HÖGSKOLAN SKÖVDE

Institutionen för kommunikation och information
Examensarbete i datavetenskap 30hp
C-nivå
Vårterminen 2008

Ambient Occlusion i Realtid

David Dikman

Ambient Occlusion i Realtid

Examensrapport inlämnad av David Dikman till Högskolan i Skövde, för Kandidatexamen (B.Sc.) vid Institutionen för kommunikation och information. Arbetet har handletts av Mikael Thieme.

30.6.2008

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: _____

Ambient Occlusion i Realtid

David Dikman

Sammanfattning

Ambient Occlusion är en teknik för ambient ljussättning i digitala tredimensionella scener. Sådana scener ljussätts vanligtvis med en konstant mängd ambient ljus på samtliga ytor oberoende av ytornas vinkel och position gentemot olika ljuskällor i scenen. Detta ger ett platt och kalt intryck och utan vidare ljussättningstekniker är det ytterst svårt att urskönja detaljer i scenen. Ambient Occlusion åthjälper detta genom att reducera mängden ambient ljus i vissa delar av scenen. Ambient ljus är en enkel approximation av det reflekterade diffusa ljuset som antas nå nästan överallt i scenen. Genom att sänka det ambienta ljuset på punkter i scenen med tät eller täckande geometri så ger Ambient Occlusion ett intryck av att det sekundära diffusa ljuset ej når dessa platser.

Pappret undersöker en äldre variant av Ambient Occlusion där mängden ambient ljus beräknas statiskt för en scen och sparas i texturer. Vidare undersöks nyare metoder där mängden ambient ljus beräknas dynamiskt på den renderade scenen i Pixel Shaders. Det senare tillvägagångssättet kallas Image Based Ambient Occlusion eller Screen Space Ambient Occlusion. Detta nya tillvägagångssätt jämförs mot den traditionella angreppsvinkeln med förberäknade texturer. Teknikerna utvärderas och jämförs mot varandra i avseende på tids- och minneskomplexitet, enkelhet och visuellt resultat utöver specifika egenskaper för de enskilda teknikerna.

Arbetets resultat beskrivs i slutet av rapporten. I resultatet presenteras hur shaderteknikerna pga sina brister inte är applicerbara i alla scener samt att förberäknad ambient occlusion är otymplig att applicera i många projekt.

Nyckelord: Ambient Occlusion, Screen Space Ambient Occlusion, Image Based Ambient Occlusion, SSAO, Obscurance, Proximity Shadow

Innehållsförteckning

1 Introduktion 4

2 Bakgrund 5

2.1 Ljussättning 5

2.2 Behov 5

2.3 Ljus 5

2.4 Global Illumination 5

2.5 Ljussättning i Realtid 6

2.5.1 Direct Illumination 6

2.5.2 Approximationer av Indirekt Ljussättning 7

2.5.3 Shaders 7

2.5.4 Post Process och Screen Space 8

2.6 Ambient Occlusion 8

2.6.1 Ambient ljus 9

2.6.2 Ambient Occlusion 9

2.6.3 Förberäknad Ambient Occlusion 11

2.6.4 Ray Tracing 11

2.6.5 Occlusion Queries 12

2.7 Relaterade arbeten 13

2.7.1 Fast precomputed Ambient Occlusion for proximity shadows 13

2.7.2 nVidias Surface Elements Technique 14

2.7.3 Hardware Accelerated Ambient Occlusion Techniques on GPU 14

3 Problem 15

3.1 Problemformulering 15

3.2 Delmål 1: Utveckla tekniker 15

3.3 Delmål 2: Utvärdera teknikern 15

4 Metod 17

4.1 Metod för Delmål 1: Utveckla tekniker 17

4.2 Metod för Delmål 2: Utvärdera tekniker 17

5 Genomförande 19

5.1 Genomförande av Delmål 1: Utveckling 19

5.1.1 Förberäknad Ambient Occlusion 19

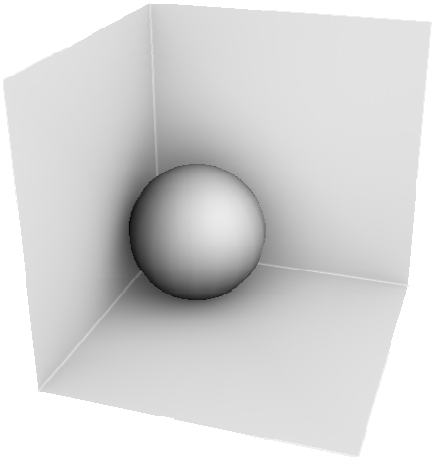
5.1.2 Screen Space Ambient Occlusion 23

5.2 Genomförande Delmål 2: Utvärdering 26

5.2.1 Förberäknad Ambient Occlusion 26
5.2.2 Screen Space Ambient Occlusion 28
6 Resultat 30
6.1 Resultat för Delmål 1: Utveckling 30
6.1.1 Förberäknad Ambient Occlusion 30
6.1.2 Crysis SSAO 30
6.1.3 Crease SSAO 30
6.1.4 Depthtexture Unsharpening SSAO 30
6.2 Resultat för Delmål 2: Utvärdering 30
6.2.1 Förberäknad Ambient Occlusion 30
6.2.2 Screen Space Ambient Occlusion 32
6.3 Analys 35
6.3.1 Visuellt resultat 35
6.3.2 Förberäknad Ambient Occlusion 36
6.3.3 Screen Space Ambient Occlusion 36
7 Slutsats 37
7.1 Diskussion 37
7.2 Fortsatt arbete 37
8 Tillkännagivanden 39
9 Referenser 40
10 Appendix 41

1 Introduktion

För att tredimensionella simuleringar skall ge ett verklighetstroget intryck så är ljussättning mycket viktigt (Akenine-Möller & Haines, 2004). Det räcker sällan med att ett objekt i en film eller ett spel ser geometriskt korrekt ut för att det skall ge ett trovärdigt intryck utan det krävs mer. Objektet måste framförallt interagera på rätt sätt med omgivningen och där spelar ljussättningen en stor roll. För att exempelvis en bil skall uppfattas stå på marken bör en skugga finnas under, likaså skall ett blankt föremål reflektera omgivningen. Ljussättningsdetaljer såsom dessa är nödvändiga för att scenen skall uppfattas verklig. I offlinerenderingar används begreppet Global Illumination som är en samling tekniker för att på ett mer fysiskt korrekt vis ljussätta tredimensionella scener. Dessa tekniker är dock i nuläget allt för komplicerade och tidskrävande för att kunna applicera i en realtidsapplikation och därför måste approximationer användas. Ambient Occlusion är en sådan approximation vilken simulerar den avsaknad av ljus som kan uppstå i skrymslen och hörn pga att en lägre mängd ljus reflekteras in till sådana ytor. Figur 1 nedan visar effekten av Ambient Occlusion i en scen.



Figur 1. Exempel på Ambient Occlusion i en scen

Rapportens syfte är att utveckla, utvärdera och jämföra ett antal tekniker för Ambient Occlusion. Specifikt kommer en teknik för att beräkna Ambient Occlusion och spara denna i texturer jämföras med ett antal tekniker baserade på renderade djupvärden, s.k. Screen Space Ambient Occlusion-tekniker. För att jämföra teknikerna kommer deras för och nackdelar samt implementationssvårighet, minneskomplexitet, visuella resultat och enkelhet att undersökas. För att kunna utföra en jämförelse av exempelvis visuellt resultat och enkelhet utvecklas teknikerna. För att motivera ämnets användbarhet förklaras dess bakgrund ur perspektivet Global Illumination och ljussättning för realism. Rapporten presenterar varför Ambient Occlusion utvecklats och varför tekniken är användbar.

Arbetet har genomförts genom att teknikerna har utvecklats och testats. Utvecklingen har skett i programmet RenderMonkey med shaderspråket HLSL samt i C# med APIet DirectX . Teknikernas visuella resultat har sedan undersökts och jämförts gentemot korrekta Ambient Occlusionresultat. Teknikernas fasta egenskaper, minnes- och tidskomplexitet vid beräkning och tillämpning resoneras och genom teknikernas utveckling ges underlag för resonemang kring deras visuella resultat och enkelhet.

2 Bakgrund

Följande kapitel förklarar bakgrunden till begreppet Ambient Occlusion, ursprunget och motiveringen. Förklaringen utgår utifrån behovet av ljussättning och svårigheterna med fotorealistisk ljussättning i datorgenererade scener. Vidare diskuteras möjligheter för att kringgå dessa svårigheter och förenklingar som kan göras utan att det visuella resultatet försämras oöverkomligt. Därefter diskuteras dagens tekniker för att simulera mer realistisk ljussättning i realtid vilket leder till de s.k. SSAO teknikerna. Till sist presenteras relaterade arbeten som p.g.a. avvikande angreppsvinklar valts bort från rapportens huvudfokus.

2.1 Ljussättning

För att motivera användandet av Ambient Occlusion måste också användandet av ljussättningstekniker generellt i 3D grafik diskuteras. Nedan ges en beskrivning på varför ljussättning tillämpas och övergripande hur den tillämpas.

2.2 Behov

För att en tredimensionell modell skall ge ett verklighetstroget intryck krävs det att den inte bara ser geometrisk korrekt ut utan också att den ljussätts på ett korrekt sätt (Akenine-Möller, et al., 2004). Detta är något framförallt filmbranschen lägger stort fokus på när de sätter samman filmade klipp med datoranimerade klipp för att skapa miljöer och effekter som annars vore svåra om inte omöjliga att producera. För att dessa datoranimerade klipp skall kunna lura betraktaren till att tro att de tillhör det filmade klippet är det avgörande att ljussättningen stämmer överrens. Detta gör att filmbranschen i mångt och mycket driver på utvecklingen av nya tekniker för ljussättning för datorgenererade modeller och scener. De mest rättframma och korrekta teknikerna som används är dock mycket krävande och att rendera vissa klipp kan ta dagar för flertalet datorer tillsammans att producera. Därför är det viktigt att använda tekniker som kan sänka processens åtgångstid (Gear6, 2007, Landis, 2002). Av dessa tekniker kommer en del till användning i andra branscher som exempelvis i datorspelsbranchen. I datorspel är dock kraven på renderingshastighet långt mycket högre, att producera en stillbild av en scen i ett realtidsspel inte kräva mer än en trettiondels sekund. I båda branscher är konceptet ljus dock lika viktigt och förenklingar och approximationer till verklig ljussättning används flitigt.

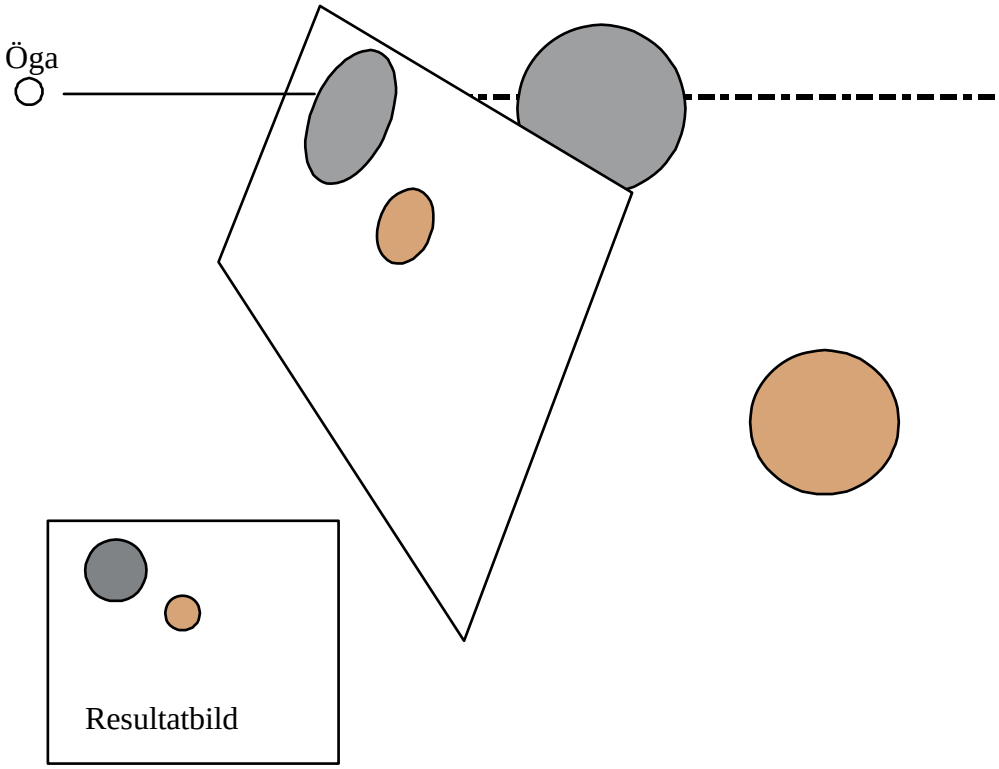
2.3 Ljus

I verkligheten är ljus egentligen photoner som reflekteras runt i omgivningen för att till sist nå betraktarens öga (Akenine-Möller, et al. 2004). Ljussättning kan ses som problemet att räkna ut de vägar som photonerna från ljuskällan färdas. Genom att veta detta kan det utrönas vilken influens ljuset får på scenen och modellerna däri. Ljussättning kan ses som problemet att räkna ut de vägar som photonerna från ljuskällan färdas.

2.4 Global Illumination

Begreppet Global Illumination representerar en samling tekniker för ljussättning där ljussättningen för punkter i scenen beräknas med hjälp av de vägar som ljuset sprids över scenen med. En ofta tillämpad teknik för detta är Ray Tracing. Ray Tracing baseras på att kameran ställs relativt till resultatbilden och att strålar sänds genom varje punkt på bilden. Strålen fortsätter sedan genom bilden som skapar ett slags

fönster och det objekt i scenen som träffas först syns på den aktuella punkten i bilden. Figur 2 nedan visar ett exempel på hur Ray Tracing skulle kunna se ut.



Figur 2. Ray Tracing illustration

För mindre scener fungerar detta rätt bra om endast en stråle kan skickas per punkt i resultatbilden. För att få ett mer photorealistiskt resultat måste dock förloppet kompliceras ytterligare. I grundutförandet som beskrivits tas inte sekundära reflektioner eller ens ljus hänsyn till, något som kräver långt mycket mer arbete. Om också ljus skall beräknas för scenen måste strålen som sänds från ögat fortsätta att studsa från den punkten på objektet den träffar runt i scenen fram tills något slags tröskelvärde. Under tiden strålen reflekteras runt ackumuleras ljus vilket sedan läggs till på ursprungspunkten. Detta betyder framförallt att ytterst många strålar måste användas för varje punkt då det annars är starkt osannolikt att en liten ljuskälla i en stor scen träffas. Tekniken blir på grund av utförandet starkt beroende på mängden geometri och ljussättningen i scenen vilket gör tillämpning i exempelvis realtid olämplig.

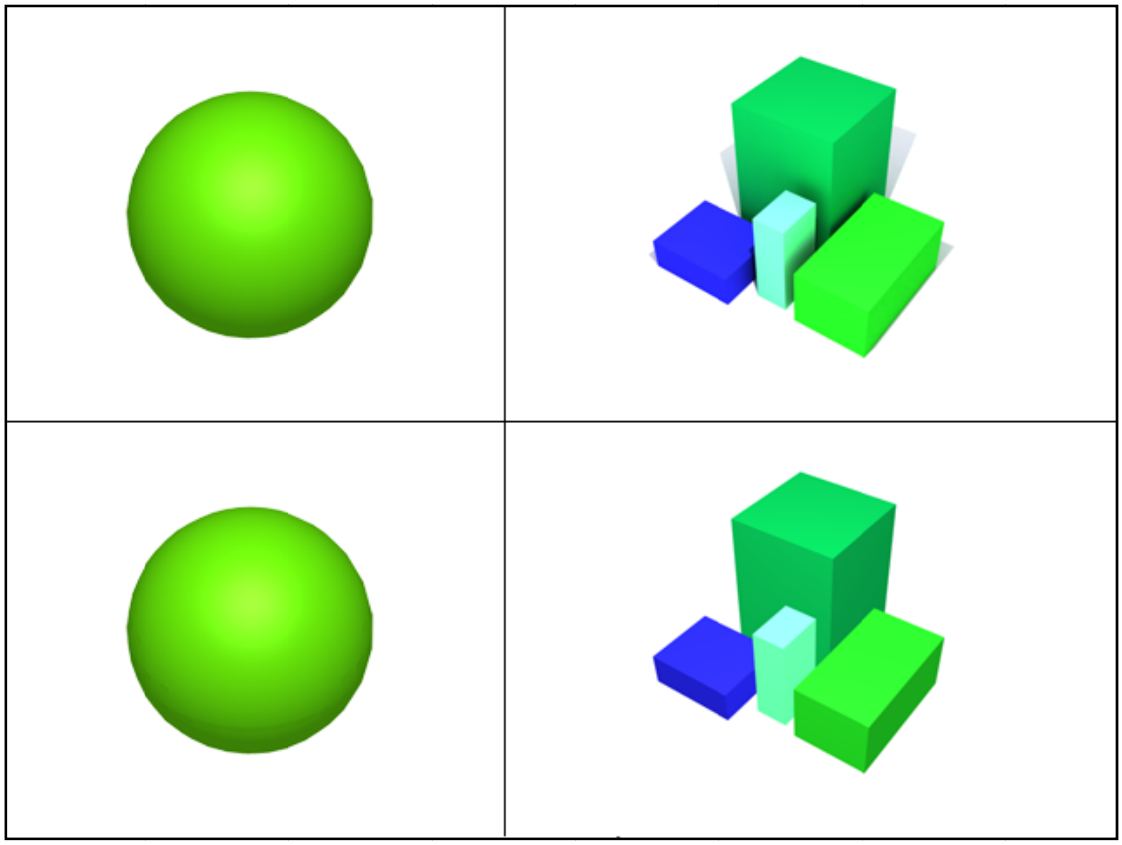
2.5 Ljussättning i Realtid

Nedan beskrivs till skillnad från i tidigare stycken tekniker för ljussättning som vanligtvis appliceras i realtidsapplikationer.

2.5.1 Direct Illumination

Direct Illumination (direkt ljussättning) betyder att endast den aktuella ljuskällan samt data för punkten för vilken ljussättningen skall beräknas används i beräkningen. Detta är en otroligt snabb men felbenägen approximation av ljussättning. Den ger ett bra resultat såvida punkten exempelvis inte bör skuggas eller reflektera något. Ljussättningstekniken arbetar alltså endast på lokal ljussättning och förbiser återstoden av scenen till fördel för prestanda. Figur 3 nedan visar överst till höger exempel på en komplicerad scen där den indirekta ljussättningen ger klart mer djup i

bilden jämfört med samma scen ljussatt endast med direkt ljussättning nere till höger. I scenerna till höger märks dock inte skillnaden mellan den övre indirekt ljussatta sfären och den undre lokalt ljussatta sfären. Detta visar på att en scens komplexitet spelar stor roll för funktion och brister av indirekt och direkt ljussättning.



Figur 3. Kontrastbild för Indirekt och Direkt ljussättning. Övre bilderna visar scenerna ljussatta med Indirekt Ljussättning och de undre bilderna visar samma scener ljussatta med Direkt Ljussättning

2.5.2 Approximationer av Indirekt Ljussättning

Då direkt ljussättning allena ger ett väldigt platt och överdrivet exakt resultat används olika tekniker för att erhålla effekten av indirekt ljussättning. Ett exempel på detta är Depth Map Shadows (ST-Laurent, 2004) som skuggar scenen genom att undersöka vilka ytor ljuset belyser och de renderade ytornas läge gentemot dessa. Detta uppnås genom att scenens avstånd från ljuset renderas utifrån ljusets utgångspunkt (som om ljuskällan vore kameran i scenen). För varje punkt i den slutligen renderade scenen projiceras punkten till djupmappen (punkten transformeras till djupmappens rymd) och dess avstånd till ljuset jämförs med aktuell punkt i djupmappen. Om avståndet överstiger värdet i djupmappen ligger ytan bakom något och bör således skuggas. Denna teknik hjälper till att simulera de skuggor som annars går förlorade vid direkt ljussättning.

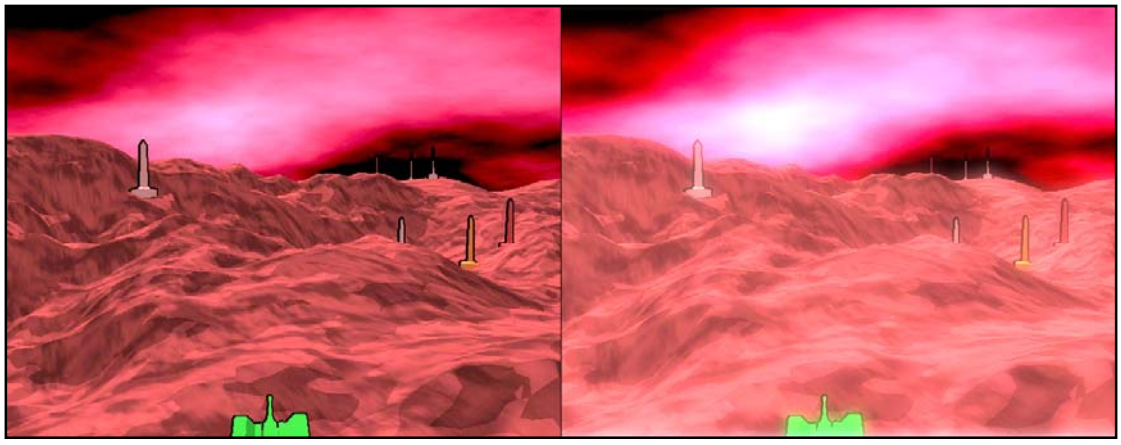
2.5.3 Shaders

Att utföra beräkningar och renderingar som exempelvis Depth Map Shadows på datorns processor skulle kräva mycket processorkraft men numera tillåter grafikkort att utvecklare kan använda grafikkortets processor istället. Grafikkortets processor är dessutom är optimerad för beräkningar på pixlar (fragment) och vertexar vilket gör att beräkningar som exempelvis de för Depth Map Shadows kan utföras långt mycket

snabbare än möjligt på datorns CPU. Detta görs möjligt genom begreppet Shaders. Shaders är små program som utvecklare kan koda och exekvera på grafikkortet som en del av den pipeline geometrin passerar genom när den renderas. I dagsläget finns två delar i pipelinen öppna för utvecklare och dessa kallas Fragment eller Pixelshaders och Vertex Shaders. Vertexshaders manipulerar geometrin efter transformation och tillåter utvecklaren att på ett smidigt sätt per vertex i modellen generera data inför Fragment Shadern eller manipulera modellens utformning (per vertex). Fragment Shadern tillåter utvecklaren att manipulera varje fragment eller pixel innan dessa renderas till skärmen. Här kan punkter ljussättas eller manipuleras på en rad olika sätt exempelvis kan scenens skuggor beräknas här endast för de faktiskt renderade pixlarna. Denna öppning i pipelinen möjliggör en långt mycket snabbare manipulation av geometri än möjligt på datorns CPU. Framförallt är pipelinen optimerad för att endast den geometri som faktiskt syns behandlas då återstoden prioriteras bort innan Fragment Shadern. Nackdelen är dock att vertexar och pixlar i shaders endast bearbetas en åt gången och möjlighet att manipulera även andra pixlar och vertexar existerar inte. Detta gör det svårt att implementera faktisk indirekt ljussättning men många andra tekniker finns utvecklade för att på olika sätt gå runt denna begränsning.

2.5.4 **Post Process och Screen Space**

En mycket vanlig användning av framförallt Fragment Shaders är så kallade Post Process eller Screen Space effekter. Med detta menas att scenen renderas i olika steg och att datan som samlas in kan användas för att förvränga eller filtrera slutresultatet. Olika stilistiska effekter som exempelvis Bloom som visas till höger i Figur 4 nedan kan uppnås genom detta men också exempelvis ren ljussättning genom Deffered Shading där olika bitar av ljussättningsdata sparas i texturer och används för att ljussätta scenen per pixel i ett slutpass. Gemensamt är är att data renderas till texturer och återanvänds för att manipulera den renderade scenens utseende. Detta kallas oftast för Screen Spaceeffekter då effekterna arbetar på pixlar på den slutgiltiga skärmbilden och då alltså i skärmens rymd snarare än i scenens världsrymd. Figur 4 nedan visar ett exempel på Post Processeffekten Bloom där scenen renderas och de ljusa partierna får blöda över till de mörkare.



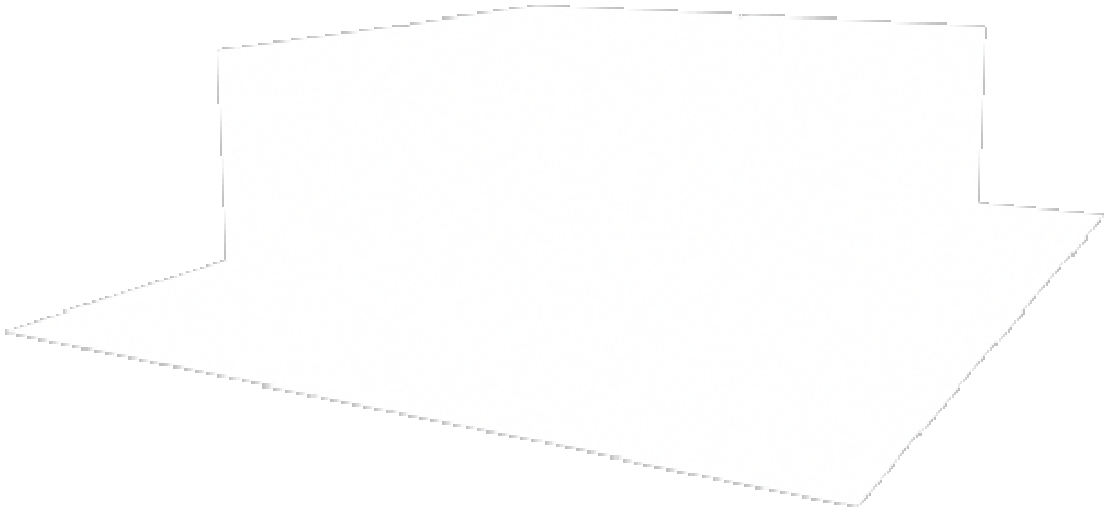
Figur 4. Bloom Exempel, Bilden till vänster visar scenen i grundutförande och bilden till höger scenen filtrerad med s.k. Bloomeffekt

2.6 **Ambient Occlusion**

För att kunna beskriva Ambient Occlusion förklaras först själva begreppet Ambient Ljus, därefter nämns lite om olika möjligheter för beräkning av Ambient Occlusiondata.

2.6.1 **Ambient ljus**

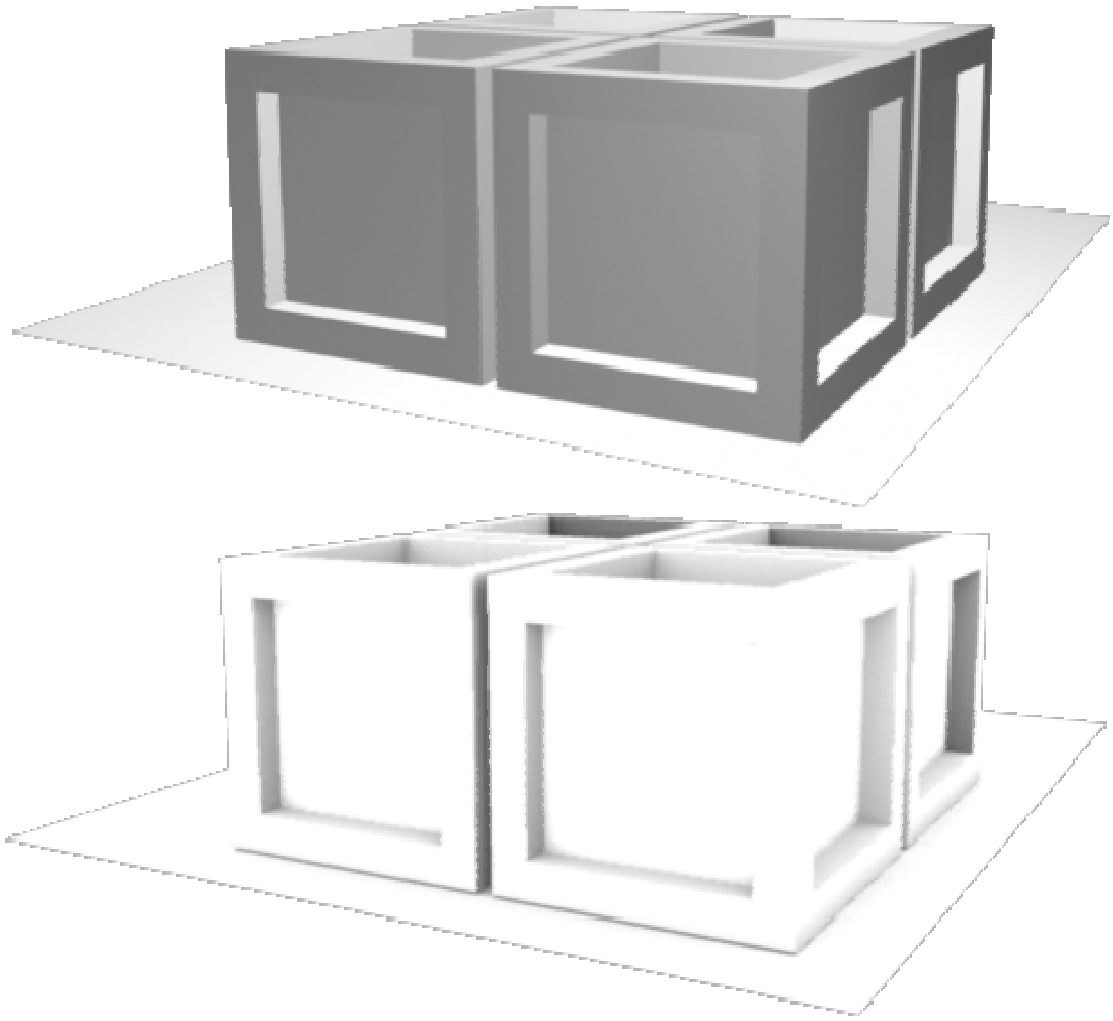
För att förklara Ambient Occlusion måste begreppet Ambient ljus klargöras. Ambient ljus används i direkt ljussättning där inget indirekt diffust ljus finns. Indirekt diffust ljus uppstår annars i Global Illuminationtekniker då ljuset tillåts reflekteras runt i scenen (Malmer, Malmer, Assarsson, & Holzschuch, 2007, Zhukov, et al., 1998, Kontkanen & Laine, 2005). Detta gör att scenen normalt blir mycket ljusare även på platser där ljuset inte direkt når fram. I direkt ljussättning approximeras denna mängd reflekterat diffust ljus med hjälp av det ambienta ljuset. I den enklaste formen adderas endast en viss mängd ambient ljus till scenen konstant över alla ytor oavsett position eller riktning vilket klart inte överensstämmer med verkligheten där punktens omgivning bestämmer mängden indirekt diffust ljus. Utseendet av en sådan konstant ljussättning ger utan ytterligare ljussättning (diffus- och/eller phongljus) ett ytterst platt och svårtolkat utseende. Detta visas i Figur 5 nedan där ett par lådor står ovanpå en platta mot vit bakgrund. En kant har lagts till för att ens kunna urskilja modellen från bakgrunden, återstoden av bilden flyter samman i brist på vidare ljussättning. Att ljussätta liksom bilden visar är givetvis långt ifrån genomtänkt och exemplet används endast för att exemplifiera den kontrast bruket av Ambient Occlusion ger mot teknikens frånvaro.



Figur 5. Ambient Ljussättning

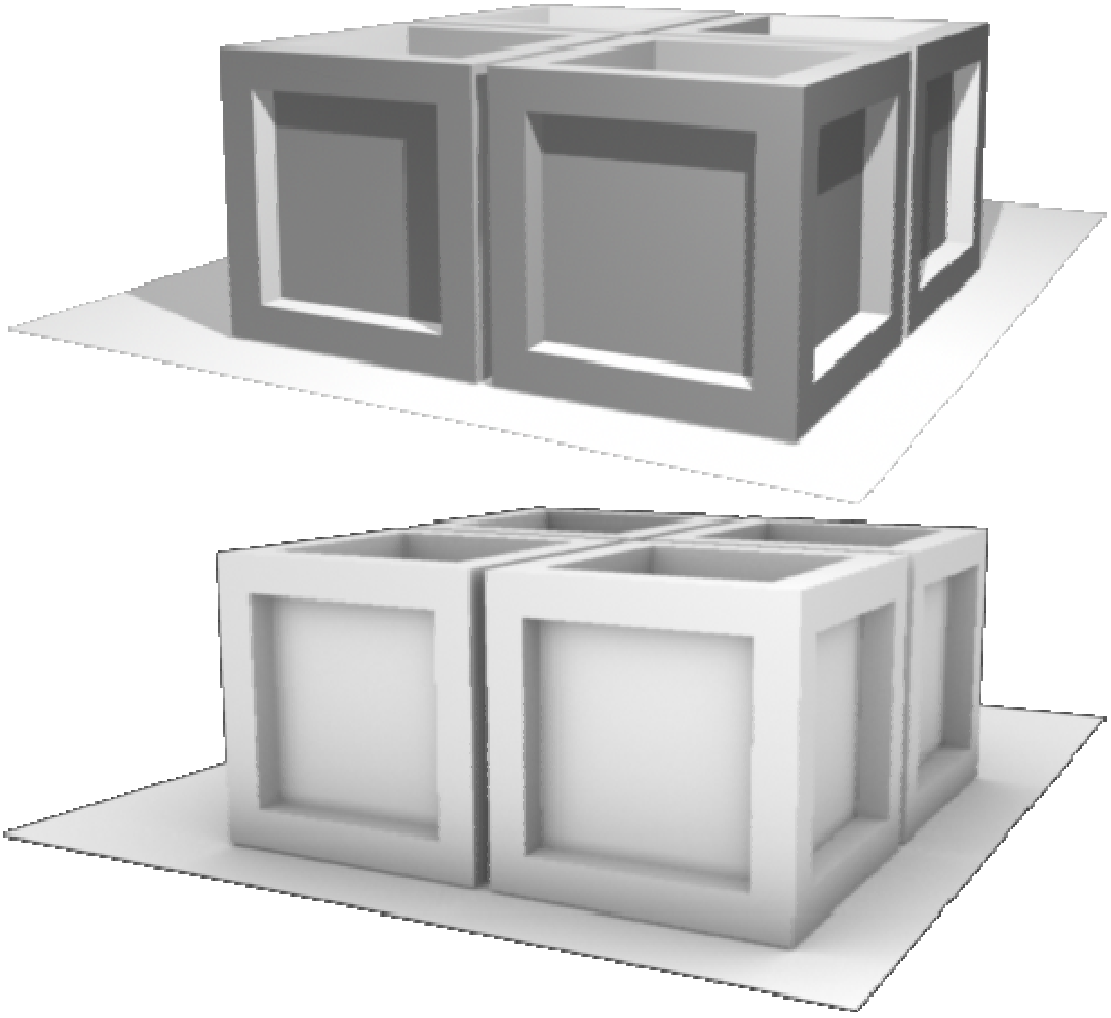
2.6.2 **Ambient Occlusion**

Ambient Occlusion används för att förbättra den konstanta approximationen av ambient ljus och kan tillämpas på olika sätt. Gemensamt är dock att det ambienta ljuset reduceras för punkter på modellen där geometrin på något sätt är skymd av omkringliggande geometri (Zhukov, et al., 1998). Detta baseras på att en punkt som är skymd sannolikt erhåller en lägre mängd diffust ljus då färre ytor kan reflektera rakt in mot punkten. Nedan i Figur 6 visas ett exempel på vanlig lokal ljussättning med endast phongbelysning i kontrast mot samma scen ljussatt med Ambient Occlusion. Där syns tydligt hur mycket Ambient Occlusion bidrar till att ge naturliga skuggor.



Figur 6. Jämförelse mellan phongbelyst scen (övre) och scen ljussatt med Ambient Occlusion (undre)

I Figur 7 nedan visas skillnaden mellan en phongbelyst scen med skuggor samt en phongbelyst scen med Ambient Occlusion. Scenen med Ambient Occlusion ger där ett varmare och mer mjukt utseende emedan den phongbelysta scenen ger ett rakt och onaturligt intryck.



Figur 7. Jämförelse mellan en scen ljussatt med phong och med skuggor (övre) samt med Ambient Occlusion (undre)

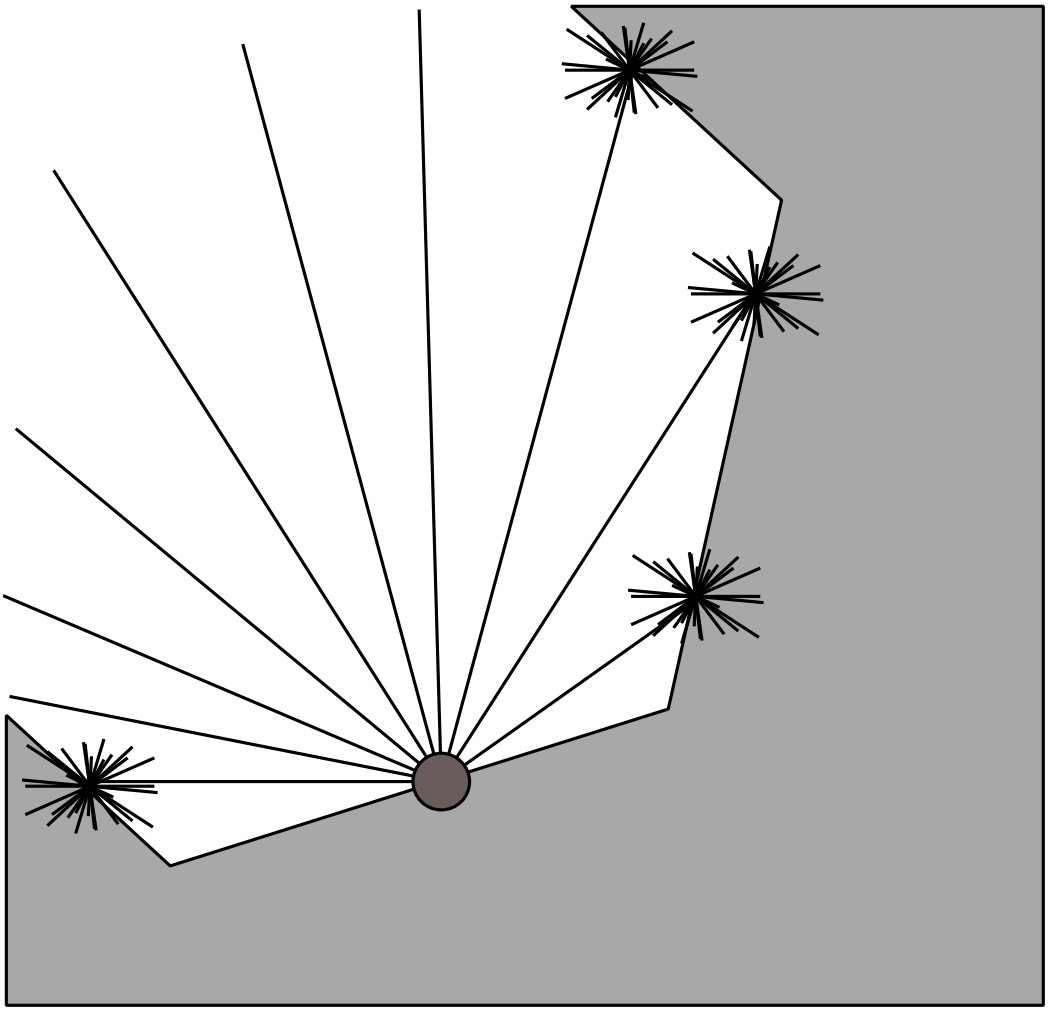
2.6.3 **Förberäknad Ambient Occlusion**

Då Ambient Occlusion såsom presenterad i (Zhukov, et al., 1998) endast är beroende på geometri och inte på varken kameravinkel eller ljussättning så kan Occlusiondata med fördel förberäknas och lagras för statiska scener. Nackdelen är givetvis att om något i scenen förflyttas eller transformeras är datan inte längre korrekt men just för statiska scener är det en möjlighet att förberäkna. Nedan beskrivs två tillvägagångssätt för att beräkna Occlusiondata och denna lagras med fördel i antingen texturer eller som separata värden per vertex i modellen. Det senare kräver en högt tesselerad modell emedan texturer tillåter mer arbitära utseenden på modellen. Texturer innehar dock nackdelen att komplicerade modeller kräver mycket stora textuer för att kunna lagra data för samtliga små ytor.

2.6.4 **Ray Tracing**

Den mest rättframma metoden till att undersöka en punkts blockering är att sända strålar från punkten över en hemisfär centrerad ovanpå punkten med riktningen i punktens normal. Avståndet till punkterna som träffas samt vinkeln mellan hemisfärens riktning och strålens avvikelse kan sedan användas för att uttröna hur mycket kollisionen skall inverka på blockeringen. Figur 8 nedan visar en något skymd

punkt. De strålar som träffar viktas framförallt med avståndet mellan normalen och strålen, faktorn Cos(V), vilket gör att de yttersta strålarna knappt bidrar till blockeringen. Att använda Ray Tracing för att utröna en punkts blockering ger fördelen att omgivningens inverkan kan kontrolleras starkt då vinklar etc kan undersökas. Nackdelen är att strålarna måste testas mot all geometri i scenen. Detta kan till mångt och mycket optimeras på många sätt men alternativet med Occlusion Queries som presenteras nedan kan ofta beräknas snabbare.

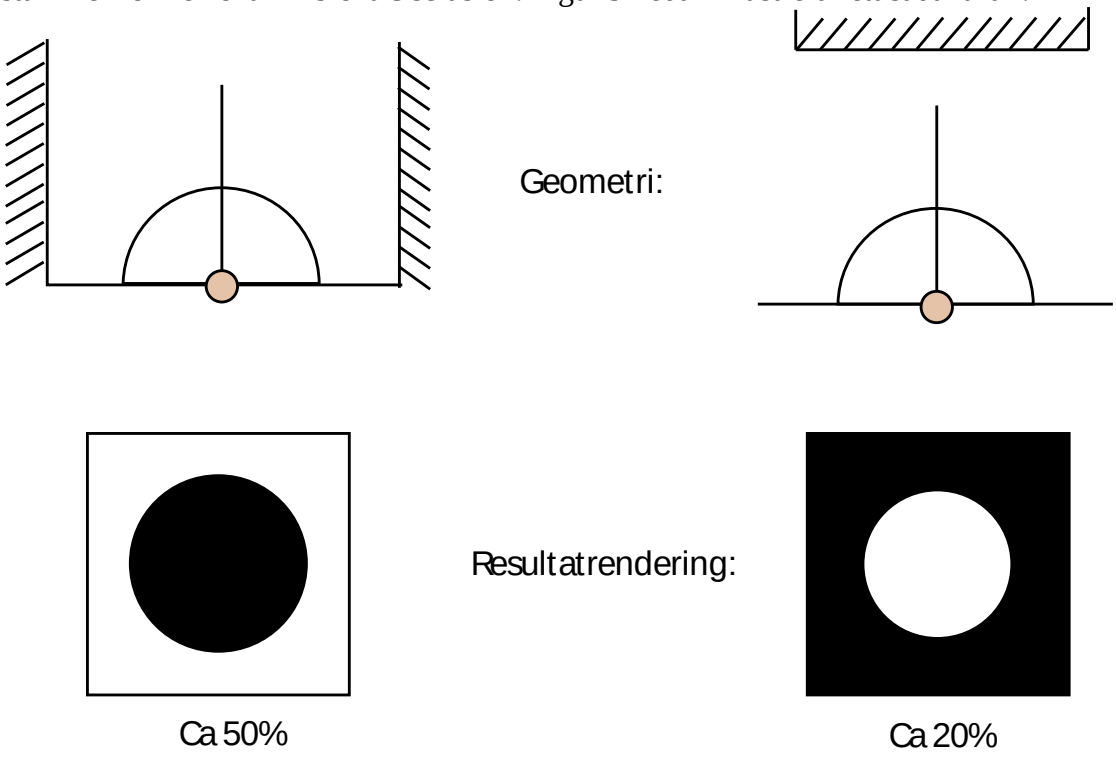


Figur 8. En något blockerad punkt

2.6.5 **Occlusion Queries**

Istället för att använda Ray Tracing och testa strålars kollision mot hela scenen kan Occlusion Queries användas. Scenen renderas i enklaste utförande utifrån punkten vars blockering skall beräknas, sedan hämtas från grafikkortet mängden synliga pixlar av omgivningen (Engel, 2005). Genom att endast rendera geometri inom ett visst avstånd (m.h.a. Far Clip Distance) kan då en punkts omgivning undersökas. Det värde för mängden synliga pixlar av scenen divideras med antalet möjligt synliga pixlar, d.v.s. exempelvis arean av skärmen (antalet pixlar på skärmen). Fördelen med detta är att grafikkortet är oerhört optimerat för att rendera scenen och att rendera en scen tar i många fall långt mycket mindre tid än att undersöka en stråles kollision mot hela scenen. Nackdelen är att tekniken i vissa fall ger inkorrekt resultat då grafikkortets rendering måste klippa viss geometri som ligger för nära, således kan vissa regioner erhålla för låg blockering beroende på normalens riktning. Vidare så har man på samma sätt mycket mindre kontroll över viktningen på geometrins fördelning över

den hemisfär som renderas. En punkts blockering beräknas endast på antalet synliga fragment och inte fragmentens fördelning över ytan. Detta betyder att en punkt omgiven av mycket geometri i pereferin (nederkanten av hemisfären) kan erhålla lika mycket blockering som en punkt något täckt rakt framifrån, något som inte alls stämmer för korrekt Ambient Occlusion. Figur 9 nedan illustrerar ett sådant fall.



Figur 9. Exempelfall där Occlusion Queries ger felaktigt resultat, fallet till vänster borde vara mer ockluderat då större andelen ljus blockeras framifrån

2.7 Relaterade arbeten

Sedan begreppet Ambient Occlusion fastställdes (Zhukov, et al., 1998) har ett antal förbättringar och varianter utvecklats där blockeringsdata antingen beräknas helt i realtid eller förberäknas och uppdateras i realtid. Nedan beskrivs några tekniker vilka ligger rapporten nära intill. Teknikerna som följer har dock inte tagits upp vidare i rapporten då rapporten snarare eftersträvar att påvisa de shaderbaserade teknikernas möjligheter än tekniker för Ambient Occlusion generellt. Teknikerna bör dock tas i beaktning för den som vill göra en mer bred undersökning på tillvägagångssätt för Ambient Occlusion generellt. Flera av följande tekniker tillhandahåller lösningar för mer korrekt Ambient Occlusion än de shaderbaserade teknikerna genom avvägningen att istället vara mer komplicerade. Av denna anledning nämns teknikernas utförande endast kort nedan, referenser nämns för teknikerna om vidare läsning är av intresse.

2.7.1 Fast precomputed Ambient Occlusion for proximity shadows

(Malmer, et al., 2007) visar att om data också för modellens externa blockering beräknas kan Ambient Occlusion också för dynamiska objekt appliceras i realtid. I ett förberäkningssteg lagras i en volymtextur modellens blockering på olika punkter omkring modellen. Genom att rendera volymtexturen med hjälp av ett slags kubiskt rutnät med samma transformation som modellen kan scenens Ambient Occlusion beräknas. Själva tekniken arbetar i följande steg:

- För varje blockerande objekt:

- Rendera baksidorna på rutnätet
- För varje beräknad pixel i en pixelshader:
  - Beräkna (genom någon slags deffered shading) världspositionen av pixeln
  - Konvertera världspositionen till en voxel (koordinat i volymtexturen)
  - Hämta Ambient Occlusionvärdet i volymtexturen med de beräknade koordinaterna
- Lägg samman blockeringsvärderna med multiplicerande blending där blockeringsvärdet används inverterat (1 – a)

2.7.2 **nVidias Surface Elements Technique**

(Bunell, 2005) presenterar en teknik för att beräkna blockeringsdatan i realtid och med hjälp av grafikkortet. Tekniken baseras på att det för varje vertex i modellen/scenen skapas ett ytelement som approximeras till en slags disk. Dessa får sedan skugga samt reflektera ljus mot varandra och på så sätt ge upphov till blockeringsdata och även indirekt ljus. Tekniken kräver dock en del förarbete dels för att skapa ytelementen och sedan också för att uppdatera ytelementen allteftersom scenen förändras. Vidare måste ytelement ordnas i en hierarki för de ytelement som möjligen kan skugga varandra vilken i sin tur lagras i en textur som passeras till grafikkortet. Tekniken kräver också en tämligen högtesselerad scen för att resultatet skall bli visuellt behagligt. Följande arbetssteg processas:

- Rendera scenen i grundutförande
- Uppdatera ytelementen
- För varje ytelement
  - Rendera med en pixelshader
    - Slå upp den aktuella hierarkin för ytelementet i texturen
    - För varje ytelement
      - Hämta data ur texturer innehållandes hierarkins ytelements data
      - Beräkna deras position och riktning gentemot det aktuella ytelementet
      - Med beräknad data beräkna influerande skuggning och ljusöverföring
- Lägg resultatet till den renderade scenen

2.7.3 **Hardware Accelerated Ambient Occlusion Techniques on GPU**

En mer intuitiv teknik för Ambient Occlusion beräknad på GPUn presenteras i (Shanmugam & Ariak, 2007) där tekniken till viss del använder en SSAO metod liknande de som behandlas i rapporten. SSAO metoden kombineras med en teknik för att beräkna Ambient Occlusion också för blockerande objekt på längre avstånd. Detta görs genom att ytor bryts ner i virtuella sfärer vilka renderas som billboards (plana texturer). Detta gör att sfärerna kan influera omgivningen på en per pixel basis vilket gör att tekniken är långt mycket effektivare sätt än ovan nämnda tekniker. Metoden ger dock en mindre detaljrik blockering men då objekten på avstånd inte erhåller någon större detaljblockering blir resultatet ändock godtagbart. Tekniken kräver viss förberäkning på CPUn för att ta beräkna sfärernas position och fördelning på objekten men övrig beräkning sker fortfarande på grafikkortet vilket ger hög prestanda och avlastning för CPUn.

3 Problem

3.1 Problemformulering

Problemställningen som rapporten behandlar är att utveckla och utvärdera tekniker för tillämpning av Ambient Occlusion i realtidsapplikationer. Sedan tekniken Ambient Occlusion presenterades (Zhukov, et al., 1998) så har ett flertal varianter och förbättringar av tekniken utvecklats. Den framförallt viktigaste egenskapen som har försökt förbättrats är att tekniken skall kunna appliceras på dynamiska scener, något som i grundutförandet inte är möjligt. Nyligen skedde framsteg i utvecklingen av teknikerna genom spelet Crysis (Mittring, 2007) där med hjälp av shaders ett visuellt resultat liknande Ambient Occlusion skapades i realtid utan att behöva bero på mängden geometri i scenen. Tekniken är på grund av sitt oberoende från geometrimängd oerhört mycket snabbare än tidigare Ambient Occlusiontekniker. Tillvägagångssättet sporrade fler försök till varianter av tekniken och flera alternativ utvecklades till att generera datan i realtid på grafikkortet.

Då ämnet fortfarande är nytt och det tidigare har publicerats i få källor (Mittring, 2007) är det intressant att gå igenom för och nackdelarna med detta nya tillvägagångssätt och att också ställa dessa nya shaderbaserade tekniker mot den förberäknade varianten av Ambient Occlusion kan ge en god motivation till båda teknikernas fortsätta utveckling och tillämpning.

För att arbetet skall vara framåtdrivande ligger fokus i ämnets framkant, de shaderbaserade teknikerna för Ambient Occlusion. Endast en teknik för förberäknad Ambient Occlusion undersöks därför även om alternativ presenteras. För att ge en vid presentation av möjliga shadertekniker för Ambient Occlusion undersöks tre stycken varianter av shaderbaserad Ambient Occlusion. De tre tekniker som undersöks är dels en s.k. Creaseshader (Fox, 2007) som baseras på skillnader i djup och normaler mellan pixlar. Dels så undersöks den shadervariant som används i spelet Crysis (Mittring, 2007). Slutligen undersöks en mycket enklare variant som bygger på att den lagrade texturen med djupvärden suddas ut och subtraheras från de riktiga djupvärdena (Pan, 08). Då ingen av nämnda tekniker finns publicerade implementationsmässigt kommer samtliga tekniker att behöva tolkas och tydas under utvecklingen.

3.2 Delmål 1: Utveckla tekniker

För att kunna jämföra de tekniker som finns i dagsläget måste dessa först utvecklas. Detta kräver vissa tolkningar och resultatet kan inte ses direkt entydigt för tekniken med tanke på att kreatörernas egna optimeringar och tydningar av teknikerna kan skilja från de som presenteras i rapporten. Utvecklingen ligger till grund för jämförandet framförallt i enkelhet där implementationen säger mycket om teknikernas svårighet. Vidare kan implementationen användas till att producera visuellt resultat som kan jämföras.

3.3 Delmål 2: Utvärdera teknikern

När teknikerna utvecklats utvärderas de och resultatet jämförs. Följande kriterier används för jämförandet:

- **Visuellt Resultet.** Det visuella resultatet om än subjektivt är det viktigaste resultatet från teknikerna då dessa har som syfte att efterlikna Global Illumination visuellt.
- **Tidskomplexitet.** Den kanske mest vanliga och viktiga aspekten av algoritmer är tidskomplexiteten (Weiss, 2006). Hur väl algoritmen skalar gentemot mängden geometri kommer därför att analyseras för samtliga tekniker, både gällande förarbete (för förberäknad Ambient Occlusion) och appliceringen i realtid.
- **Minneskomplexitet.** Oavsett vilken plattform en algoritm appliceras på är minnekrav ofta en viktig aspekt av algoritmen. Även om grafikkort som krävs i realtidsapplikationer för tredimensionella scener likt de som presenteras i rapporten har en stor mängd minne att bruka så finns ändock en gräns på minne. Just den förberäknade Ambient Occlusiontekniken kräver en avgörande mängd minne vilket gör minneskravet till en mycket viktig del i analysen.
- **Enkelhet.** En algoritm eller tekniks enkelhet betyder mycket för dess användbarhet. En ytterst otymplig eller svår algoritm blir inte eftertraktad att använda till skillnad från en algoritm som lätt kan implementeras och därigenom går snabbt att använda. Således är enkelheten en värdefull aspekt av tekniken att undersöka.

4 Metod

Kapitlet nedan beskriver det tillvägagångssätt som använts för att åstadkomma det resultat som rapporten presenterar. Metoderna nedanför har genomförts under projektets gång och deras genomförande och resultat beskrivs ingående i kapitlen följande detta.

4.1 Metod för Delmål 1: Utveckla tekniker

För att ge underlag för Delmål 2 måste teknikerna utvecklas. Då teknikerna inte är någon fast vetenskap eller exakta publicerade material utan ofta en kombination av många olika inslag måste en fristående utveckling ske för att resultatet i rapporten skall kunna bli korrekt för just de tekniker som utvecklas.

- **Implementation.** För att faktiskt kunna utveckla teknikerna för testning och utvärdering (se nedan) är implementationen kritisk. Implementationen genomfördes i två spår, dels implementationen för shaderteknikerna och dels implementatonen för den förberäknade Ambient Occlusiontekniken. Nedan följer en beskrivning om detaljerna kring dessa implementationer.
    - **Förberäknad Ambient Occlusion.** De förberäknade teknikerna utvecklades i ett program där resultatet kunde visas direkt för enklare okulär felsökning. Implementationen skrevs i språket C# med biblioteket DirectX 9 för utritning. Ray Tracing algoritmen och strukturen skrevs från grund och likaså trianguleringen av polygoner som också används av Occlusion Querytekniken vilken använder DirectX 9 biblioteket för att utföra Occlusion Queries.
    - **Screen Space Ambient Occlusion.** De shaderbaserade teknikerna utvecklades efter sina vardera exempel presenterade dels i kod och text publicerat på Internet samt i kod medföljande spelet Crysis (Mittring, 07).

4.2 Metod för Delmål 2: Utvärdera tekniker

Utvecklingen av teknikerna förbättrar möjligheterna att utvärda dem på en experimentiell bas. Att tillämpa en experimentiell utvärdering är viktigt då det visuella utseendet kan skilja mycket emellan olikt utformade scener vilket kan vara svårt att bedöma med säkerhet utan att faktiskt presentera resultatet.

- **Experiment.** För att de olika implementationerna skall kunna betraktas som korrekta måste deras resultat evauleras. Då resultatet är visuellt och beror till stora delar på indatans utformning och parametrars inställning gentemot dessa lämpar sig deras evaulering bäst att utgöras av experiment. Genom att utsätta implementationerna för olika värden av parametrar och typer av indata kan olika resultat erhållas och värderas. Därför är experiment en rent nödvändig metod att tillämpa på problemet.
    - **Visuellt resultat.** En mycket viktig del av utvärderingen är det visuella resultatet som ges av de experiment teknikerna utsätts för. Utvärderingen av det visuella resultatet är givetvis långt ifrån vetenskaplig då den är ytterst subjektiv och empirisk men då teknikernas faktiska syfte är ett visuellt resultat måste detta ändå tas i beaktning. Resultatet ställs med fördel mot ett korrekt Global Illuminationresultat för att ge en något mer objektiv jämförelse.

- **Algoritmanalys.** För att de olika implementationerna skall kunna jämföras med varandra måste de utvärderas. Genom att analysera vissa egenskaper av implementationerna kan dessa ställas mot varandra i jämförelsen. Nedanstående kriterier undersöks för jämförelsen.
    - **Tidskomplexitet.** Genom logiskt resonemang ett uttryck bildas för förhållandet algoritmen bär mellan tidsåtgång för beräkning och mängd indata.
    - **Minneskomplexitet.** Genom resonemang kan ett förhållande mellan mängd indata och minneskrav urskönjas. Minnet kan krävas vid beräkning eller vid lagring av resultatdata. Utvecklingen av teknikerna ger också praktiska exempel på mängden data som måste lagras och dess minneskrav.
    - **Enkelhet.** Genom att implementera teknikerna ges underlag för att resonera kring enkelheten av tekniken. Utvecklingen ger insikt i såväl teknikens faktiska uppbyggnad såväl som dess svårighet att implementera.

5 Genomförande

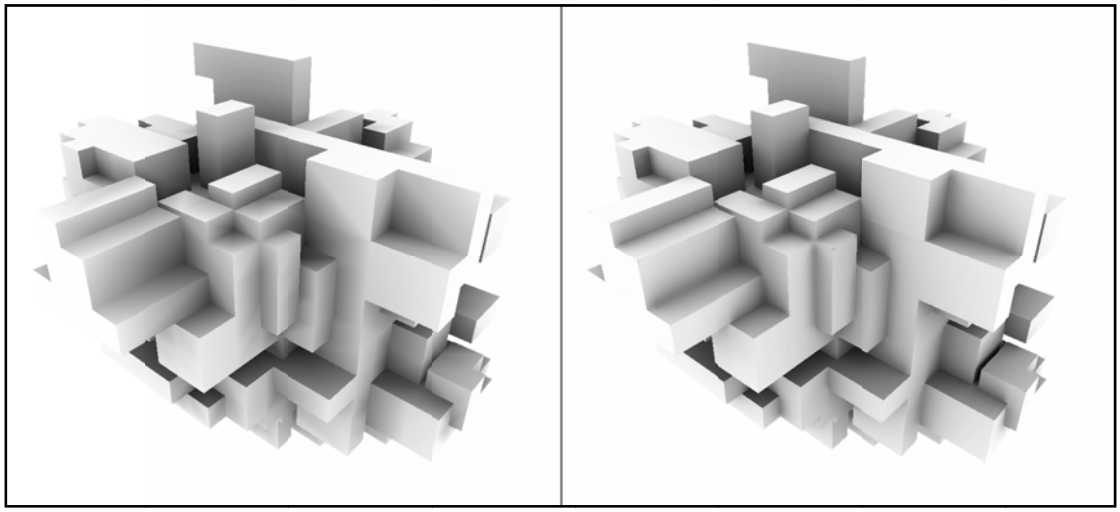
I följande kapitel presenteras genomförandet av de delmål som satts upp för problemet. Delmålens genomförande undersöks här efter varandra med störst fokus på utvecklingen vilken ligger till störst grund för resultatet.

5.1 Genomförande av Delmål 1: Utveckling

Nedan förklaras utvecklingen av samtliga tekniker. Beskrivningen går igenom teknikernas funktionalitet och utförande samt den generella tanken bakom tekniken.

5.1.1 Förberäknad Ambient Occlusion

Redan i (Zhukov, et al., 1998) presenterades en modell för att förberäkna Ambient Occlusiondata att applicera i en realtidsapplikation. Den teknik som presenteras i rapporten följer samma modell men gör en mer praktisk undersökning av tekniken där två olika tillvägagångssätt för beräkning av blockeringsdata undersöks. Den data som erhålls kan för mer komplexa modeller beräknas per vertex, dvs varje punkt i modellen erhåller ett separat blockeringsvärde att färgsätta modellen med. Alternativt kan datan beräknas för varje subyta på modellen och sparas i texturer vilket den lösning som presenteras i rapporten. Detta är praktiskt för mindre detaljerade modeller och ger möjlighet till att avväga mängden detaljrikedom mot andelen minne. Nackdelen med att spara datan i texturer är att mycket komplexa modeller då kräver ytterst högupplösta texturer för att samtliga blockeringsvärden skall kunna lagras. Figur 10 nedan visar ett exempel på en modell där datan i figuren till höger sparats i texturformat och beräknats på subytor och i figuren till vänster sparats per vertex. Vid närmare inspektion syns det hur per vertexlagringen ger upphov till interpoleringsfel över trianglarna på modellen, i mitten av vissa fyrkantiga stycken syns då ett förljusning. Likaså är den mörka randen i skarpa kanter mer jämn och följer längst med i modellen till höger där datan lagrats i textur.



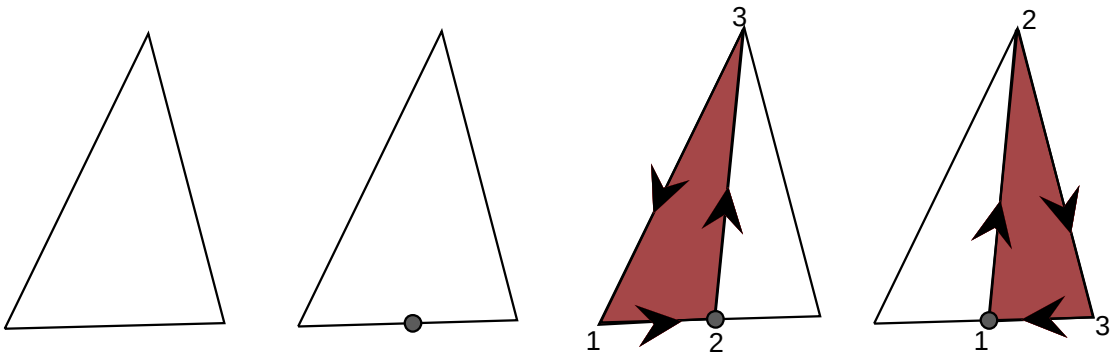
Figur 10. Jämförelse av pervertex och texturlagring av blockeringsvärde. Till vänster visas blockeringsvärde spara per vertex till höger sparat i textur.

Tesselering till subytor

För att kunna beräkna och spara ner högdetaljerade blockeringsvärden för modellen måste den delas upp i subytor. Varje yta på modellen delas då i hälften till dess subytorna endast täcker 1px i texturen där blockeringsvärdet kan lagras. Modellens

alla ytor äger s.k. texturkoordinater vilka innehåller den yta i texturen där värdet i realtid sedan skall hämtas ifrån. Dessa värden delas liksom koordinaterna för subytan och således kan de användas som koordinater också när datan skall lagras i texturen.

När ytorna skall tesseleras delas de på hälften. Detta sker genom att kanterna på ytan jämförs i längd och i mitten av den längsta kanten skapas en ny punkt. Denna punkt används för att knyta samman de två stycken nya halvorna som består vardera av mittenpunkten samt en av kantens punkter och den återstående punkten i ytan. Figur 11 nedan visar hur trianguleringen går till.



Figur 11. Exempel på hur halvrymdstriangulering går till

Koden nedan visar hur punkterna för ytan extraheras och sätts samman. Funktionen GetVertex(int) returnerar den valda punkten och i det fall indexet överstiger antalet punkter går det runt igen till 0, GetVertex(3) returnerar alltså för en yta med 3 punkter punkt 0.

```
static public Triangle[] HalfspaceTriangulate(Triangle triangle)
{
     float edgeLength = 0.0f;
     int longestEdge = 0;
     for(int i = 0; i < 3; ++i)
     {
           float length = ((Vector3)triangle.GetVertex(i).Position –
                  triangle.GetVertex(i + 1).Position).Length();
           if(length > edgeLength)
           {
                 edgeLength = length;
                 longestEdge = i;
           }
     }

     Vertex middlePoint = new Vertex();
     middlePoint.Position = new Vector3();
     middlePoint.TexCoord = new Vector2(0.0f, 0.0f);
     middlePoint.Normal = triangle.Normal;
     ColorValue newColor = new ColorValue
                (0.0f, 0.0f, 0.0f, 0.0f);
     for(int i = longestEdge; i <= longestEdge + 1; ++i)
     {
           Vertex vertex = triangle.GetVertex(i);
           ColorValue color =
                 ColorValue.FromArgb(vertex.Color);
           newColor.Red += color.Red * 0.5f;
           newColor.Blue += color.Blue * 0.5f;
           newColor.Green += color.Green * 0.5f;
           newColor.Alpha += color.Alpha * 0.5f;
           middlePoint.Position += vertex.Position * 0.5f;
           middlePoint.TexCoord += vertex.TexCoord * 0.5f;
```

```
        }

      Triangle[] triangles = new Triangle[2];
      triangles[0] = new Triangle
            (triangle.GetVertex(longestEdge), middlePoint,
            triangle.GetVertex(longestEdge + 2));
      triangles[1] = new Triangle(middlePoint,
            triangle.GetVertex(longestEdge + 1),
            triangle.GetVertex(longestEdge + 2));

      return triangles;
}
```

Ray Tracing

Som Ray Tracing används endast en enkel kollisionsundersökning mellan strålar och polgoner. För varje subyta undersöks ett godtyckligt antal strålar. Vissa källor (Pharr & Fernando, 2005) nämner att slumpvis valda strålar kan användas och antalet av dessa strålar som träffar närliggande geometri delas med antalet strålar för att erhålla blockeringsvärdet. Detta ger dock ett tämligen så instabilt resultat och varierar i mångt och mycket på vilka strålar som väljs ut (såvida inte väldigt många strålar används). Bättre är att uniformt placera strålar från normalen och avvikande därifrån. Framförallt ger detta bättre resultat då strålarnas avvikning från normalen kan användas som faktor på strålens faktiska influens på blockeringsvärdet (tidigare nämndes att geometri rätt framför ytan ger större inverkan på blockeringen). Koden nedan visar den metod som undersöker kollision mellan en polygon och en stråle (Akenine-Möller, et al., 2004). Först undersöks huruvida strålen överhuvudtaget går emot polygonens yta genom att undersöka punktprodukten. Därefter beräknas de barycentriska koordinaterna för se vart på polygonen strålen träffar, om koordinaterna ligger inom polygonens yta kan avståndet beräknas och om avståndet är positivt träffar strålen ytan och resultatet kan returneras.

```
static public bool IsIntersection(Ray ray, Triangle triangle,
      float &distance, float &u, float &v)
{
      u = 0.0f;
      v = 0.0f;
      distance = 0.0f;

      if (Vector3.Dot(triangle.Normal, ray.Direction) < 0.0f)
            return false;

      float epsilon = 0.000001f;

      Vector3 edge1 = triangle.Vertices[1].Position –
            triangle.Vertices[0].Position;
      Vector3 edge2 = triangle.Vertices[2].Position –
            triangle.Vertices[0].Position;

      Vector3 p = Vector3.Cross(ray.Direction, edge2);
      float a = Vector3.Dot(edge1, p);
      if (a > -epsilon && a < epsilon)
            return false;

      float f = 1.0f / a;
      Vector3 s = ray.Origin - triangle.Vertices[0].Position;
      u = f * Vector3.Dot(s, p);
      if (u < 0.0f || u > 1.0f)
            return false;
```

```
      Vector3 q = Vector3.Cross(s, edge1);
      v = f * Vector3.Dot(ray.Direction, q);
      if (v < 0.0f || u + v > 1.0f)
            return false;

      distance = f * Vector3.Dot(edge2, q);
      if (distance < 0.0f)
            return false;

      return true;
}
```

Occlusion Query

Istället för att använda Ray Tracing kan grafikkortets processorkapacitet utnyttjas genom Occlusion Queries (Pharr, et al., 2005). Hela scenen inom ett visst avstånd renderas och sedan efterfrågas hur många fragment/pixlar som syns. Genom att dela detta värde med antalet möjliga fragment/pixlar erhålls en faktor på hur stor del del av subytans vy som täcks av närliggande geometri. Koden nedan visar ett exempel på Occlusion Queries m.h.a. DirectX 9.

```
Matrix oldView = mDevice.Transform.View;
Matrix oldWorld = mDevice.Transform.World;
Matrix oldProjection = mDevice.Transform.Projection;
mDevice.Transform.Projection =
Matrix.PerspectiveFovLH((float)Math.PI / 2.0f, 1.0f, float.Epsilon,
mMaxDistance);
mDevice.Transform.World = Matrix.Identity;

// Render
Surface oldSurface = mDevice.GetRenderTarget(0);
mDevice.SetRenderTarget(0, mQueryTexture.GetSurfaceLevel(0));
mDevice.BeginScene();
{
    //mDevice.Transform.View = Matrix.LookAtLH(point + direction *
0.025f, point + direction * 5.0f, new Vector3(0.0f, 0.0f, 1.0f)); /*
Verkar inte ge någon större skillnad */
    mDevice.Transform.View = Matrix.LookAtLH(point + direction *
           0.025f, point + direction * 5.0f,
            Vector3.Cross(point, direction));
    mDevice.Clear(ClearFlags.Target | ClearFlags.ZBuffer,
            Color.Black.ToArgb(), 1.0f, 0);

    mQuery.Issue(IssueFlags.Begin);
    foreach (Model m in mModels)
        m.Render();
    mQuery.Issue(IssueFlags.End);

    //int pixels = mTextureQuality;
    int pixels = 700 * 600;
    UInt32 visible = 0;
    bool dataReturned = false;
    while (dataReturned != true)
        visible = (UInt32)mQuery.GetData(typeof(UInt32), true,
            out dataReturned);

    occlusion = (float)visible / (float)pixels;
}
mDevice.EndScene();
mDevice.SetRenderTarget(0, oldSurface);
```

5.1.2 **Screen Space Ambient Occlusion**

Samtliga Screen Space Ambient Occlusiontekniker använder sig av Vertex och Fragment Shaders. Anledningen till att de kallas Screen Spacetekniker är att de agerar på pixlar snarare än geometri och närväl scenens data renderats till skärmtexturer används dessa för att göra Occlusionberäkningarna.

Crysis Shader

Till spelet Crysis (Mittring, 2007) så användes en uppsättning shaders för att återskapa mörkare partier vid närliggande geometri genom att undersöka djupbuffern. För varje pixel på skärmen samplas omkringliggande texlar i djupbuffern och skillnaden däremellan används för att beräkna huruvida punkten torde vara förmörkad eller ej. Något speciellt med den variant av djupskillnadssampling som används i Crysis shader är deras slumpvis roterade kärna för samplingar. En välbalanserad brustextur används för att spegla ett antal samplingspunkter över plan. Det plan som används för att spegla avgörs av pixelns texturkoordinater vilka används för att sampla brustexturen ytterst högfrekvent. Resultatet blir att samplingspunkterna är harmoniskt psuedoslumpade. Figur 12 nedan visar ett exempel på hur Occlusiondatan för en viss skärm i spelet Crysis kan se ut. Koden för shadern är tillgänglig genom datan som följer med spelet.

Figur 12. Ambient Occlusionexempel ur spelet Crysis (Mittring, 2007). Bild tillgänglig på http://en.wikipedia.org/wiki/Image:SSAO.jpg [hämtad 2008-05-11]

Depthtexture Unsharpening

Depthtexture Unsharpening tekniken är den enklaste och kanske snabbaste av alternativen (Pan, 08). Tekniken baseras på att djupvärden för skärmen lagras och ges oskärpa. Den efteråt suddiga bilden jämförs med orginaldjupvärden och av resultatet ges en skugga. Djupvärdena lagras i en 16bitars textur (för hög upplösning och färre flyttalsfel). Djupvärdet beräknas genom att ta vyprojektionens transformerade punkts z-värde och skala om detta till intervallet [0, 1] och multiplicera detta med en faktor för att kunna skala effekten beroende på scenens storlek. Nedan visas koden ur vertex

shadern för att spara ner djupvärdet. Detta interpoleras sedan över vertexarna och sparas ned i texturbuffern utan vidare beräkning i pixel shadern.

```
Output.Position = mul( Input.Position, matViewProjection );
Output.Depth = ( Output.Position.z / farClip ) * depthScale;
```

Den lagrade texturbuffern ges sedan oskärpa genom ett tvåpass filter där pixelinformationen i första passet suddas i x led och sedan i yled. Koden nedan visar hur varje pixel ges oskärpa genom att blanda pixelinformationen i den aktiva pixeln med omkringliggande pixlars data. Skillnaden mellan oskärpningskoden i xled och i yled är endast att texturkoordinaterna Texcoord adderas med float(0.0f, offset * i) istället för float(offset * i, 0.0f). När texturen samplas undersöks också om punkten tillhör bakgrunden i vilket fall pixeln bortses från så att bakgrunden inte skuggas av objektet.

```
float4 ps_main(float2 Texcoord : TEXCOORD0) : COLOR0
{
   float offset = (1.0f / 512.0f) * samplingDistance;

   float sum = 0.0f;
   float samples = 0.0f;
   for(int i = -3; i < 4; ++i)
   {
      float sample =
           tex2D(screen, Texcoord + float2(offset * i, 0.0f));
      if(sample > 0.0f)
      {
         sum += sample;
         samples += 1.0f;
      }
   }

   sum *= 1.0f / samples;


   return sum;
}
```

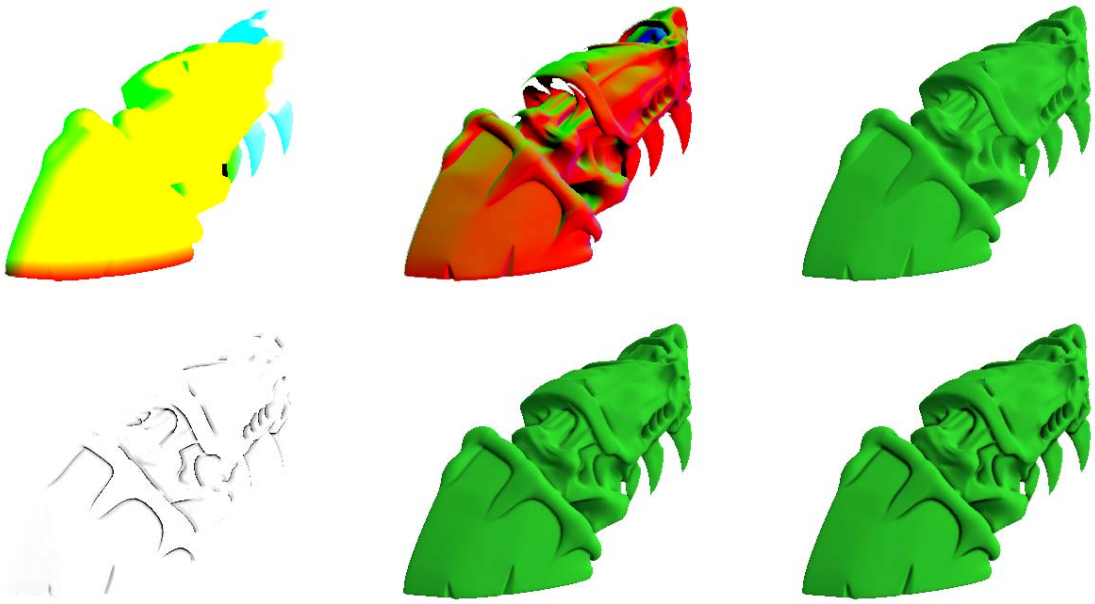
Efter att oskärpan är tillämpad renderas scenen normalt och i ett pass därefter läggs skillnaden i djupbuffrarna till resultatscenen med följande pixelshader. Variabeln blurFactor används för att skala djuptexturen med oskärpa för att öka skillnaderna i djupvärden. Variabeln factor används för att lokalt öka influensen av skillnaden i slutresultatet.

```
float4 ps_main(float2 Texcoord : TEXCOORD0) : COLOR0
{
   float2 texcoord = Texcoord - float2(0.5f, 0.5f);
   texcoord *= blurFactor;
   texcoord += float2(0.5f, 0.5f);

   float diff = tex2D(depthOriginal, Texcoord) –
      tex2D(depth, texcoord);
   float reduction = 1.0f - diff * factor;

   return reduction;
}
```

Crease Shader (Fox, 2007)

Genom att förutom djupbuffern också använda en textur med lagrade normaler kan effekten av Screen Space Ambient Occlusionshaders förbättras markant då ytors

faktiska vinkel mot varandra kan användas för att begränsa blockeringen av plana ytor. Denna teknik kan ses som en vidarearbetning på s.k. Crease Shaders som används för att finna skarpa kanter mellan geometri genom att sampla omkringliggande pixlar och undersöka punktprodukten mellan normalerna. Punktprodukten i detta fall (då normalerna är normaliserade) ger ett värde i intervallet [-1, 1] där värdet är 0 i det fall normalerna är vinkelräta. Denna variant av shader kräver dock betydligt fler renderingspass än den som presenteras från Crysis (se ovan). Koden nedan använder dessutom istället för en djupbuffer en textur lagrandes geometrins position vilket gör avståndsberäkningarna enklare. Utöver pass för att rendera normalerna och positionerna måste även scenen renderas samt blockeringsvärdena beräknas. Blockeringsvärderna blir dessutom ytterst lokala så för att förbättra resultats spridning görs med fördel två stycken pass med dels vertikal dels horisontell s.k. Gaussian Blur. Figur 13. Resultat av Crease Shader renderingspass. Övre vänstra bilden visar första renderingspasset och nedre vänstra bilden visar 4e renderingspasset där Ambient Occlusionvärden beräknas. Figur 13 nedan visar resultatet från varje pass där bilden överst till vänster är första passet då positionerna lagras. Anledningen till att resultatet av passet ser enfärgat ut är att positionerna lagras i världskoordinater och därför överskrider det färgintervall [0, 1] som kan skrivas ut på skärmen, värdena lagras dock normalt i texturen.



Figur 13. Resultat av Crease Shader renderingspass. Övre vänstra bilden visar första renderingspasset och nedre vänstra bilden visar 4e renderingspasset där Ambient Occlusionvärden beräknas.

Nedan följer koden för pixelshadern i det renderingspass som genererar occlusiondatan. SampleRandom som nämns när närliggande texlar skall samplas är en Array med 32st samplingsvärden.

```
sampler Position;
sampler Normal;

float SampleRange;
float2 TextureSize;
float MinimumCrease;
float4 Params;
float4 ps_main(float2 Texcoord : TEXCOORD0) : COLOR0
{
```

```
    const float2 Right = float2(1.0f / TextureSize.x, 0.0f);
    const float2 Up = float2(0.0f, 1.0f / TextureSize.y);
    const float2 Center = Texcoord + (Up + Right) * 0.5f;

    float3 centerNormal   = tex2D(Normal, Center).xyz;
    float3 centerPosition = tex2D(Position, Center).xyz;
    float sampleRange = SampleRange;

    float4 sum = 0.0f;
    for(int i = 0; i < 24; ++i)
    {
       float2 sampleUV = Center + SampleRandom[i].xy *
            (Right + Up) * sampleRange;
       float3 samplePos = tex2D(Position, sampleUV).xyz;

       float3 toCenter = samplePos - centerPosition;
       float dist = length(toCenter);
       toCenter = normalize(toCenter);

       float centerContrib = saturate((dot(toCenter, centerNormal) –
            MinimumCrease) * Params.y);
       float rangeAttenuation = 1.0f - saturate(dist / Params.x);
       sum += centerContrib * rangeAttenuation;
       sampleRange *= 1.1f;
    }
    sum *= 1.0f / Params.z;

    return sum;
 }
```

5.2 Genomförande Delmål 2: Utvärdering

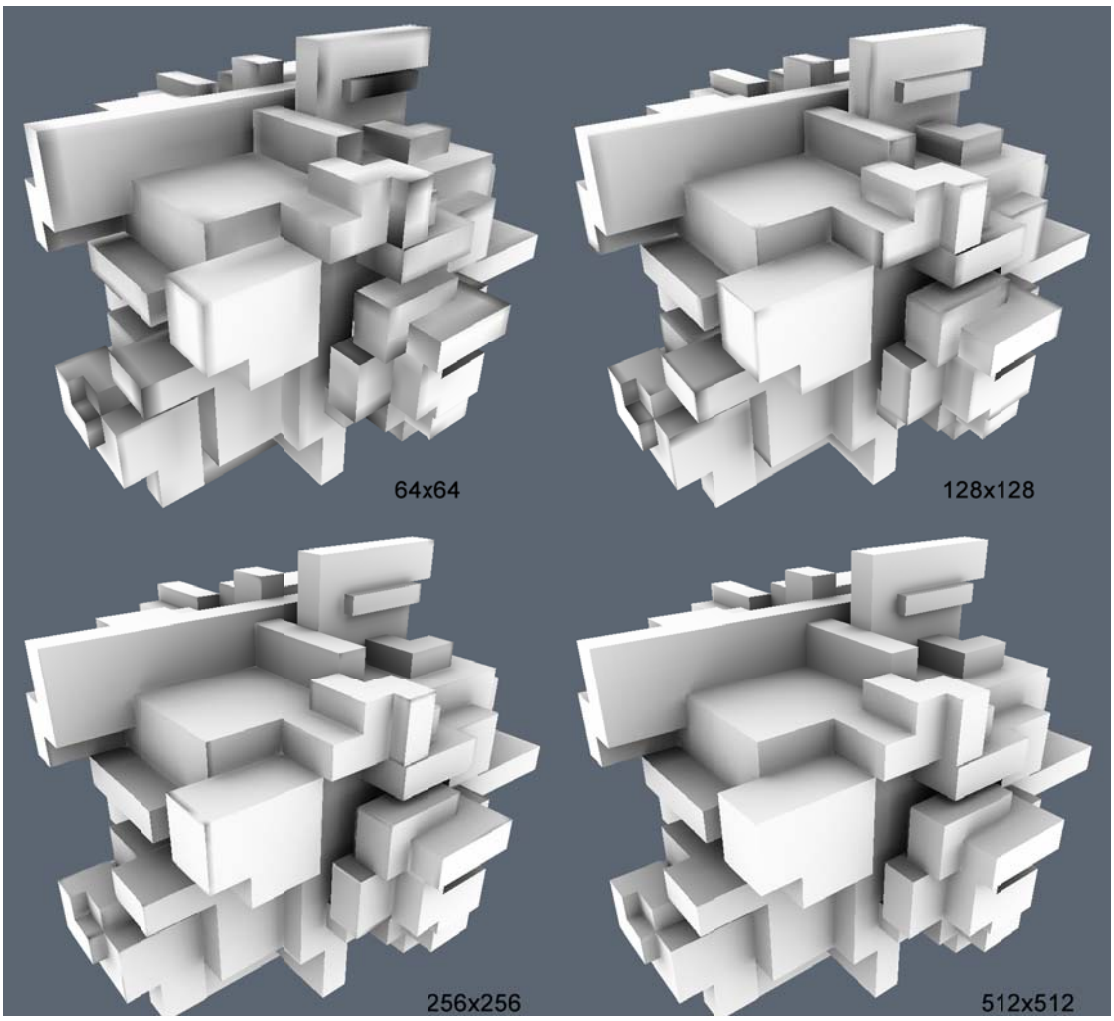
Nedan beskrivs de observationer av teknikerna som gjorts utifrån de kriterier som presenterats för problemet.

5.2.1 Förberäknad Ambient Occlusion

Den teknik av Ambient Occlusion som betraktats i rapporten lagras i texturer och skall för båda beräkningstekniker (Ray Tracing och Occlusion Queries) ge samma visuella resultat. Detta betyder att endast enkelheten av de skiljda metodernas implementation och dess tidskomplexitet skiljer dem emellan. Dessa egenskaper diskuteras således för de två tillvägagångssätten separat.

- **Enkelhet.** Gemensamt för båda modeller av beräkning är att en implementation måste stödja inläsning av modeller och dessutom hantering av modellernas struktur. Därefter måste modellerna kunna tesseleras till subytor vars blockeringsresultat skall mappas till pixlar på texturen. Också teknikernas beräkningskod är en implementationssvårighet vilket presenteras nedan.
    - **Ray Tracing.** Att implementera kod för Ray Tracing innebär mycket ibland svårtolkad 3D matematik. Ofta är algoritmerna som krävs geometriskt svårtolkade såväl som kodmässigt svårtolkade på grund av deras många optimeringar.
    - **Occlusion Queries.** Visst arbete ligger i att förstå sig på och kunna arbeta mot ett 3D API (som exempelvis DirectX 9) vilket krävs när Occlusion Queries används. Vidare måste modellerna som lästs in kunna renderas som ett steg i blockeringsberäkningen. Själva proceduren är dock intuitiv och lättolkad.

- **Visuellt resultat.** För att kunna utvärdera det visuella resultatet jämförs resultatet av en rendering mot en riktig rendering användande Global Illumination. Det visuella resultatet jämförs subjektivt utifrån kriteriet att skuggningen av modellen bör överrensstämma mot ett resultat beräknat med Global Illumination.
- **Tidskomplexitet.** Förberäknad Ambient Occlusion äger den fördelen att närväl blockeringsdatan är beräknad behöver den endast appliceras och med dagens grafikkort är detta ett mycket acceptabelt tidskrav (Fernando, 2004) (objekt textureras ändå i de flesta fall). Problemet är att samtliga objekt kräver egna texturer och således måste en stor mängd texturdata överföras till grafikkortets minne. Dessutom så kommer den aktuella datan på grafikkortet att behöva bytas ofta då en komplett scens blockeringstexturer inte kommer kunna samlas i minnet samtidigt utan istället måste bytas ut mot texturer för de för tillfället synliga objekten. Förberäkningstiden för en komplex scen kan dessutom bli tämligen lång. Detta då båda tekniker undersökta är beroende av mängden geometri.
    - **Ray Tracing.** Då Ray Tracing metoden för att beräkna Ambient Occlusion sänder strålar från samtliga subytor vilka måste testas mot samtlig geometri vilket kräver en stor mängd arbete. Antalet subytor beror dessutom på storleken på texturen (vilken bestämmer hur små subytorna behöver vara) vilket är direkt kopplat till den slutgiltiga kvalitén på resultatet.
    - **Occlusion Queries.** Liksom metoden med Ray Tracing kräver Occlusion Queries att hela scenen skall renderas för samtliga subytor. En rendering av scenen kan dock med hjälp av grafikkort ofta genomföras mycket snabbare än ett test för en stråle mot hela scenen, dessutom måste scenen bara renderas en gång per subyta emedan flertalet strålar måste skickas från subytan med Ray Tracingmetoden.
- **Minneskomplexitet.** Då förberäknad Ambient Occlusion måste lagras krävs en viss mängd texturminne för varje modell som tekniken skall appliceras på i scenen. Modeller som placeras ut på flera platser i scenen måste dessutom ha separata texturer beroende på vart de är placerade då deras omgivning kan ge annan blockeringsdata. Mycket komplexa modeller kräver också stora texturer för att kunna lagra all data på ett bra sätt men även enklare objekt kräver ofta texturstorlekar från 256x256 pixlar och uppåt då mindre texturer kan göra att blockeringsdatan inte kan lagras korrekt. Figur 14 nedan visar hur en modell ser ut texturerad med olika texturstorlekar. Ett resonemang för förenkling av textureringen är att ambient occlusiondatan bakas in i texturerna, dvs ljuset på texturen sänks direkt och den modifierade texturen används för att texturera objektet i realtid. Detta är dock först och främst för många objekt mer minneskrävande då texturerna oftast har fyra färgkanaler emedan Ambient Occlusiontexturen endast kräver en kanal. Två hundra objekt med samma textur kan således använda samma textur med fyra kanaler och sedan en varsin Ambient Occlusiontextur. Om datan skulle bakas skulle varje objekt tvunget ha en fyra kanalstextur och då krävs nästan 4ggr mer minne. Dessutom kan Ambient Occlusiontexturen ofta vara i lägre upplösning än den faktiska detaljtexturen vilket kan spara ytterligare minne. Den viktigaste anledningen till varför bakning inte fungerar är dock att texturen endast skall användas för att sänka just ambient ljus, om datan bakas i texturen så appliceras fortfarande ljussättning ovanpå vilket till stora delar förstör meningen med Ambient Occlusion.



Figur 14. Exempel på Ambient Occlusion för olika texturstorlekar

5.2.2 Screen Space Ambient Occlusion

Då utvärderingen av samtliga SSAOtekniker angripits på liknande vis presenteras här bara en kort genomgång på motiveringarna bakom teknikernas utvärderingar. Resultatet av utvärderingen för de enskilda teknikerna återfinns i följande kapitel.

- **Tidskomplexitet.** Samtliga SSAOtekniker äger egenskapen att de i stora drag är oberoende av mängden geometri som renderas. Då teknikerna arbetar i Screen Space på de pixlar som renderats så är antalet berörda element ständigt konstant och beroende endast på skärmupplösningen. Teknikerna kräver givetvis också ytterligare data än endast den renderade scenen och blir på så sätt också till viss del beroende av geometrin då denna kan behöva renderas flera gånger. Detta kan i vissa fall dock ändå vara ett redan nödvändigt steg för andra Screen Spacetekniker som appliceras i många av dagens spel vilket gör att den data som renderas på så sätt redan finns tillgänglig för SSAOteknikerna att använda.
- **Minneskomplexitet.** Det enda minneskravet som SSAOteknikerna innehar är det minne som krävs för lagringen av exempelvis djupvärden eller normaler i texturer på grafikkortet. Dessa texturer kan i vissa fall kräva mycket minne och framförallt stödjer en del av dagens grafikkort inte vissa texturformat vilket kan gör teknikerna svåra att tillämpa.

- **Enkelhet.** Teknikerna skiljer i komplexitet men gemensamt är ändå att de alla kräver ytterst lite kod jämfört med andra tekniker för Ambient Occlusion. Att teknikerna komprimeras i lite kod betyder också att de är lättare att förstå än ett stort och komplext system för beräkning och vissa av teknikerna är dessutom intuitiva och således lättare att implementera.
- **Visuellt resultat.** Det visuella resultatet jämförs för teknikerna mot deras rendering utan shadern aktiv vilket ger en god bild av teknikens förbättring av modellens intryck.

6 Resultat

6.1 Resultat för Delmål 1: Utveckling

6.1.1 Förberäknad Ambient Occlusion

Applikationen som förberäknar Ambient Occlusiontexturer gör detta genom att först ladda in modellerna i scenen. Varje yta i modellen delas sedan itu till subytor tills de resulterande subytorna fyller mindre än en pixel i resultattexturen. Om Ray Tracing används sänds då strålar från subytan som testas mot återstående ytor i scenen, de som träffar något räknas in som blockerande ytor och ett resultat beräknas. Om istället Occlusion Queries används så centreras en kamera på subytan riktad i ytans normalriktning. Därefter renderas scenen och antalet pixlar av scenen som syns divideras med antalet möjliga pixlar vilket ger ett blockeringsvärde. Blockeringsvärdet placeras sedan in i den pixel på resultattexturen som subytan fyller.

6.1.2 Crysis SSAO

Tekniken renderar först scenens avstånd till kameran vilket lagras i en textur. För varje pixel i texturen samplas sedan omkringliggande pixlar och skillnaden i djupvärden används för att beräkna huruvida de omkringliggande pixlarna kan skymma den aktuella pixeln. Resultatet renderas sedan ovanpå den faktiska scenen.

6.1.3 Crease SSAO

Tekniken kräver att normaler och positioner för scenen renderas innan själva beräkningen kan ske. Sedan beräknas resultatet genom att för varje aktuell pixel sampla vilken position i världen som pixeln har. Kring den positionen plockas ett antal punkter ut vilka projiceras tillbaka till skärmen och samplas i positionstexturen, skillnaden i djup används sedan pixlarna emellan (om de inte är på för långt avstånd) för att se om den samplade punkten kan skymma den aktuella pixeln. Resultatet begränsas sedan med punktprodukten mellan de två punkternas normaler för att undvika att punkter på plana ytor blockerar intilligande punkter. Resultatet ges oskärpa för ett mindre stelt utseende innan det renderas ovanpå den slutgiltiga scenen.

6.1.4 Depthtexture Unsharpening SSAO

Genom att rendera scenens djupvärden och ge resultattexturen oskärpa och sedan subtrahera resultatet från orginaldjupvärdena ges en skuggning av bakomliggande punkter i scenen.

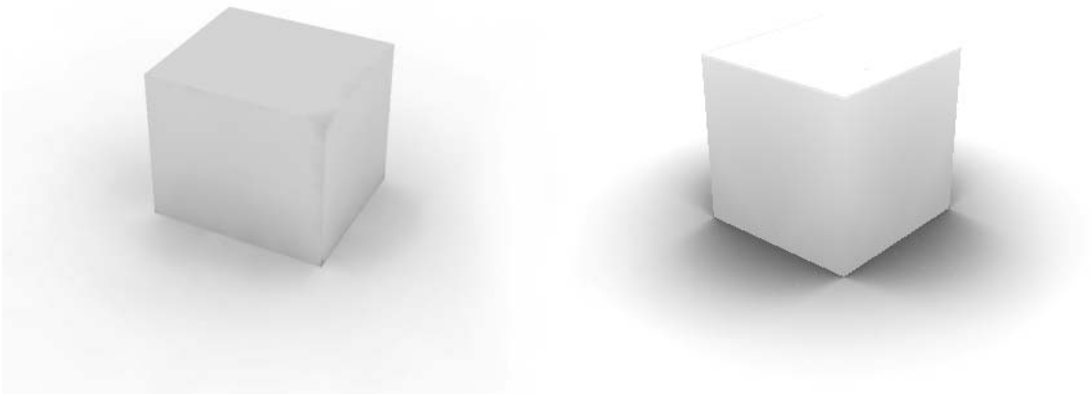
6.2 Resultat för Delmål 2: Utvärdering

6.2.1 Förberäknad Ambient Occlusion

Mellan de två metoderna (Occlusion Queries och Ray Tracing) för förberäknad Ambient Occlusion skiljer endast metoden för hur blockeringsdatan beräknas, denna utvärderas således i slutet av stycket för båda tekniker. En mycket viktig aspekt och nackdel av de förberäknade teknikerna är att de när datan lagras i texturer blir permanenta och i en scen med rörliga objekt blir då occlusiondatan inkorrekt. Detta är en begränsning av teknikerna som spelar stor roll i deras användbarhet. Teknikerna bär annars följande egenskaper:

- **Minneskomplexitet.** Då samtliga modeller i scenen som skall textureras med Ambient Occlusion måste äga en specifik textur kan detta snabbt eskalera till hundratals texturer. Dessutom bör texturerna vara kring 256x256 pixlar stora och ofta krävs än större texturer. En sådan textur med 8bitar data per pixel kräver 64kb texturminne på grafikkortet. För ett par hundra sådana objekt krävs rätt snart upp emot 20-30mb minne vilket för dagens grafikkort är en klart märkbar kostnad. Dessutom krävs i de flesta fall än större texturer för större objekt samt att det i interaktiva medier som spel oftast finns över tusentals objekt i scenen vilket gör att minneskraven blir en viktig nackdel att ta i betraktning för teknikens tillämpning.
- **Tidskomplexitet.** När väl texturer för Ambient Occlusion beräknats är dess tidskrav i realtidstillämpning med dagens hårdvara knappt nämnvärd. Själva beräkningen kan dock vara långt mer kostsam. Nedan beskrivs skillnaden i tidsåtgång för beräkning för de två olika metoderna.
    - **Ray Tracing.** Då varje yta delas in i ett antal subytor vilka testas mot samtliga andra ytor blir komplexiteten O(N^2) och dessutom beroende på antalet subytor som ytorna delas in i. Detta gör att beräkningen är kraftigt beroende på mängden geometri i scenen. För att dessutom få högre kvalité på resultatet måste ett antal strålar sändas från varje subyta vilket direkt multiplicerar beräkningstiden med antalet strålar som används. Detta gör Ray Tracingmetoden mycket starkt beroende på mängden geometri som beräknas. Antalet strålar som testas kan uppstiga emot 512st (Fernando, 2004).
    - **Occlusion Queries.** Att beräkna blockeringsvärdet med Occlusion Queries måste liksom med Ray Tracing (se ovan) göras för varje subyta. Skillnaden är dock att någon direkt beräkning mot varje annan yta i scenen inte behöver ske utan då scenen istället renderas så blir komplexiteten snarare O(N) där N är antalet subytor. Renderingen som sker på grafikkortet kan nämligen göras långt mer optimerat och styr således inte tidsåtgången lika hårt som i Ray Tracingmetoden. Framförallt så behöver scenen endast renderas en gång jämfört med Ray Tracing där ett mycket högt antal strålar måste testas för ett fullgott resultat.
- **Enkelhet.** Implementationen av ett program för att skapa Ambient Occlusiontexturer kräver många stora delkomponenter som inläsning och hantering av modeller samt lagring av texturer med mera. Vidare måste stöd sedan finnas för att modellerna i realtidsapplikationen skall kunna renderas med texturerna på rätt sätt. Nedan beskrivs också implementationssvårigheten för de två beräkningsteknikerna.
    - **Ray Tracing.** Att skriva kod för Ray Tracing är komplicerat och algoritmerna är ofta svårtolkade och kräver kunskap i 3D matematik. Vidare så är implementationen ytterst svår att felsöka. Fördelen med att skriva Ray Tracing algoritmer är att många optimeringar kan göras för att eliminera mängden element som måste behandlas.
    - **Occlusion Queries.** För att kunna använda sig av Occlusion Queries måste en viss grad av implementation göras emot något 3D Api som exempelvis DirectX eller OpenGL. Själva tekniken är dock lättförklarad och intuitiv och kan felsökas visuellt genom att scenen renderas.
- **Visuellt Resultat.** Figur 15 nedan visar en jämförelse mellan en scen renderad med Global Illumination och en scen renderad med Occlusion Queryberäknad

Ambient Occlusion. Resultatet av Ambient Occlusionscenen är klart överdrivet men i övrigt kan tydliga likheter scenerna emellan urskönjas. En fyrkantig skugga möter på båda bilder kubens nederkant. Vidare så appliceras Ambient Occlusiontexturen i realtid på objektet emedan scenen renderad med Global Illumination krävde närapå en minuts renderingstid. Nämnas bör att det visuella resultatet för Ray Tracing med tillräckligt antal strålar och lika stor textur ger samma slutresultat som Occlusion Queries.



Figur 15. Resultat av Occlusion Queryberäkning för Ambient Occlusion jämfört med en rendering med Global Illumination. Till vänster visas en scen renderad med Global Illumination och till höger en scen renderad med Ambient Occlusion. För större bild se Appendix 4.

6.2.2 Screen Space Ambient Occlusion

Nedan följer resultat av en generell utvärdering av de aspekter SSAOteknikerna har gemensamt. Därefter följer utvärderingar för de enskilda teknikerna.

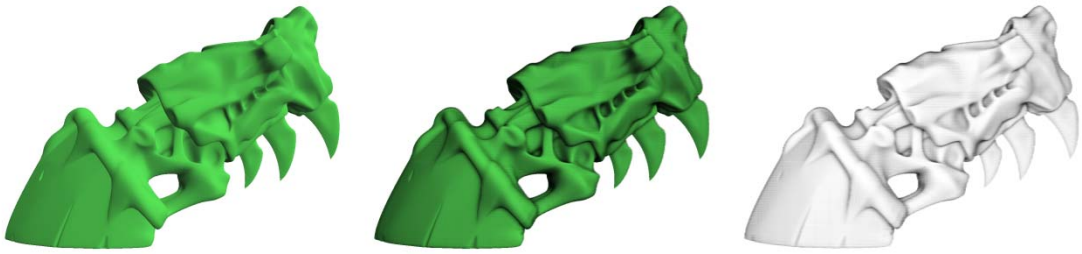
- **Minneskomplexitet.** Då Screen Spaceteknikerna endast agerar på pixlar så krävs endast lagring av texturer för skärmdata. Skärmdatan sparas och manipuleras dessutom rakt på grafikkortet och behöver således inte heller sändas mellan huvudminnet (ram minnet) och grafikkortets minne. Vidare så är mängden minne direkt beroende på skärmstorleken då antalet pixlar i skärmen bestämmer minneskravet.
- **Tidskomplexitet.** Tidskomplexiteten för Screen Space effekter är precis som minneskomplexiteten konstant beroende på skärmstorleken, O(1). Detta gäller för antalet pixlar och i de fall geometri måste renderas i flera pass blir tekniken mer beroende av mängden geometri.
- **Enkelhet.** En Screen Spaceshader kan i det enklaste utförandet kräva ytterst lite kod och kan dessutom vara ytterst intuitivt utförd. Svårare med användandet av shaders i allmänhet är dock att applikationen måste inneha en speciell struktur och design för renderingen vilken tillåter att shaders används. Detta kan i många fall lägga mycket arbete på skapandet av applikationen men då många applikationer även använder shaders till annat brukar detta vara ett redan avklarat och motiverat problem.
- **Visuellt resultat.** Till skillnad från de förberäknade metoderna för Ambient Occlusion kan inte helt korrekt Ambient Occlusiondata (Zhukov, et al., 1998) erhållas med dagslägets shadermetoder. Detta gör att det visuella resultatet varierar kraftigt mellan de olika metoderna. Vidare så är det visuella resultatet som en approximation i Screen Space oftast starkt beroende på geometrins utformning. Vissa av teknikerna presenterade nedan har exempelvis problem

med plana ytor i låga vinklar emedan andra kan uppvisa artefakter vid höga avstånd.

Crysis Ambient Occlusion

Den implementation av Crysis Shader som gjorts kräver pixelshader 3.0 samt 5st renderingspass av vilka 2 pass renderar geometrin.

- **Tidskomplexitet.** Crysis Ambient Occlusionshader är en mycket arbetskrävande sådan och använder trots endast 8-16 samplingar Shadermodel 3.0 p.g.a. den stora mängden övriga instruktioner som måste appliceras på samtliga samplingar. Resultatet är dessutom lite kornigt och behöver därför suddas ut något vilket kräver 2st Gaussian Blur pass.
- **Enkelhet.** I grundutförandet (som Crytek, utvecklarna av Crysis, skrivit shadern) är den ytterst svårförstådd. Även med omskrivning tar shadern hänsyn till många olika detaljer som exempelvis skalning av effekt beroende på avstånd från kameran, sådana specifika detaljer gör shadern ytterst svårtolkad. Den agerar dock på ett logiskt sätt och kan därför ses som enkel att implementera. Den största svårigheten ligger i att anpassa de olika samplingsparametrarna till att stämma överrens med skalan på scenens geometri.
- **Visuellt Resultat.** Resultatet är bland det bästa av presenterade shaders i pappret och kan klart jämföras med ett korrekt Ambient Occlusionresultat. Det är dock likt samtliga SSAOtekniker tämligen lokalt vilket är oundvikligt. Tekniken är pga det relativt låga samplingsvärdet (8/16) något kornigt vilket till viss del kan vara svårt att sudda bort med Gaussian Blur men det är inte omöjligt. Figur 16 nedan visar resultatet av den Crysis Shader som implementerats. Vänster i bilden visas modellen med endast diffusbelysning och i mitten syns detsamma fast med Ambient Occlusion, till höger visas den Occlusiondata som genererats.



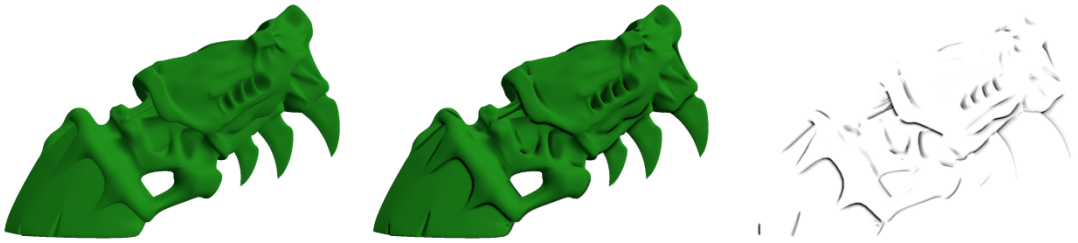
Figur 16. Crysis Resultat, till vänster visas modellen med diffusbelysning. I mitten visas modellen med diffusbelysning och Ambient Occlusion och till höger visas Ambient Occlusiondatan. För större bild se Appendix 2.

Crease Shader

- **Minneskomplexitet.** Då både normaler och positioner sparas krävs två stycken ytterst stora texturer. I implementationen som genomförts användes 16bitar per färgkanal för att ge lite extra precision men 32bitar skulle nästan behövas. Texturerna kräver med 16bitar data per kanal nära på sex megabyte styck vilket kan vara klart påfrestande för grafikkort som tillverkats för några år sedan.
- **Tidskomplexitet.** Crease shadern är något mer krävande än övriga varianter då den kräver att scenen renderas tre gånger. En gång för positioner/djup, en gång för normaler och en gång för själva scenen. Givetvis är detta gratis om deferred shading ändå används men för enbart Ambient Occlusionteknikens

skull blir det en rejäl overhead. Shadern arbetar dessutom på ett extra pass då normalerna också måste sparas utöver positionerna.

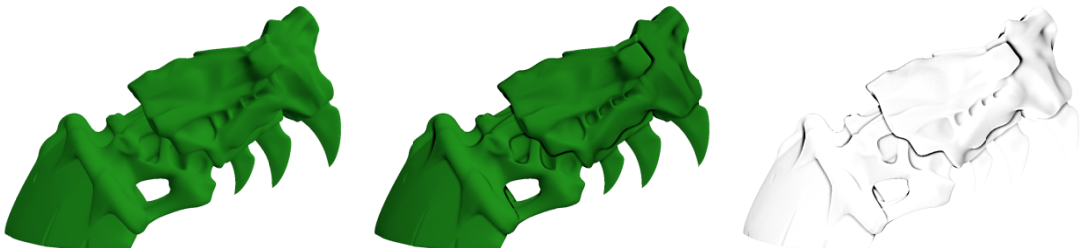
- **Enkelhet.** Creaseshadern är ytterst intuitiv och således mycket enkel, den utgör ett gott och smidigt alternativ för utvecklare med mindre krav på prestanda.
- **Visuellt resultat.** Det visuella resultatet är på vissa sätt mer lokalt än de andra shaderlösningarna då endast närliggande hörn skuggas. Detta ger fördelen att plana ytor inte lider av skuggning men ger också nackdelen att skrymslen och vrår med lite avstånd inte skuggas. För mer rät och strikt geometri kan tekniken ge bättre resultat än andra alternativ men ger annars ett något undermåligt resultat när det gäller mjukhet och naturlighet. Figur 17. Crease Shader resultat, till vänster visas modellen med diffusbelysning. I mitten visas modellen med diffusbelysning och Ambient Occlusion och till höger visas Ambient Occlusiondatan. nedan visar ett resultat av shadern.



Figur 17. Crease Shader resultat, till vänster visas modellen med diffusbelysning. I mitten visas modellen med diffusbelysning och Ambient Occlusion och till höger visas Ambient Occlusiondatan. För större bild se Appendix 1.

Depthtexture Unsharpening

- **Minneskomplexitet.** Trots att tekniken endast arbetar på oskärpning av djuptexturen krävern den tre stycken högprecisionstexturer, en för att lagra orginaldjupet och två stycken för att kunna göra vertikalt och horisontellt oskärpningspass.
- **Tidskomplexitet.** Då tekniken endast baseras på oskärpa av djuptexturen kan både antalet samplingar hållas lågt och antalet pass kan reduceras. Desto fler samplingar som görs desto bättre blir givetvis resultatet men i båda fallen krävs inga ytterligare beräkningar vilket gör tekniken till en ytterst snabb sådan.
- **Enkelhet.** Tekniken är klart den enklaste tekniken av de som presenterats men har däremot nackdelen att den endast kan användas till vissa typer av scener.
- **Visuellt resultat.** Skuggningen är inte alls lik korrekt Ambient Occlusion då skuggningen sker på samtliga objekt bakom och liknar en slags svart glöd omkring kanterna på objektet men kan vid vissa fall ge god effekt ändå. Tekniken fungerar i synnerhet bra för att framhäva små detaljer på modeller. Figur 18 nedan visar en resultatbild för tekniken.

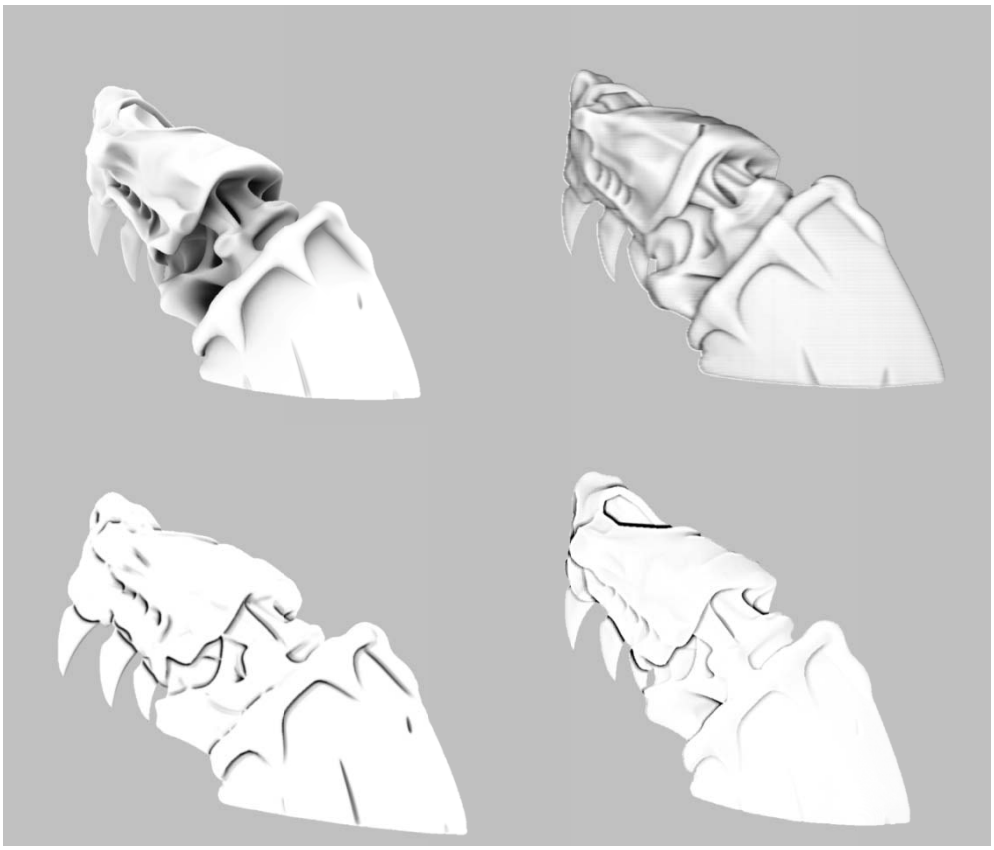


Figur 18. Depthtexture Unsharpening resultat, till vänster visas modellen med diffusbelysning. I mitten visas modellen med diffusbelysning och Ambient Occlusion och till höger visas Ambient Occlusiondatan. För större bild se Appendix 3.

6.3 Analys

6.3.1 Visuellt resultat

Figur 19 nedan visar en jämförelsebild mellan de olika teknikerna. Utifrån den reducering av ljus som visas kan de jämföras mot idealit vilket bilden längst upp till vänster, förberäknad Ambient Occlusion, kan sägas utgöra. Förberäknad Ambient Occlusion ger som kan ses en naturlig och mjuk skugga i de skymda ytorna av modellen emedan både Creaseshader (nere till vänster) och Depthtexture Unsharpening (nere till höger) inte alls skuggor dolda partier utan snarare framhäver skarpa kanter. Crysis shader (uppe till höger) ger ett resultat som närmast kan jämföras med resultatet från förberäknad Ambient Occlusion. Där syns hur ingröpningar i modellen om än endast ytliga sådana ges skuggor och således framhäver modellen detaljer mer naturligt.



Figur 19. Jämförelse mellan slutresultat. Överst till höger visas Förberäknad Ambient Occlusion, överst till vänster Crysis Shader. Nere till vänster visas Crease Shader och nere till höger visas Depthtexture Unsharpening Shader.

6.3.2 **Förberäknad Ambient Occlusion**

Kvalitéen på förberäknad Ambient Occlusion är jämförbar med riktig Global Illumination och således är tekniken en fullgod approximation. Den stora nackdelen är dock att tekniken endast i det utförande som presenterats i rapporten tillåter att blockering beräknas för statiska objekt. Vidare så är minneskraven för den förberäknad Ambient Occlusion väldigt stort och för komplexa scener kan texturminnet som krävs bli en avgörande faktor för användandet av tekniken. Förberäkningstiden kan ofta vara en överkomlig kostnad men i stora dynamiska projekt där scenens upplägg ofta förändras kan det bli ett onödigt och klumpigt delsteg i produktionen att behöva omberäkna blockeringen för scenen då den uppdateras.

6.3.3 **Screen Space Ambient Occlusion**

Creaseshadern lider tyvärr för mycket av tvånget att använda normalerna i beräkningen, något som i många fall är onödigt dyrt. Vidare så är Depthtexture Unsharpeningteknikens tillvägagångssätt klart fördelaktigt i enkelhet och prestanda men går tyvärr inte att applicera i många scener, det kan klart vara en avvägning att göra om applikationen handahåller sådana scener där det passar in bra. Crysis shader tycks mest lämpad för mer organiska scener där många mindre skrymslen och detaljer syns även om det också fungerar bra på större organiska geometrier. Problemet med Crysis shader är att den agerar i Shadermodel 3.0 något som fortfarande har svagt stöd hos många grafikkort. Det finns dock förenklingar av Crysis shader som ger ifrån ett lika fullgott resultat med färre samplingspunkter.

7 Slutsats

Rapporten har framställt tre stycken metoder för att generera Ambient Occlusion i realtid med hjälp av shaders samt diskuterat implementationer som förberäknar Ambient Occlusion att lagra i texturer. Analysen har generaliserat de egenskaper som teknikerna äger och problemställningen har lösts. Nedan diskuteras resultatet och en slutsats görs innan framtida arbeten diskuteras.

7.1 Diskussion

I dagsläget så är kvaliten av förberäknad Ambient Occlusion på alla sätt överlägsen de realtidsberäknade varianter som presenterats. Detta är inte heller konstigt med tanke på den mer detaljerade beräkningsprocessen och därigenom också kostnaden för tekniken. Tekniken är dock svår att applicera i realtidsapplikationer då den endast stödjer statiska scener, något som sällan förekommer i realtidsprogram, speciellt inte interaktiva sådana. Förberäknad Ambient Occlusion kommer således sannolikt endast att tillämpas inom offlinerendering där det kan användas som en optimering(Landis 2002). Screen Spaceteknikerna också de många nackdelar. Det framförallt viktigaste problemet med teknikerna är att resultatet som ges inte alls är jämförbart med ett korrekt Ambient Occlusionresultat. Teknikerna kan dock fortfarande användas som en stilistisk förbättring till förmån för det naturliga utseendet av en scen. Analysen redogör även att för detaljerade eller högtesselerade scener kan förbättras av teknikerna. Små skiftningar och brytningar i modellerna kan framhävas och således kan ett bättre djup erhållas i scenen. Framförallt Crysis shader ger ett bra resultat i dessa scener och i exempelvis scener med mycket detaljerad vegetation som exempelvis i spelet Crysis (Mittring 2007) ger shadern ett djup i scenen. Shaderteknikerna har dock på grund av det mycket enkla utförande tyvärr få förbättringar som kan göras för att resultatet skall bli mer generellt oberoende på scenes. Slutsatsen är alltså att shaderteknikerna kan appliceras för vissa scener med gott resultat men är ytterst beroende på geometrins utformning.

7.2 Fortsatt arbete

Screen Space shaders för Ambient Occlusion har många som kan optimeras och förbättras. Olika funktioner för blockeringstermer går att skapa och bättre algoritmer för att sampla längre bort från orginaltexeln utan att ta hänsyn till geografiskt avlägnsa punkter skulle kunna förbättra resultatet markant. Likaså kan tekniker för att bättre ge resultatet oskärpa (för att göra det mjukare) utvecklas. Men om ämnet skall uttökas lite vidare så finns också goda möjligheter för att använda också normaler för att exempelvis tillåta diffust ljus att reflekteras runt i skärmbilden för att erhålla Screen Space Radiosity.

Andra metoder användande Screen Spaceberäkningar skulle också kunna poneras, exempelvis att skugga omkringliggande geometri med hjälp av s.k. imposters. Vidare skulle mer data kunna användas i beräkningen för djupvärdena, exempelvis skulle scenen kunna renderas där kameran skiftar något i position för att få effekten av Depth of Field, genom detta skulle annars gömd djupinformation kunna extraheras och användas i Screen Space då Ambient Occlusionvärdena beräknas. Grafikkortet kan också användas på många andra sätt i beräkningen av Ambient Occlusion. Mest intiutivt vore att beräkna Ray Tracing på själva grafikkortet. Annars om mer öppenhet i geometridata skulle ges i grafikkorten skulle modeller exempelvis kunna skugga områden omkring modellen med hjälp av någon slags tredimensionell glödning där

närliggande geometri erhåller blockering successivt som modellerna renderas redan i vertexshadern innan geometri har klippts bort. Som grafikkorten blir mer öppna för utvecklarna kommer mer processorkapacitet kunna användas till mer orginella lösningar och inom de kommande åren torde många långt mycket mer orginella lösningar på problemet Ambient Occlusion utvecklas. Tids nog kommer Global Illumination kunna beräknas i realtid men även då kommer den cinematiska grafiken att vara än närmre verkligheten och realtidsapplikationer kommer att behöva approximera problem även då.

8 Tillkännagivanden

Tack uträckes till kreatören bakom modellen använd för att påvisa resultatet av SSAOteknikerna, Nicholas Hurtig, tack och bock.

9 Referenser

Pan, M. (2008) *Screen-Space Ambient Occlusion with Composite Nodes.* Tillgänglig på Internet: http://mpan3.homeip.net/?ssaowcn (Hämtad 08.03.03)

Akenine-Möller, T., Haines, E. (2004) *Real-Time Rendering.* Wellesley: A K Peters Ltd.

Bunnell, M. (2005) Dynamic Ambient Occlusion and Indirect Lighting. I: Pharr, M., Fernando, R (red:er), *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation* (s. 223-233). Upper Saddle River, NJ : Addison-Wesley

Engel, W. (red) (2005) *Shader X4 Advanced Rendering Techniques.* Hingham, Mass. : Charles River Media

Fernando, R. (red) (2004) *Gpu Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Upper Saddle River, NJ : Addison-Wesley.

Fox, M. (2007) *Ambient Occlusion Crease Shading.* Tillgänglig på Internet: http://www.shalinor.com/code.html (Hämtad 08.03.03)

Gear6 (2007) *Solving Rendering Bottlenecks in Computer Animation.* Tillgänglig på Internet:
http://www.gear6.com/files/Gear6_WP_Animation.pdf (Hämtad 08.03.03)

Kontkanen, J. Laine, S. (2005) Ambient Occlusion Fields. *Symposium on Interactive 3D Graphics and Games*, 41-48

Landis, H. (2002). Production-Ready Global Illumination. *Sigraph 2002 course 16: Renderman in Production*, 87-101

Malmer, M., Malmer, F., Assarsson, U. & Holzschuch, N. (2007) Fast Precomputed Ambient Occlusion for Proximity Shadows. *Journal of Graphics Tools*, *12*, 57-71

Mittring, M. (2007) Finding Next Gen: CryEngine 2. *International Conference on Computer Graphics and Interactive Techniques*, 97-121

Shanmugam, P., Ariak, O. (2007). Hardware Accelerated Ambient Occlusion Techniques on GPUs. *Symposium on Interactive 3D Graphics and Games*, 73-80

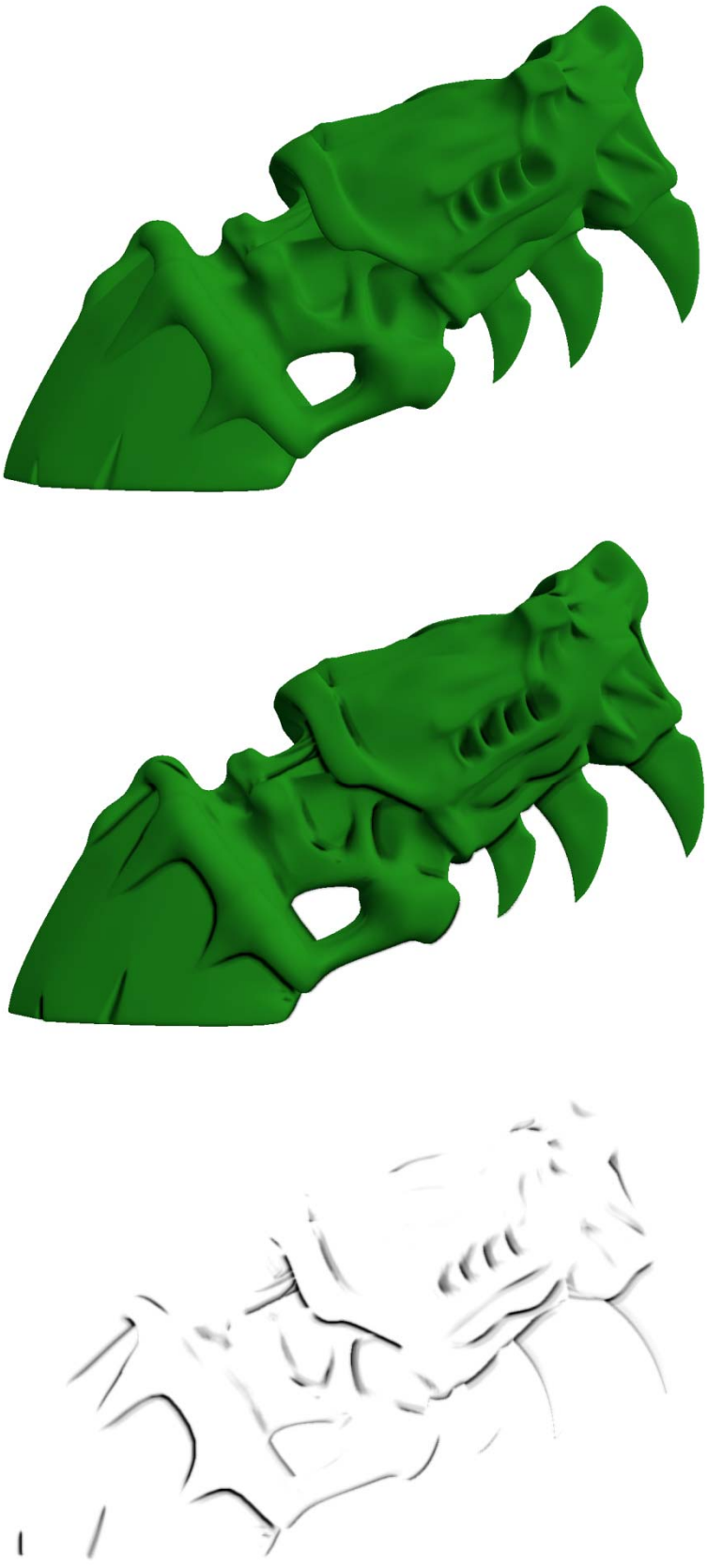
ST-Laurent, S. (2004). *Shaders For Game Programmers And Artists*. Boston, MA : Thomson/Course

Pharr, M., Fernando, R. (red:er) (2005) *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation.* Upper Saddle River, NJ : Addison-Wesley

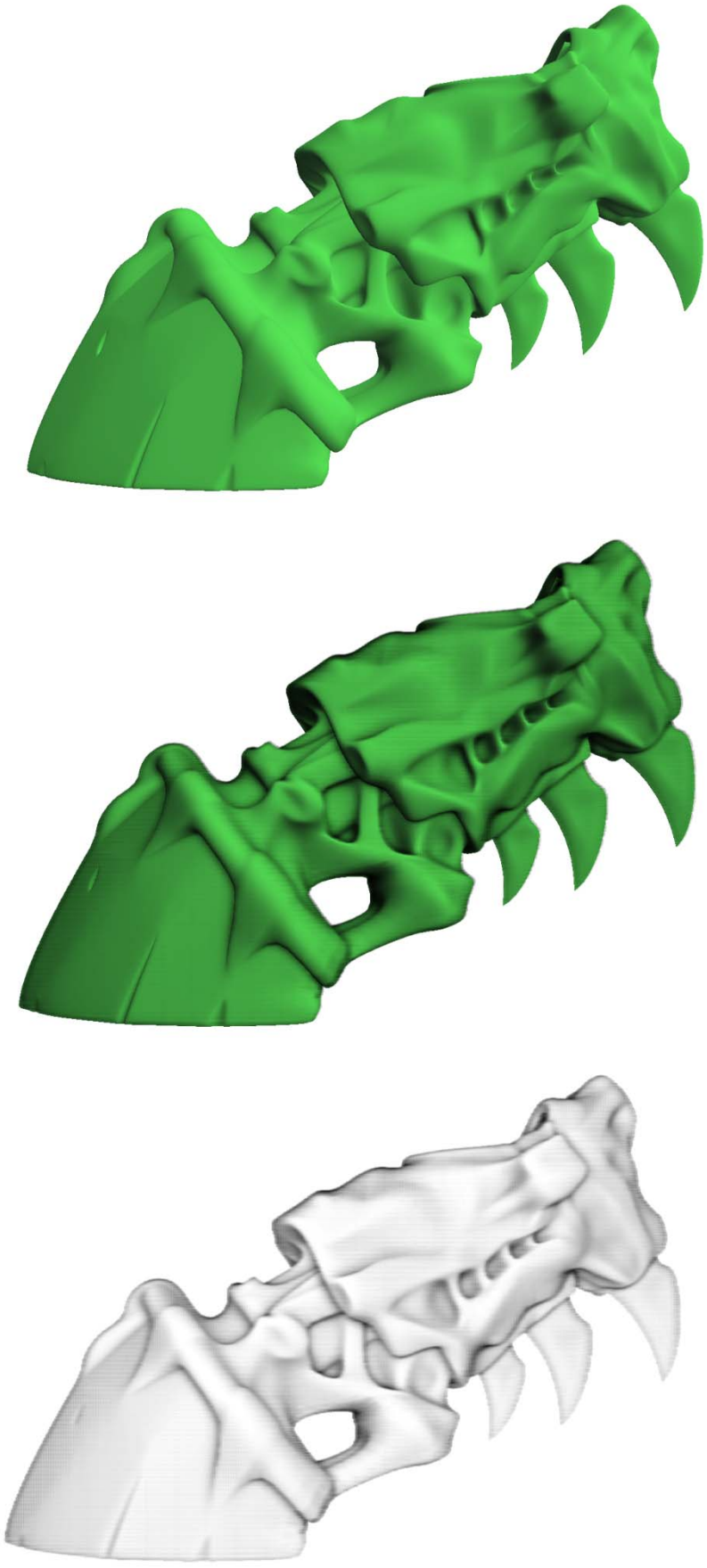
Weiss, M. (2006) *Data Structures and Algorithm Analysis in C++*. London: Addison-Weasly

Zhukov, S., Iones, A., Kronin, G. (1998) An Ambient Light Illumination Model. *Rendering Techniques '98 (Proceedings of the 9th EG Workshop on Rendering*, 45-56

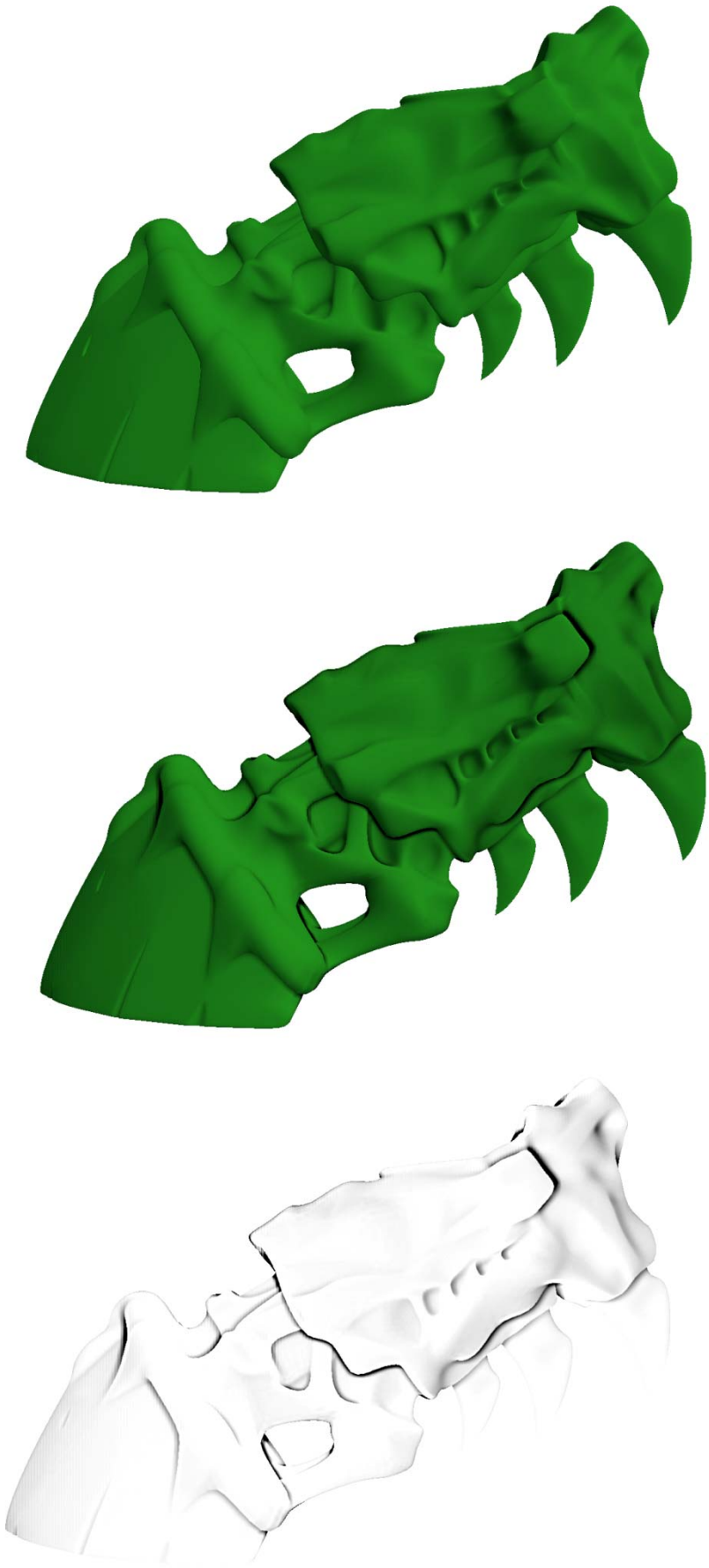
10 Appendix



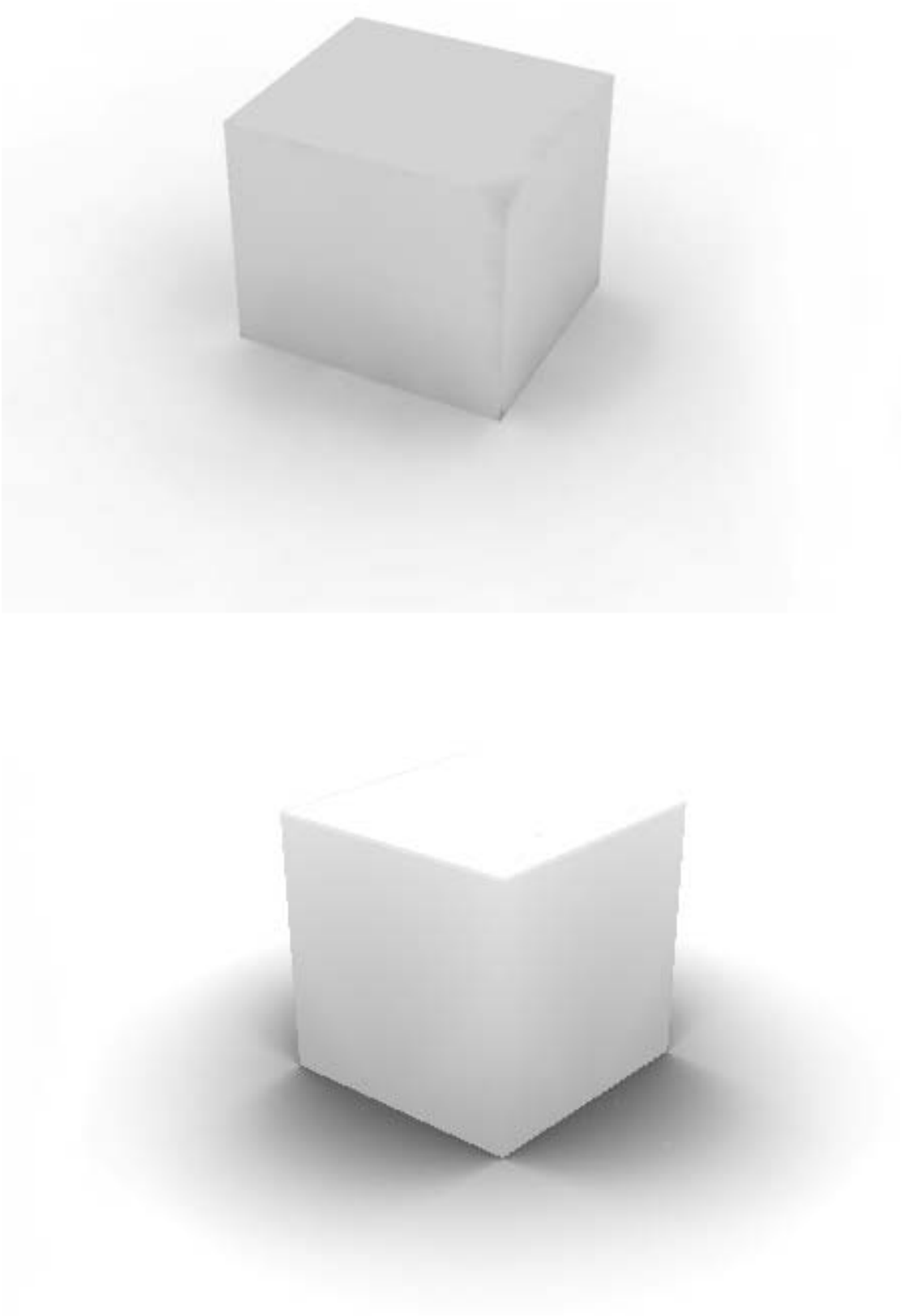
Appendix 1. Creaseshading, förstorad jämförelsebild



Appendix 2. Crysis shader, förstorad jämförelsebild



Appendix 3. Depthtexture unsharpening, förstorad jämförelsebild



Appendix 4. Förberäknad Ambient Occlusion, förstorad jämförelse bild. Ovan visas Global Illumination, nedan visas Ambient Occlusion