisips, uppbyggd matematiskt och har exakta dimensioner. En annan anledning är att applikationen inte tar i åtanke effekter som refraktion (när strålarna passerar flaskorna), interferens och diffraktion. Ett ojämnt mönster kan dock tyckas urskilja i kollisionerna. Detta beror på att strålarna får en slumpad riktning och ju färre strålar desto större håligheter kan uppstå, men ju fler strålar som används desto mer jämn blir fördelningen.

5 Slutsatser

5.1 Resultatsammanfattning

Studier kring projektet har gjorts gällande mikrovågor och realtidsljussättning, detta tillsammans med diskussioner med Gisip har lagt grunden för implementeringen av applikationen. Själva applikationen modellerar en kub som står hos Gisip. Applikationen skapar texturer till kuben, samt räknar ut antalet skärningar med sfärer som har ställts ut vid utvalda positioner. Dessa positioner testades därefter med den riktiga kuben på Gisip, genom att ställa ut flaskor fyllda med vatten. Man kunde på detta sätt räkna ut en procentuell distriubering av strålarna, för att sedan jämföra dessa med datan från applikationen. Resultatet visade att simulationen endast fungerar approximerat och att man bara kan påvisa hur systemet borde vara under optimala förhållanden. Verkligheten kan dock vara långt ifrån optimal och därför ge stora variationer i resultatet.

5.2 Diskussion

Som tidigare nämnt är det svårt att simulera verkligheten. Många olika faktorer kan tillsammans kraftigt påverka resultatet negativt och få resultatet att tyckas vara slumpmässigt. I det här arbetet hade Gisips kub popnitar, svetsskarvar och skeva dimensioner. Detta visade sig kunna ge stora variationer om hur resultatet blev. Trots detta kunde man påvisa approximationer om hur kuben skulle ha betett sig i optimala förhållanden, vilket kan vara ett bra verktyg för att simulera och utesluta värsta-fall-scenarion.

Schmitz och Kobbelt (2006) har tidigare påvisat att det är fullt möjligt att simulera mikrovågors spridning. Den stora avvägningen gällande precisionen i simuleringen sker i åtanke om den ska vara i realtid eller inte. Denna applikation kördes i realtid och den klarade precis gränsen på cirka 15 bilder per sekund. För att åstakomma detta resultat användes CUDA vid uträkningarna (Nvidia, 2009).

CUDA var ett bra verktyg som passade perfekt till detta problem. Styrkan ligger i att varje stråle kan köras parallellt med varandra och detta utnyttjas via GPU:n som får ansvaret att utföra alla beräkningar. Nackdelen är att datan först måste kopieras till GPU:n och sedan kopieras tillbaka till CPU:n för att någon skall kunna utföras. Detta är kostsamt och för att få prestandavinst så måste algoritmen läggas upp så att detta utförs så sparsamt som möjligt. Hade applikationen varit mera geometrifylld så hade en stackless KD-tree traversal (Popov, 2006) kunnat användas. Det hade dock antagligen endast sänkt prestandan i denna applikation.

Björn Karlsson gjorde en liknande applikation till Gisip parallellt med detta arbete. Karlssons arbete hade dock inte kravet på realtid och lösningen använde sig av photonmappning istället för en ray tracer. Detta leder till att simuleringen kan få mycket högre precision och ett mer korrekt resultat. Den riktigt stora styrkan hade legat i att en sådan simulering skulle kunna använda sig av interferens och diffraktion, vilket denna applikation inte skulle kunna åstakomma i realtid med dagens teknik. Eftersom denna simulering var så liten så var inte skillnaden stor i precision mellan realtids- och icke-realtidssimulering. Hade arbetet legat på en större simulering så hade skillnaden kunnat varit mycket större.

5.3 Framtida arbete

Gisips slutgiltiga mål med arbetet är att ha ett verktyg där de kan simulera mikrovågornas spridning när de skapar prototyper av deras produkter. Detta på grund av att de vill slippa skapa prototyperna fysiskt varje gång. Detta arbete tillsammans med Björn Karlssons arbete är ett delsteg mot detta verktyg.

För att vidareutveckla detta så hade en applikation kunnat skapas som till exempel körs på .NET för GUI. Denna applikation skulle till exempel kunna läsa in CAD-modeller för att sedan köra simuleringar på dem. Applikationen skulle då kunna vara uppdelad i två delar. Ena delen består av en realtidssimulering som ger ständig feedback på uppdateringar som sker i systemet. Detta skulle bli en approximerad simulering som inte har så hög precision. Känner man sig nöjd så kan man gå in i del två. Del två består då av Karlssons teknik som kör en mer fullständig och högprecis simulering på modellen.

Ett ytterligare tillägg till applikationen skulle kunna vara att lägga till simulering över tid samt rörliga delar som till exempel ett rullband där man kan lägga en låda som skjuts framåt i systemet. Man kan då genom hela simuleringen få fram ett resultat där man kan stega igenom simuleringen och se vart temperaturerna höjs. Till detta skulle man då dock behöva implementera luftflöde och ventilation för att få ett korrekt resultat

En annan använding för detta skulle kunna vara vid vågrörelsebaserade partikelsystem för att producera intressanta visuella effekter som uppstår vid till exempel interferens och diffraktion, vid kollision med diskreta objekt.

Ytterligare användningsområden kan även inbegripa de som behandlar varierande mängder energiansamlingar från olika typer av elektromagnetisk strålning, även något så vanligt som solsken i till exempel ett växthus. Detta system kan användas för att upptäcka områden som är mer eller mindre utsatta av solsken och låta dessa blommor växa bättre. Man skulle då kunna koppla detta till en genetisk algoritm för att optimera blommornas position i växthuset.

Referenser

Alonso, M. & Finn, E.J. (1992), *Physics*. Wokingham: Addison-Wesley

Alphonce, R. et al. (1998), *Fysik för gymnasieskolan B* (sid. 216-253). Stockholm: Natur och kultur

Mattson, P. (2003), *Realtids-strålföljning med geometriska primitiver på programmerbara grafikprocessorer* Hämtad 2010-02-05 från ”http://his.diva-portal.org/smash/record.jsf?searchId=1&pid=diva2:3196”

Allgyer, M. (2008), *Real-time Ray Tracing using CUDA* Hämtad 2010-03-14 från “http://www.handsfreeprogramming.com/masters/masters_project_report.pdf”

Nvidia (2009), *Nvidia Cuda Programming Guide 2.3* Hämtad 2010-02-05 från “http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.3.pdf”

Parker, S. & Martin, W. & Sloan, P.J. & Shirley, P. & Smits, B. & Hansen, C. (1999) Interactive ray tracing. In *Proceedings of the 1999 Symposium on interactive 3D Graphics*, sid. 119-126

Ploeg, A.J. (2006), *Interactive Ray Tracing the replacement of rasterization?* Hämtad 2010-02-01 från “http://www.few.vu.nl/~kielmann/theses/avdploeg.pdf”

Popov, S. (2006), *Stackless KD-Tree Traversal for Ray Tracing on Graphics Hardware* Hämtad 2010-02-01 från http://graphics.cs.uni-sb.de/index.php?id=popov_06_master_thesis

Ryoo, S. & Rodrigues, C. I. & Baghsorkhi, S. S. & Stone, S. S. & Kirk, D. B. & Hwu, W. W. (2008), Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, sid. 73-82

Schmitz, A. & Kobbelt, L. (2006), Wave Propagation Using the Photon Path Map. In *Proceedings of the 3rd ACM international Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor and Ubiquitous Networks*, sid. 158-161

Schmitz, A. & Wenig, M. (2006), The Effect of the Radio Wave Propagation Model in Mobile Ad Hoc Networks. In *Proceedings of the 9th ACM international symposium on Modeling analysis and simulation of wireless and mobile systems*, sid. 61-67

Sherrod, A. (2008), *Game graphics programming*. Boston, MA: Course Technology/Charles River Media/Cengage Learning

St-Laurent, S. (2004), *Shaders for game programmers and artists* (sid. 155-178). Boston, MA: Thomson/Course