EXAMENSARBETE

Tesselering och Displacement

Ökad realism inom realtidsgrafik

Jonas Malm
2013

Teknologie kandidatexamen
Datorgrafik

Luleå tekniska universitet
Institutionen för konst, kommunikation och lärande

LULEÅ TEKNISKA UNIVERSITET

Tesselering och Displacement

Ökad realism inom realtidsgrafik

Jonas Malm

Examensarbete

Juni, 2013

Teknologie Kandidatexamen

Datorgrafik

Förord

Detta examensarbete är den sista delen av tre år som student på gscept, LTU Skellefteå.

Jag vill tacka mina handledare Samuel Lundsten och Gustav Sterbrant för hjälp och stöd under hela utbildningstiden.

Resultatet i denna rapport är en blandning av egna experiment, slutsatser och tankar om de realismgivande tekniker som används inom realtidsgrafik.

Denna rapport är det slutgiltiga steget i min kandidatexamen inom datorgrafik.

Sammanfattning

Optimering är ett ständigt problem att brottas med som inom spelbranschen. En datorgrafiker som arbetar med spel ska inte bara skapa så estetiskt tilltalande modeller som möjligt utan måste även få dessa att fungera tillsammans med de tekniska krav som ställs utav industrins spelmotorer och hårdvara.
Tesselering och displacement mapping är exempel på industrins olika tekniker för att ge spel en mer realistisk rendering genom att förbättra siluetter, ge mer korrekta detaljer och samtidigt avbelasta tunga beräkningar från CPU till GPU genom att utföra geometrins polygonhantering på grafikkortet.

Tesselering och Displacement Mapping har börjat öppna upp nya möjligheter för hur moderna spel kan se ut och som datorgrafiker så måste du kunna identifiera när, var och hur dessa tekniker bör användas.

Detta examensarbete går igenom modellering, bakning och användning av displacement mapping på tesselerade geometrier inom en rad av grafiska användningsområden med hjälp av ett par enkla tester i hopp om att göra dig som läsare redo för att skapa realistiska modeller.

Abstract

Optimization is a constant problem you struggle with in the gaming industry. The goal for as a computer graphic artists working with games is not only create aesthetically pleasing models, but must also to get these to work along with the technical demands set by the game engines and hardware of the industry.
Tessellation and displacement mapping are examples of the industry's techniques to give games a more realistic rendering by enhancing silhouettes, more accurate details while take heavy calculations from the CPU to the GPU by performing geometry polygon management on the graphics card.

Tessellation and Displacement Mapping is opening up to new possibilities for how modern games may look and as a graphical artist you must be able to identify when, where and how to use these techniques.

This thesis takes you through modeling, baking and the usage of displacement mapping on tessellated geometries withing a variety of graphical applications with a few simple tests in hopes of preparing you, the reader for the creation of realistic models.

Ordlista

Vertriser – En koordinat i en tredimensionell rymd som tillsammans med flera andra bygger upp polygoner.

Polygon – En tredimensionell yta mellan minst tre punkter, vertriser.
Mesh – Ett nät av polygoner.

Shader – renderings program som skapar korrekt ljus och färg i en bild på en grafiskt processerande hårdvara.

Normal – Vektor som bestämmer både ytriktning och hårdhet/mjukhet på angränsningar mellan olika vertriser.

Normal Map – Texturkara som projicerar detaljer från en högupplöst modell till en lågupplöst version av samma modell.
Texel – Texturpixel. Grundläggande enheten i texturespace.
CPU – Centralprocessor. Datorenhet som utför programoperationer som beräknar och hanterar data.

GPU – Grafikprocessor. Mikroprocessor som är optimerad för grafikrelaterade beräkningar på ditt grafikkort. Hanterar signaler som skickas från din dator till din bildskärm.

Frame rate – Antalet bilder som kan visas per sekund. Högre bildhastighet ger i jämnare rörelse i film och spel.

Spelmotor – En spelmotor är en mjukvara som bygger upp strukturer inom ett spel. Spelmotorn renderar ut grafiken och hanterar fysikberäkningar och annat som behövs för att ett spel skall fungera.

Editor – Program i en spelmotor där du använder dig av en samling verktyg för att placera ut, sortera och bygga ihop dina assets till ett färdigt spel.

Cryengine – Modern spelmotor. Utvecklas av CryTech.
Frostbite3 – Modern spelmotor.
UDK – Står för Unreal Development Kit, och är en modern spelmotor.

Unity – Modern spelmotor.

LOD – Level of Detail. Inställning som styr vilken detaljnivå ett objekt bör ha baserat på avståndet till renderingskameran.

Mipmaps – Kopior av en textur i olika storlek och detaljnivå som renderas beroende på avstånd från kameran.
Units – Avståndsenhet i en spelmotor.

Dokument Historia

Version	Datum	Reason for Change
1,0	30/05-2013	Första versionen
1,1	9/6-2013	Andra versionen

Innehållsförteckning

1 Inledning 6
1.1 Bakgrund 6
1.2 Problembeskrivning 6
1.3 Syfte 7
1.4 Avgräsningar 7

2 Teori 8
2.1 Tesselering 8
2.1.1 Tessellation Control Shader *10*
2.1.2 Primitive Generator 11
2.1.3 Tessellation Evaluation Shader 11
2.2 Smooth Tessellation 11
2.2.1 Curved PN-triangles 11
2.2.2 Phong Tessellation 12
2.2.3 Adaptive Tessellation 13
2.3 Flat Tessellation 14
2.4 Displacement Mapping 15

3 Metod 16
3.1 Verktyg 16
3.2 Displacement Mapping *17*
3.2.1 Highpoly Construktion *17*
3.2.2 Lowpoly Construction *18*
3.2.3 Baking *19*
3.2.4 Problem och Artefakter 19
3.3 Tesselering och Displacement - Användningsområden 23
3.3.1 Landskap 23
3.3.1.1 Terrain Editors 23
3.3.1.2 Terräng Geometrier 26
3.3.2 Karaktärer 28
3.3.2.1 Animationer 32
3.3.2.2 Previsualization 32
3.3.3 Props 33
3.4 Metodkritik 35

4 Resultat 36
4.1 Displacement Mapping 36
4.2 Tesselering av Terräng 36
4.3 Tesselering av Karaktärer 36
4.4 Tesselering av Props 37

5 Diskussion 38

6 Slutsats 39

7 Referenser 40

1 Inledning

1.1 Bakgrund

Dagens spelindustrin strävar efter att kunna rendera bilder i realtid så snabbt och så smidigt som möjligt. Alla tekniska och estetiska aspekter av ett spel bör vara så optimerade som möjligt för att kunna ge spelaren en flytande spelupplevelse utan att kunden skall behöva köpa den absolut senaste och dyraste hårdvaran. Denna optimering har länge varit en flaskhals för skapandet av realism och inlevelse inom spelbranchen och utvecklare har tvingats använda sig av modellerings-, texturerings- och ljussättnings-tekniker som är lättare för den aktuella hårdvaran att hantera.

De tekniska framsteg som gjorts under de senaste åren har givit spelindustrin och dess kunder tillgång till snabbare, kraftigare och billigare hårdvara vilket i sin tur givit utvecklare möjligheten att tänja på gränserna i sina respektive spelmotorer.

Grafikkort kan rendera trianglar väldigt effektivt, så en vanlig pipeline för spelgeometrier involverar hanteringen av triangulerade meshes.
Antalet trianglar är avgörande för renderingshastigheten i alla spel och optimering av geometrier betyder oftast kantiga siluetter och en plattare matrialkänsla. Detta blir mer tydligt ju närmare spelaren beskådar dina geometrier. Att öka detaljnivå och polygon antal direkt på din modell för att motverka detta kan blir snabbt för tungt för din dator att hantera. Detta beror på att hantering av 3d modeller sker först på din centralprocessor (CPU). Modellens trianglar måste sedan skickas till grafikprocessorn (GPU) för att renderas och visas på din skärm. Denna process kan vara krävande då minnes bandbredden mellan din CPU och GPU är begränsad.

Tesselering och displacement maps är ett exempel på industrins olika tekniker för att ge spel en mer realistisk rendering genom att förbättra siluetter, mer korrekta detaljer och samtidigt avbelasta tunga beräkningar från CPU till GPU genom att utföra geometrins polygonhantering på grafikkortet. Allt detta utan att förlama renderingshastigheten.

1.2 Problembeskrivning

Nya tekniker ställer nya krav på grafiker. Trots kraftigare hårdvara så måste man fortfarande ha optimering i åtanke och att man endast adderar detalj på de objekt och i de områden, där det gör mest nytta. En grafiker bör därför inte endast lära sig arbetsflödet kring tesselering och displacement mapping utan också hålla koll på vilka objekt och grafiska arbetsområden där det kan göra skillnad.

1.3 Syfte

Med denna rapport så vill jag ge en bild av hur arbetsflödet kring tesselering ser ut och vilka tekniker som används i de spelmotorer som finns på marknaden. Jag vill även belysa skillander och likheter i arbetsflödet mellan användandet av Displacement maps och Normal maps och hur man kan kombinera de två. Mitt mål med denna rapport är att en grafiker ska lättare kunna identifiera när, hur och var det är bäst att använda sig av dessa tekniker inom realtidsgrafik för att få ett så snyggt och optimerat resultat som möjligt.

1.4 Avgräsningar

Detta dokument kommer endast fokusera på tesselering inom realtidsgrafik utifrån en grafikers perspektiv och förutsätter att läsaren har en viss förkunskap inom de olika arbetssätt och arbetsroller som kan förväntas av en datorgrafiker.

Hänvisningar till mer ingående information gällande den programmering och de matematiska formler som berör ämnet kan förekomma.

2 Teori

För att förstå vad du som grafiker kan förvänta dig av en renderingspipeline som innehåller ett tesselerings workflow så måste du först förstå vad det som händer i en shader med tesseleringsmöjligheter.

2.1 Tesselering

Tesselering innebär en uppbrytning av en mesh till flertalet mindre primitiver. Denna uppbrytning sker i en Tesselerings shader som i sin tur är uppdelad i tre steg och är en viktig del av vertex hanteringen i den nya renderings pipelinen.

Huvudproblemet som tesselering behandlar är 3D-modellernas detaljrikedom och polygonantal. En detaljerad modell betyder fler trianglar som kräver mer datorkraft för att bearbetas. När samma modell befinner sig på ett större avstånd från kameran så vill man använda en modell med lägre polygonantal för att kunna fokusera resurserna på att beräkna de objekt som är närmast kameran. Det är helt enkelt en fråga om att balansera GPU krafter och fördela mer kraft till prioriterade området där små detaljer är mer märkbara.

Lösningen på detta är att utföra tesselering direkt på din GPU, som oftast är optimerad för parallella operationer. Detta kan även kallas för hardware Tessellation. Tesselering kan hantera flera polygoner och patches parallellt eftersom de beräknas oberoende av varandra.

Genom att implementera tesselation i din hårdvara, kan en grafisk pipeline hantera lowpoly modeller och sedan rendera med högre detalj. Tesselering som implementeras på din hårdvara kan generera en otrolig mängd visuella detaljer och ge stöd för deplacement mappning utan att behöva hantera onödigt tunga modellstorlekar som kan leda till sämre frame rate.

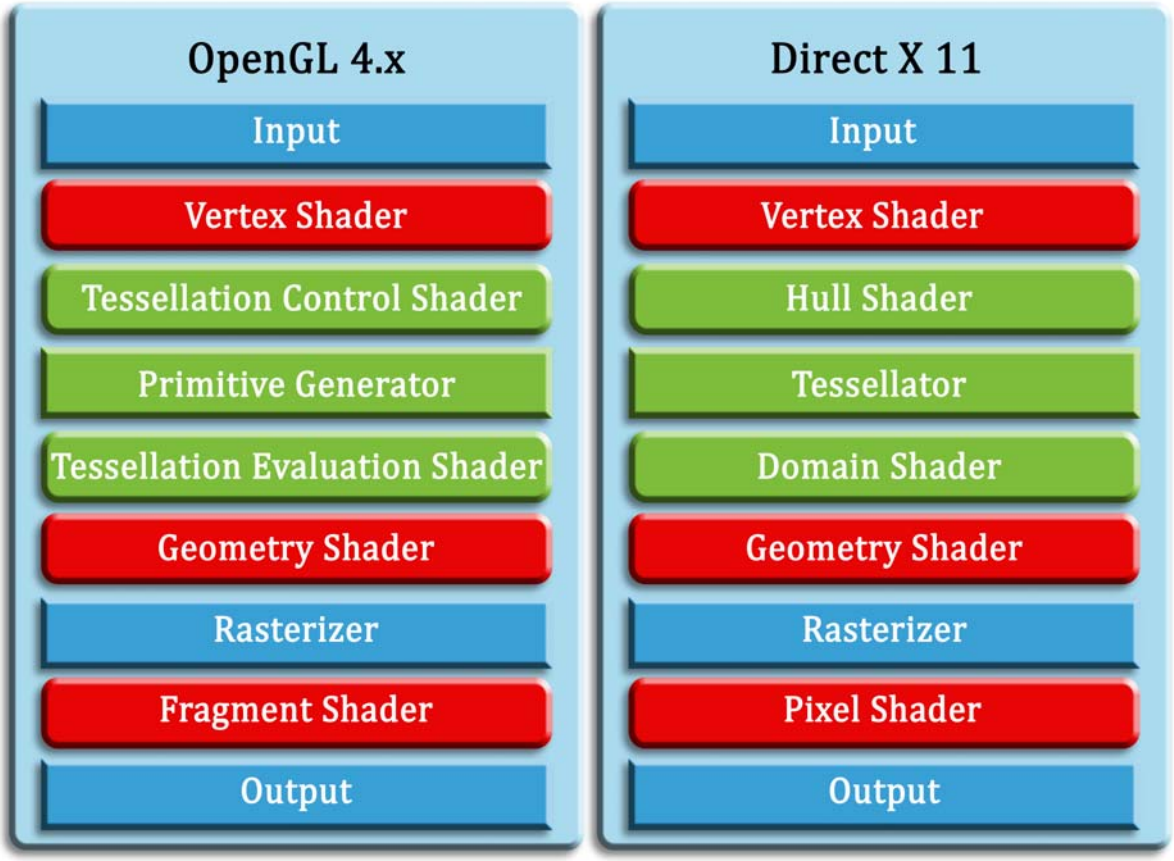


Bild 1: *Renderings pipelines inom realtidsgrafik för OpenGL 4.x respektive Direct X 11. Tesselerings Shadern är markerad med grönt. Programmeringsbara Shadersteg har avrundad form, medan fasta funktioner har kantig form.*

I detta kapitel kommer jag gå igenom de shadersteg som finns i en tesselerings shader - renderingspipelinen. Det finns skillnader mellan DirectX 11 shaders och OpenGL 4.x shaders rent tekniskt, men dessa skillnader är ur en grafikers perspektiv inte avgörande för ett bra resultat. I denna förklaring av hur tesselerings shaders fungerar så håller jag mig till OpenGL 4.x pipelinen.

Det första steget i OpenGL 4.x tesselerings shader kallas för Tessellation Control Shader (TCS), det fasta funktionssteget kallas för Primitive Generator (PG) och det andra shader steget kallas Tessellation Evaluation Shader (TES).

Det första som händer är att din geometri input passerar en *vertex shader* som identifierar geometrins vertriser och processerar dessa till en ordnad sekvens av kontroll punkter (*control points)* som bildar enkla primitivesr såsom isolinjer, quads eller trianglar. Denna enkla primitiv som skapas kallas för en *patch* som sedan skickas in i Tesselerings Shadern.

2.1.1 Tessellation Control Shader

En Tessellation Control Shader (TCS) körs för varje kontroll punkt som den läser in och avgör hur mycket tessellation som skall utföras per patch samt hur de nya vertriserna ska genereras.

En patch beräknas oberoende av de kringliggande kontrollpunkterna vilket leder till att Tesselerings Shadern kan beräkna många av dessa operationer samtidigt. Grafikkort är optimerade för sådana parallella operationer och detta är anledningen till att tesselering direkt på en GPU (Hardware Tessellation) är fördelaktigt.

Det viktigaste per-patch resultat en TCS skapar är Tessellation Levels (TLs). Tessellation Levels kontrollerar antalet uppdelningar som ska ske i Primitive Generator steget. Tessellation Levels är två variabler som bestämmer den inre och den yttre uppdelningen av den aktuella patchen.

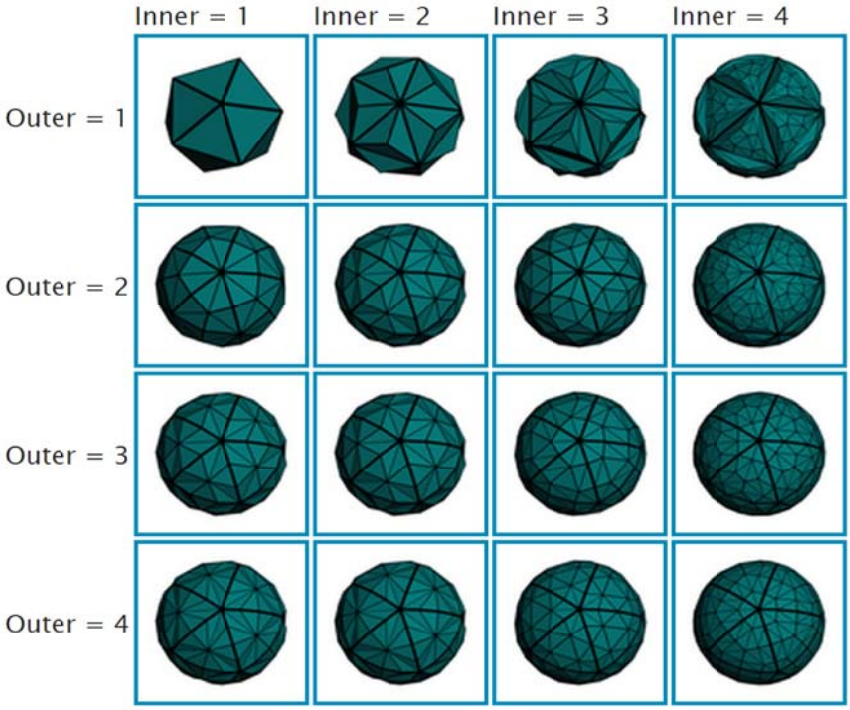


Bild 2. Diagram på hur tesselerings levels kan arbeta, triangel för traingel över hela geometrin. Bild från

Inner Tessellation Level bestämmer antalet inramade primitiver som ska skapas samtidigt som Outer Tessellation Level bestämmer hur många gånger varje edge ska delas upp. Dessa *levels* styrs ofta av en *Tessellation Factor* i en shader. Det är en variabel som du som grafiker kan påverka direkt i shaders och se direkta resultat.

Det är även här i TCS som geometrins Level of Detail (LOD) ställs in per patch. Detaljnivån på en geometri kan få en sömlös övergång från högupplöst till lågupplöst beroende på avståndet till kameran. Denna funktion kan oftast styras i en shader där du manuellt ställer in det max och minimum avstånd från kameran som du vill att tesseleringen ska ske.

2.1.2 Primitive Generator

Det är i detta fasta funktionssteg själva uppdelningen av geometrin sker. En Primitive Generator tar hänsyn till de Tessellation Levels som skickas från TCS och utför uppdelningen därefter.

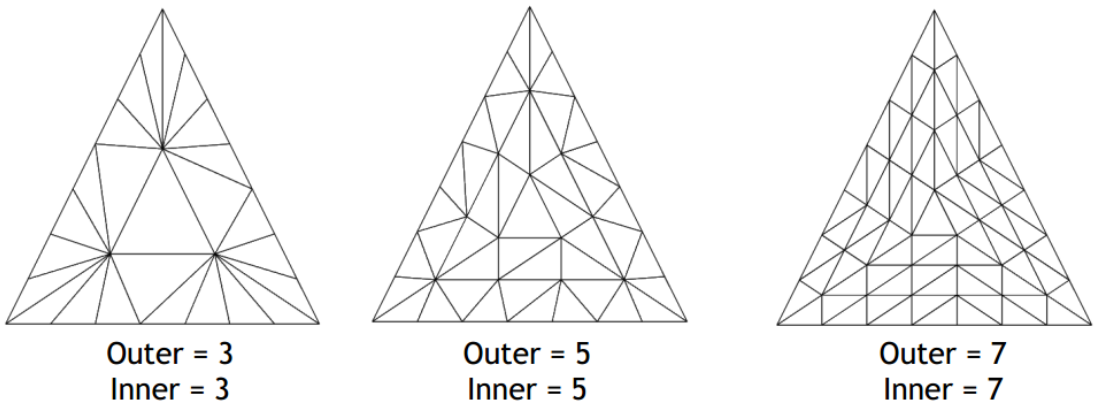


Bild 3. Olika Tessellation Levels.

2.1.3 Tessellation Evaluation Shader

Beräknar position på varje vertex som produceras av Primitive Generator och utvärderar ytan i UVW kordinater. En TES interpolerar attribut och tillämpar förskjutningen från displacement mappen genom att hämta ut den förskjutningsfaktorn som finns på varje texturkordinat för varje vertris att avslutningsvis skriva ut vertrisens slutgiltiga position

2.2 Smooth Tessellation

Det finns flertalet tekniker för att subdivera geometrier i din Tessellation Control Shader. Phong Tessellation och Curved PN Triangles fokuserar på att lösa problemet med kantinga siluetter genom att generera smooth tessellation över ytan på varje patch.

2.2.1 Curved PN Triangles

Curved PN Triangles[1] är ett triangel interpolations system som arbetar direkt på triangulerade geometrier vars hörn består av positioner och normaler (PN står för Point-Normal).

Först, för varje triangel i den aktuella meshen så skapas en triangulär eller kubisk patch som enbart härstammar från dess vertex positioner och normaler, ingen information från angränsande patches krävs. Denna patch tesseleras sedan för rendering.

Den resulterande ytan är jämnare än den tidigare polygonala ytan, men har inte tangent kontinuitet i allmänhet, och som resulterar i skuggning diskontinuiteter. För att dölja dessa diskontinuiteter i normalerna interpoleras självständigt använder antingen linjär eller kvadratisk interpolation. Dessa normaler är inte de sanna ytnormaler, men de ger ett slätt utseende till ytan.

Curved PN Triangles fokuserar på att ge en jämnare silhuett och en mer organisk form för triangel-baserade modeller för en minimal kostnad, utan att besvära modellens förberedelse, modifiering, och rendering. Curved PN Triangles kan ses som en triangel multiplikator då PN Traingles tar in en platt triangel och skickar ut många subtrianglar i samma format.

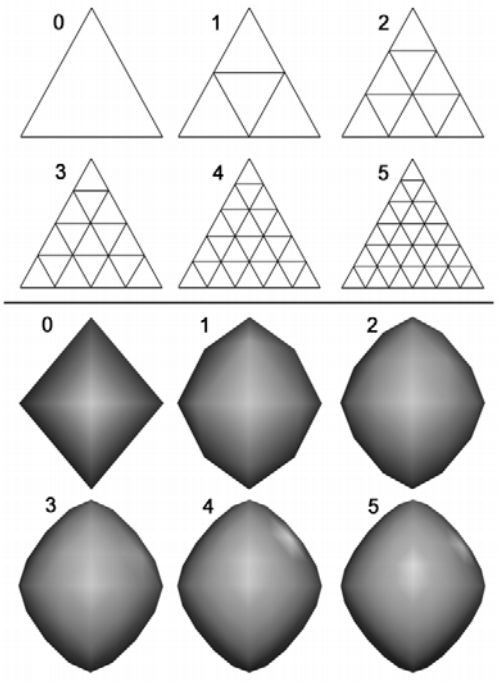


Bild 4. Hur Subtriangulering ser ut i LOD 0 till 5.
Övre: Uniform u,v grid. Undre: Visuell effekt på geometri.
Bild fr. Vlachos et al. 2001[1]

2.2.2 Phong Tessellation

Phong Tesselering[2] är en geometrisk version av en Phong Shader Interpolering som nu interpolerar vertex positioner iställer för normaler.

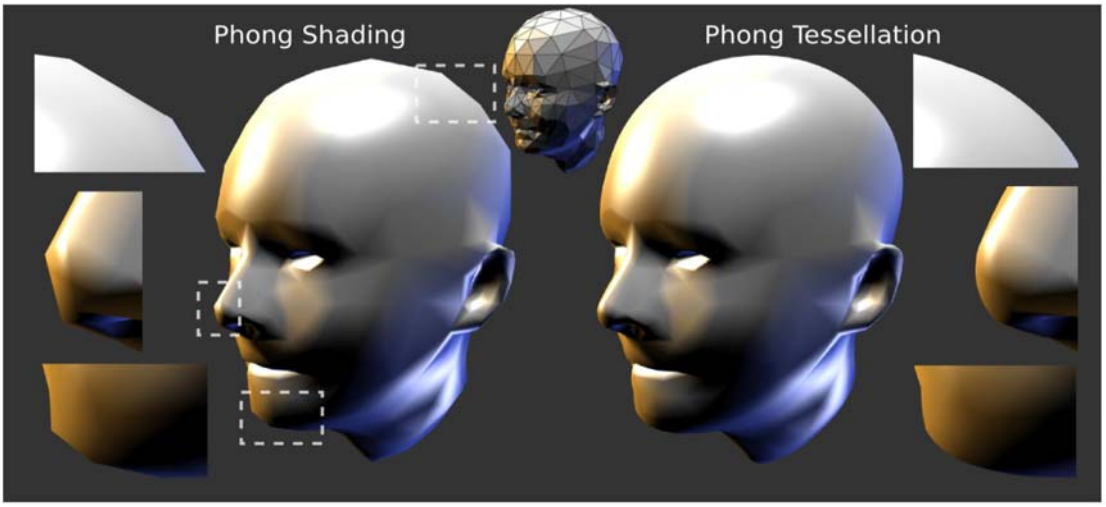


Bild 5: Phong Shading tillsammans med Phong Tesselering.
Bild från Phong Tessellation, T. Boubekeur, M. Alexa. TU Berlin.

Phong Tessellation genererar tangentplan för varje vertex i en patch, detta plan är anpassat till varje normal, och utför sedan en barycentric interpolation[3] på de definierande tangentplanen för att hitta en ny punkt **(p)**. Denna process illustreras här nedanför.

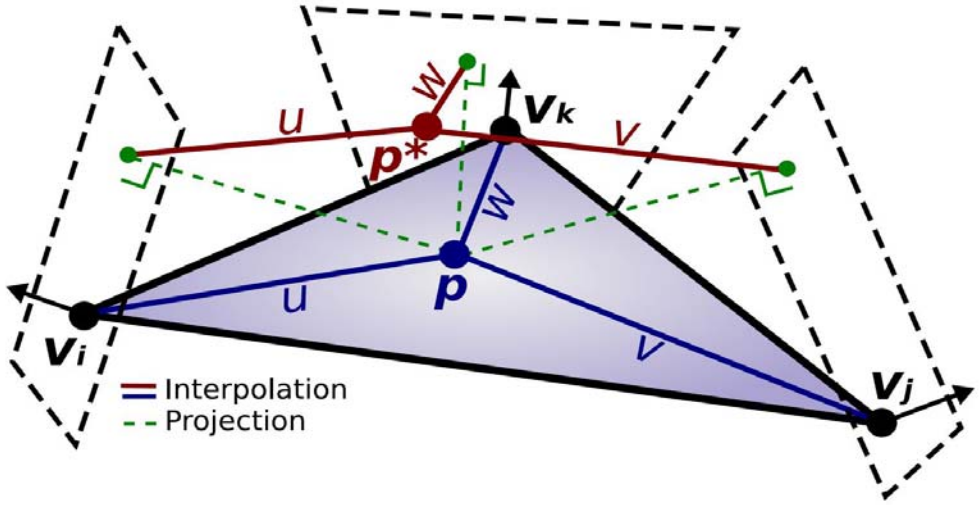


Bild 6: Phong Tessellation principen. Interpoleringen av vertex tangetplanen definerar en kurva för varje triangel. Bild från Boubekeur and Alexa 2008.

En nyligen genererad vertex **p** projiceras på de tre tangentplanen, visas med de gröna streckade linjerna. Den nya vertex plats **p*** hittas genom barycentric interpolering av tangentplanens projektion. Denna interpolering definerar en kurva för varje triangel som sedan används för att skapa en mjukare form.

Denna process skapar en mjuk yta precis som PN-triangles, men dess beräkningar är enklare eftersom inga kontrollpunkter måste beräknas, endast vertex positioner.

2.2.3 Adaptive Tessellation

Phong Tessellation och Curved PN Triangles fokuserar på att lösa problemet med kantinga siluetter genom att generera smooth tessellation över ytan på varje patch. Problemet är att detta sker över hela modellen, något som kan anses vara onödigt krävande. Den Tessellation Level som behövs för att generara en mjuk outline behövs inte appliceras över hela din geometri utan kan fokuseras på specifika områden.

Adaptive Tessellation kombinerar användningen av olika metrics[4][10] som läser av olika data från en input geomentri till en egen *contour metric* som känner av vilka patches som är nästan parallella med kameran. Denna *contour metric* kan sedan användas för att identifiera områden som inte behövs tesseleras lika mycket.

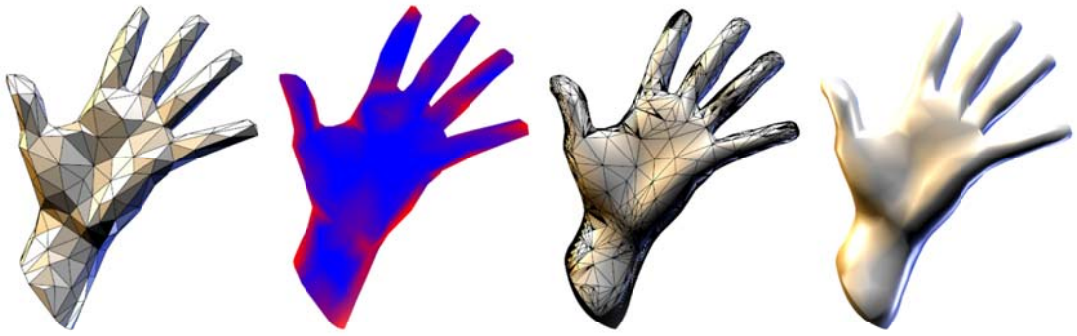


Bild 7. Geometri som går igenom Adaptive Tessellation.
1. Input. 2. Metric Computation. 3. PN-triangle Tessellation. 4. Result.

2.3 Flat Tessellation

Precis som det låter så går det att tesselera en geometri med ett platt resultat utan att mjuka upp patches. Flat Tessellation används oftast på hard surface modeller som man inte vill ska få en mjukare silhuett. Flat tessellation adderar endast polygoner innanför de redan existerande polygonerna på en lågupplöst modell och kan på så sätt göra geometrin redo för displacement mapping utan att direkt störa geometrins silhuett.

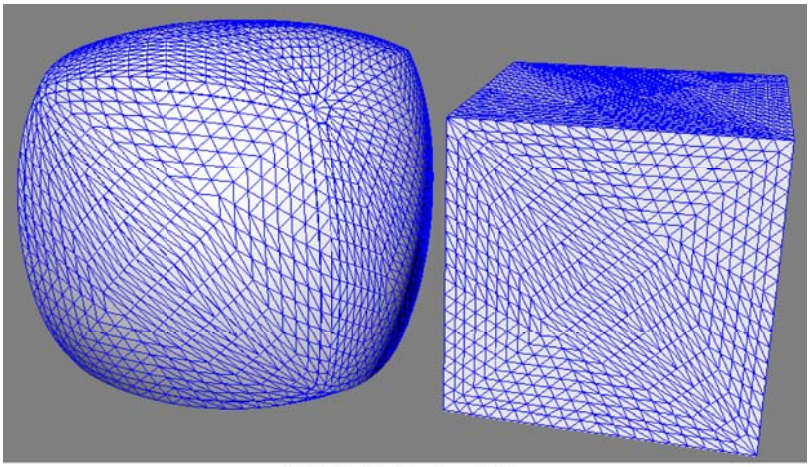


Bild 8. Soft edge kub.
PN-triangles t.v. Flat Tessellation t.h.

2.4 Displacement Mapping

Displacement Mapping är en teknik som blir allt vanligare inom realtidsgrafik. Det är används för att överföra detaljinformation från en högupplöst till en lågupplöst modell[5]. Displacement mapping ger dig möjlighet att förflytta eller förskuta (skjuta ut och dra in) geometri på din modell för att ge en större känsla av djup och detaljrikedom. Att använda sig av en Displacement Maps är ett bra sätt att låsa upp en extra nivå av detaljer som inte är möjligt med traditionella beräkningsmodeller såsom bump, normal och parallax mapping.

En heightmap är nästan detsamma en displacement map. I båda fallen så mäts avståndet mellan den hög och lågupplösta modellen, men skillnaden ligger i hur man bakar ut texturen.

En Heightmap är avståndet mellan en lågupplöst modell och en högupplöst modell i en riktning. Denna riktning är antingen den lågupplösta modellens vertrisnormal, vid *uniform ray distance* bakning, eller en *cage direction,* om du bakar ut med hjälp av en *cage*.

En Displacement Map är avståndet mellan en subdiverad version av en lågupplöst modell och en högupplöst modell. Mätningsriktningen är densamma som vid bakningen av en heightmap fast nu med större precision då vertrisnormalerna eller *cage directions* är fler och sitter tätare ihop.

I en 8-bitars bild så har varje färgkanal (röd, grön, blå, alpha) 256 färger. Varje färg representerar en höjd som displacement mappen kan pressa din geometri till, med 50% grått (r128,g128,b128) som ursprungshöjd. Användningen av en 8-bitars textur betyder då 256 nivåer som geometrin kan förskjutas till, 128 nedåt och 128 uppåt, per färgkanal. Samtidigt så har en 16 bitars kanal 65536 färger, dvs. 32768 nivåer uppåt och 32768 nivåer nedåt.

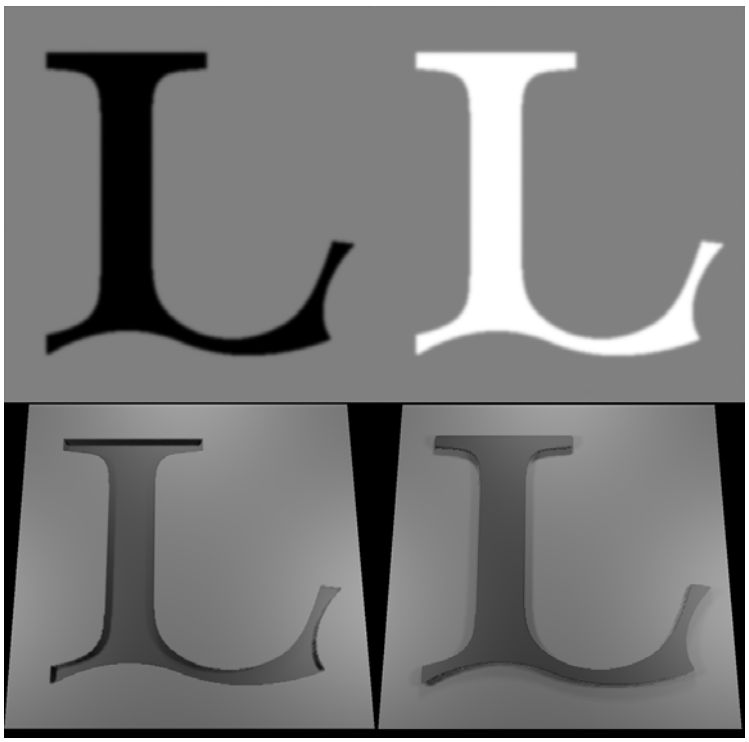


Bild 9. Displacement Nedåt (t.v.) Displacement Uppåt (t.h.)

3 Metod

Här nedan kommer jag gå igenom hur jag har arbetat med tesselering och displacement mapping och applicerat dessa på ett landskap, en karaktär och en prop i ett försök att skapa en bild av vad som kan vara viktigt att tänka på vid användningen av dessa tekniker.

3.1 Verktyg

Autodesk Maya

3d applikation där jag kommer använda mig av olika verktyg för att modellera och UV-mappa lågpolygon modeller.
Kommer även användas för Tesselerings/Displacement tester i viewport 2.0.

Autodesk Mudbox

3d applikation där jag med hjälp av ett antal verktyg kommer skulptera detaljer och baka diverse data till maps.

Adobe Photoshop

Bildhanteringsprogram där jag skapar texturekartor till modellerna.

Xnormal

En gratis programvara som används för att baka ut olika data från förhållandet mellan angivna högupplösta modeller och lågupplösta modeller. Jag använde det för att få ut normal, displacement och ambient occlusion data till texturer.

Nebula3

Spelmotor. Kommer användas för Tesselerings/Displacement tester.

3.2 Displacement Maps

För att kunna arbeta med tesselerade geometrier till fullo, så bör man ha en bra förståelse för hur displacement maps fungerar i praktiken.

3.2.1 Highpoly Construktion

Att skapa en högupplöst modell för bakning av displacement har ett liknande arbetsflöde som krävs för att kunna baka normal maps, med ett fåtal undantag.
Displacement maps kan endast förskjuta geometri utåt eller innåt från modellens yta, inte i någon annan riktning.

Detta medför att utstickande detaljer som böjs ut över annan geometri bör ha stödjande geometri i den lågupplösta modellen då dessa detaljer annars kommer att skjutas ut ur underliggande yta (se bild 10).

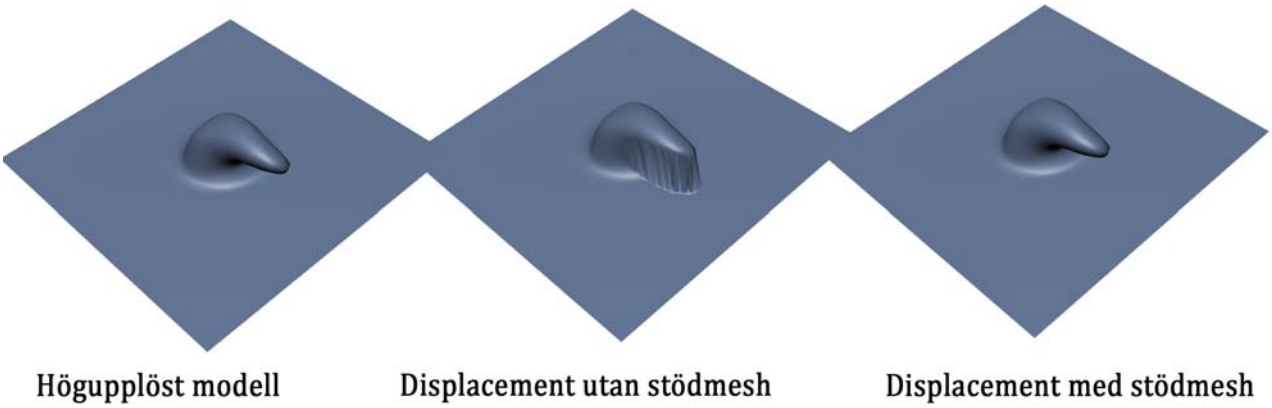


Bild 10. Denna artifakt uppstår när man bakar ut displacement maps med hjälp av den raytrace metod som är standard för att att baka ut andra texturer, t.ex. normal och AO.

3.2.2 Lowpoly Topologi

Att optimera ett objekt för spelgrafik kan vara en utmaning. Att få ner polygon antalet så mycket som möjligt men samtidigt behålla bra silhuett och goda deformerningar kan bidra till ett ojämt flöde av edgeloops över din geometri. Displacement maps kräver dock att man är lite mer givmild än normalt vid optimeringen av modellen.

Enkla tester visade fort att förskjutning av geometri på en låg upplöst modell fungerar bäst med en jämn topologi. Tester visade även att geometrier med jämnstora quads/trianglar gav ett bättre resultat.

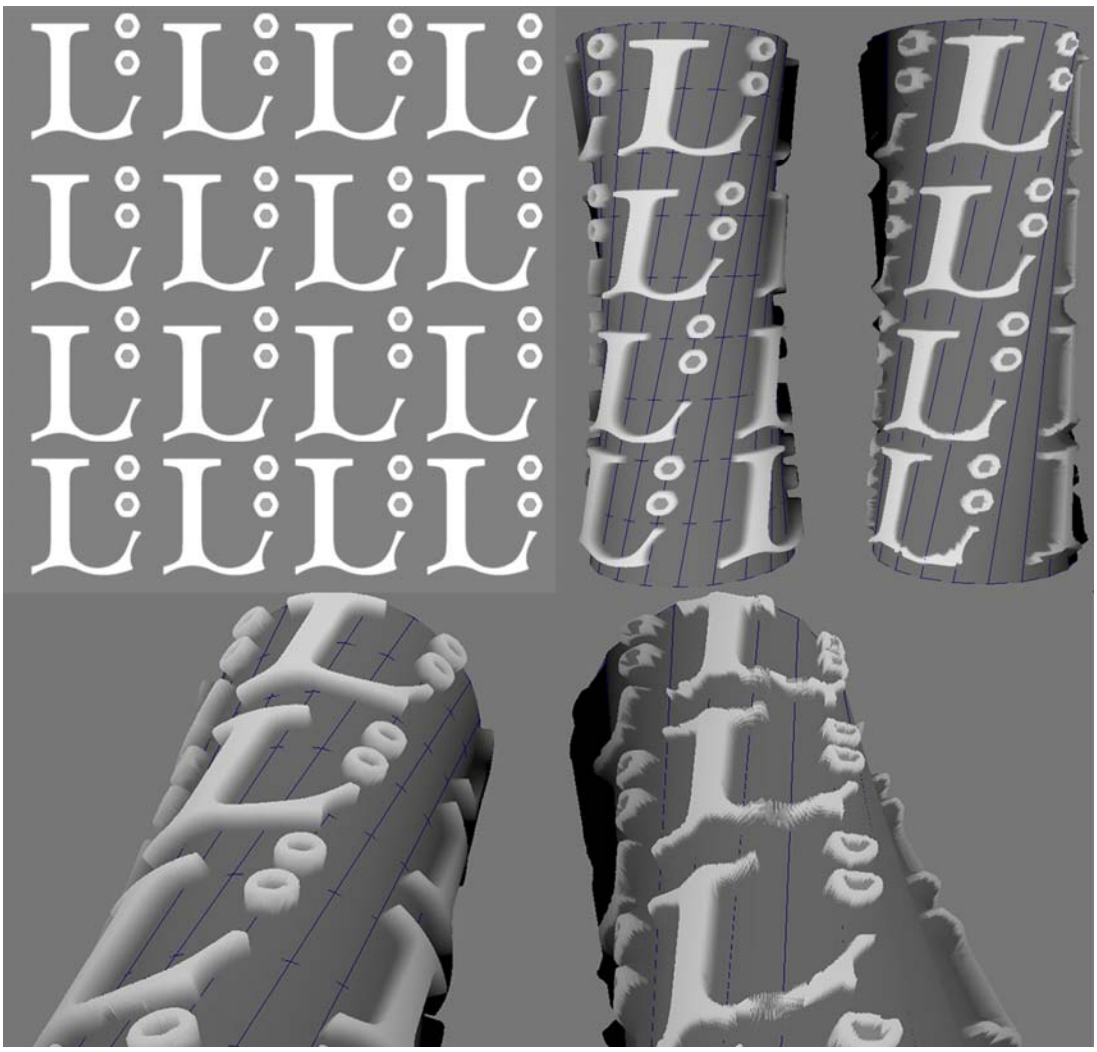


Bild 11. Displacement Map uppe till vänster appliceras på två cylindrar. Tydliga artefakter uppstår på den högra cylindern då denna inte har jämn topologi. (Cylinder t.v: 360 tris, t.h: 40 tris.)

Observera att detta inte direkt betyder att din geometri måste ha mycket fler polygoner, utan snarare ett jämnare storleksförhållande mellan dina quads/trianglar. Detta kan kräva lite mer planering och arbete på din lågupplösta modell, men det kan leda till mycket stabilare och mer realistiskt resultat.

3.2.3 Baking

Att baka ut en displacement map från en högupplöst till en lågupplöst modell har som sagt ett liknande arbetsflöde som krävs för att kunna baka normal maps. De flesta program som stödjer bakningen av normal-data och ambient occlusion stödjer oftast även bakning av displacement eller height maps.

Du borde ta extra hänsyn till sömmar, UV-stretching och mjuka/hårda normaler då du arbetar med Displacement maps. De artifakter som kan uppstå vid bakning av normal maps förstärks och syns ännu mer då förskjutningen från en displacement map kan få din geometri att spricka och skjuta igenom närliggande geometri.

Vid användningen av displacement maps så är kvaliteten av förskjutningen av din geometri direkt beroende av mängden färginformation i din textur.
När man bakar ut/exporterar en displacement map från sin 3d modell så är bit djupet avgörande för ett bra slutresultat.
Om möjligt, så bör man baka ut maps med 16- eller 32-bitar per kanal. Extra bitdjup kan ge större frihet att redigera innehållet och hjälper till att undvika artifakter. Dock bör man ha i åtanke vilken ändamål modellen har samt vilken motor man ska använda, då inte alla kan hantera 16-, och 32-bitars format.

Raytrace Baking

Raytrace bakning, som kan användas både vid uthämtandet av displacement och normal maps kräver även du som grafiker separerar vissa delar av din geometri för att undvika artefakter i trånga områden. Trots denna uppdelning av geometrier så bör man baka från hela modellen samtidigt då displacement värderna kan beräknas olika.

Subdivision Baking

Skulpterings program såsom Autodesk Mudbox har en alternativ bakningsmetod. Denna metod kräver dock att du arbetar med en grundgeometri(basemesh) som har en UV-map samt att din target mesh och din source mesh har samma topologi. Denna metod kan ge dig mindre artifakter i trånga utrymmen på din modell då metoden utgår från vertex förhållanderna istället för seach distance raycasting som ofta kan kollidera med närliggande geometrier.

3.2.4 Artefakter och Problem

Bit-storlek

Att baka ut i större bitdjup än nödvändigt kan vara bra för att vara säker på att man använder sig av så hög kvalitet som möjligt i sitt arbete. Om man arbetar med t.ex. UDK som inte har stöd för 16-, eller 32-bits texturer, så blir man dock tvungen att pressa ihop sin textur till 8-bitar, vilket bör göras utan att tappa alltför mycket detalj i processen.

Banding

Ett mindre bitdjup kan resultera i *banding* då det inte finns tillräckligt med bildinformation för att korrekt definiera ett område som ska förskjutas.

Om skillnaden mellan din high- och lowpoly modell är för liten, kommer den resulterande heightmapen innehålla väldigt lite information / kontrast vilket också kan bidra till förskjutningsartifakter.

Om möjligt, så bör man baka ut maps med 16 eller 32 bitar per kanal. Extra bitdjup kan ge större frihet att redigera innehållet och hjälper till att undvika ränder.

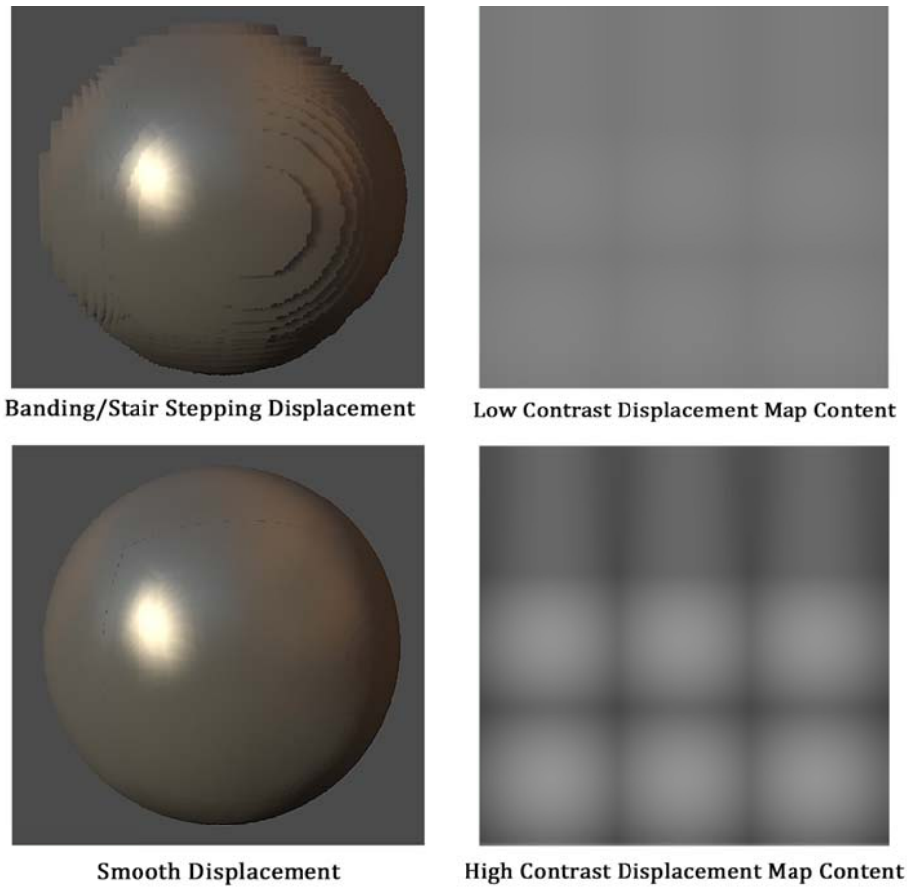


Bild 13. Banding

Seams

Förskjutningsartifakter vid UV-sömmar kan uppstå på grund av två andledningar:

Ofullständig filtrering av texels och otillräckligt bitdjup vid objektets UV-sömmar.

Det finns ett par saker man kan göra för att motverka dessa förskjutningsartefakter.

Minimera antalet UV-sömmar per geometri, dvs ha mer sammanhängande uv-grupper i geometrins UV-map och gömma eventuella sömmar i trånga områden på modellen där artifakterna inte syns.

Ofullständig filtrering av texels vid UV-sömmar kan motarbetas med större texturstorlek. En displacement map är sällan mindre än 1024x1024 då färre texlar kan leda till förlorad detaljrikedom vid sömkanter.

Editering och komprimering av texturer och dess bitdjup kan leda till texturartefakter som sedan kan synas i förskjutningen.
Ett trick kan vara att måla 50% grått (r128,g128,b128) med en mjuk pensel längs med alla uv-kanter på din heightmap.

Stretching

Stretching artefakter uppstår vid skarpa och extrema höjdskillnader i din displacement map.
Ett bra sätt att motverka detta på är att arbeta med ett liknande arbetsflöde som vid bakning av normal-data, att vid modelleringen av en högupplösta modell använda sig av bevlade kanter och lutande plan istället för hårda 90° kanter.

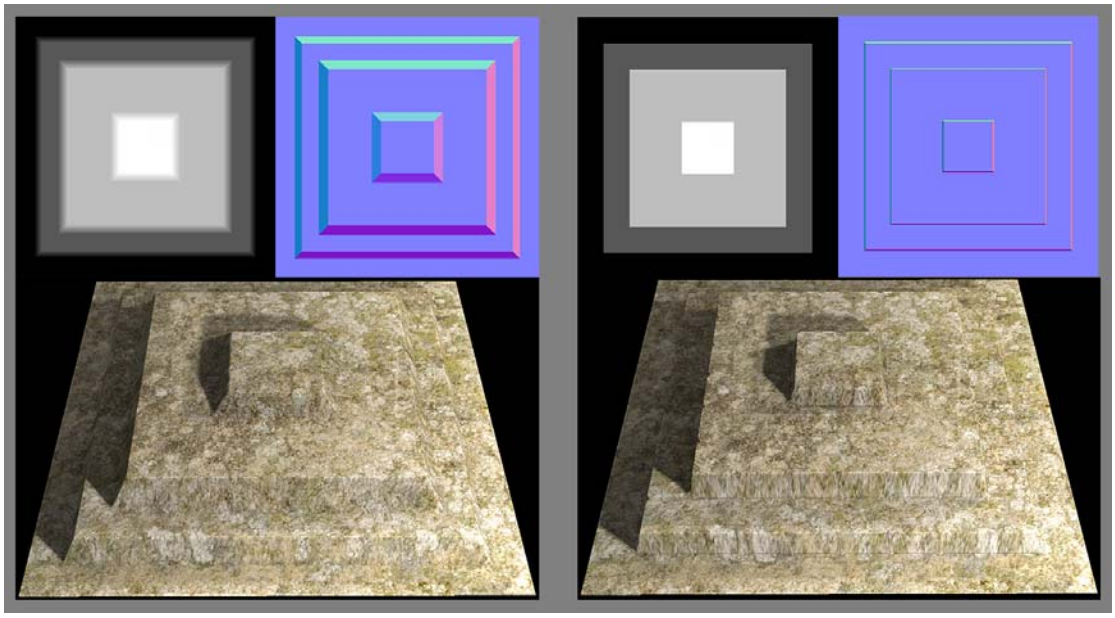


Bild 14. Mjukare former utan strecthing t.v.
Hårda kanter och streching artifakter t.h.

Du kan även editera dessa hårda kanter efter en bakning med hjälp av blurfilter i Photoshop.

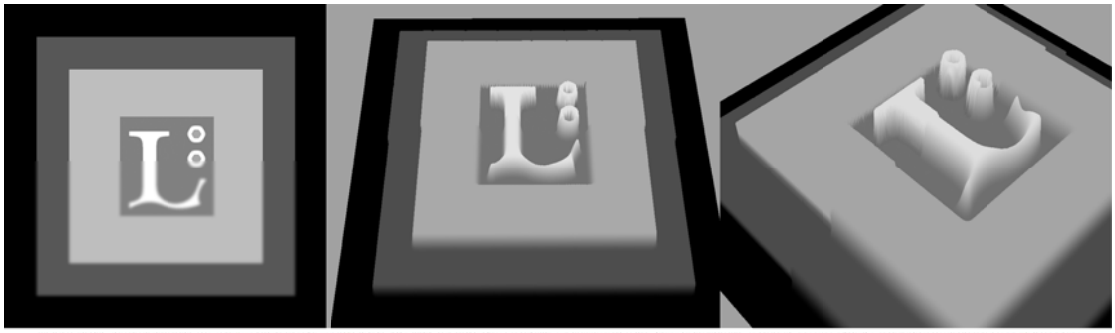


Bild 15. Displacement map i Nebula3. Blurfilter läggs över den nedre delen av texturen (t.v.)
Stretching artefakterna mjukas upp och ger en finare displacement förskjutning (mitten, t.h.)

Att matcha den lågupplösta modellen till den högupplösta versionen kan också bidra till färre stretching artefakter. Men om du matchar lowpoly geometrin alltför mycket så bidrar detta till en kontrastlös displacement map och förskjutningen blir allför subtil för att vara till någon nytta.

Att öka *tessellation levels* eller *tessellation factor,* dvs öka mängden uppdelningar av varje patch i tesselerings shadern, kan ge mindre stretching artefakter, men det kan även leda till onödiga tessleringsberäkningar och onödigt högupplöst geometri.

Det uppstår alltid någon sorts stretching vid användningen av displacement maps, men ha i åtanke att de flesta stretchingartifakter är så små att de knappt syns när det beskådas från planerat avstånd från kameran. Så lägg inte ner onödig tid på att åtgärda problem som inte kommer märkas förrän man zoomar in riktigt nära.

3.3 Tesselering och Displacement - Användningsområden

3.3.1 Landskap

Att tesselera terränggeometrier är det mest självklara användningsområdet inom tesselering[9][10] eftersom rendering av ett så stort objekt som ett landskap i realtid kräver ett bra LOD system. Att rendera Highpoly i direkt närhet till spelaren och lowpoly i horisonten utan synliga sömmar kan vara avgörande för att kunna ge spelaren en djupare spelupplevelse.

Tesselering och LOD av terränger är så pass viktigt att de flesta spelmotorer har inbyggda terrain editors med specialskrivna shaderlager och verktyg som kan ge grafiker större möjligheter att skapa stora och detaljrika terränger snabbt och enkelt.

För att utforska tesselering och displacement mapping av terränger så valde jag att gå igenom två olika arbetssätt för att öka detaljnivån på ett landskap:
heightmap/geometri som är skapad i en terrain editor.
heightmap/geometri som är skapad utan terrain editor.

3.3.1.1 Terrain Editors

De flesta moderna spelmotorer har en terrain editor (UDK, Cryengine, Unity m.fl.) där man kan skapa plangeometrier med adaptive tessellation i stor skala och använda verktyg för att modellera dessa direkt i editorn. De editorverktyg som finns är relativt simpla och lätta att använda. Man kan snabbt bygga upp ett enkelt landskap som sedan kan användas som grund för spelet.

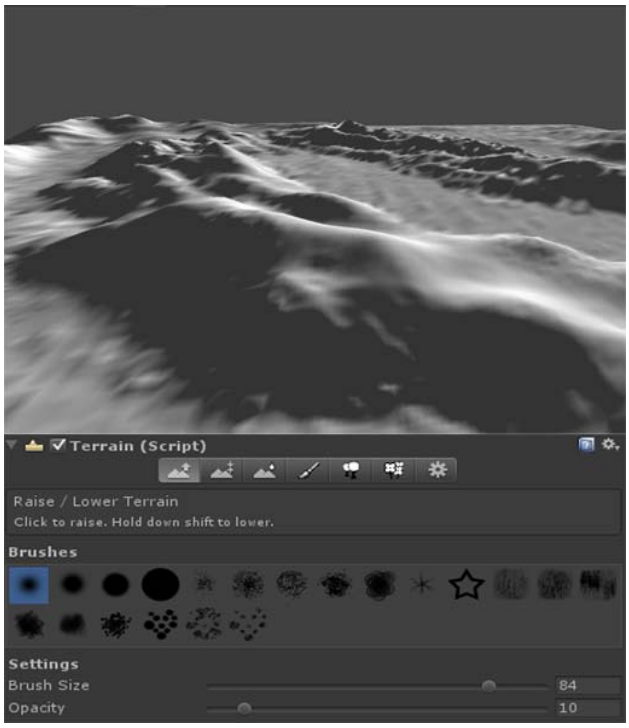


Bild 15: exempel på editorverktyg och plangeometri från Unity4.
Terrängshaders i moderna spelmotorer kan koppla samman en mängd matrial som i sin tur kan använda sig av ett flertal texturer (diffuse-, normal- och specular). Du kan fort få fram ett övertygande landskap med stor variation i både form och textur.

Detta beror helt på vad det är för spel man vill skapa. Ett tredjepersons-spel med en stor och öppen värld har kanske inte samma krav på detaljrikedom som ett firstperson spel har.

Men om man önskar att skapa mer realistiska terränger så måste man ta denna grundläggande terränggeometri vidare till nästa nivå.

Man kan exportera en height map från de flesta editors. I detta exempel Unity4 så finns alternativet under Terrain>Export Heightmap. där du kan exportera en 16-bit raw height map som standard.

Denna map används sedan för att skulptera om ett plan i Mudbox med hjälp av funktionen som man hittar UVs & Maps>Sculpt using map>New Operation.

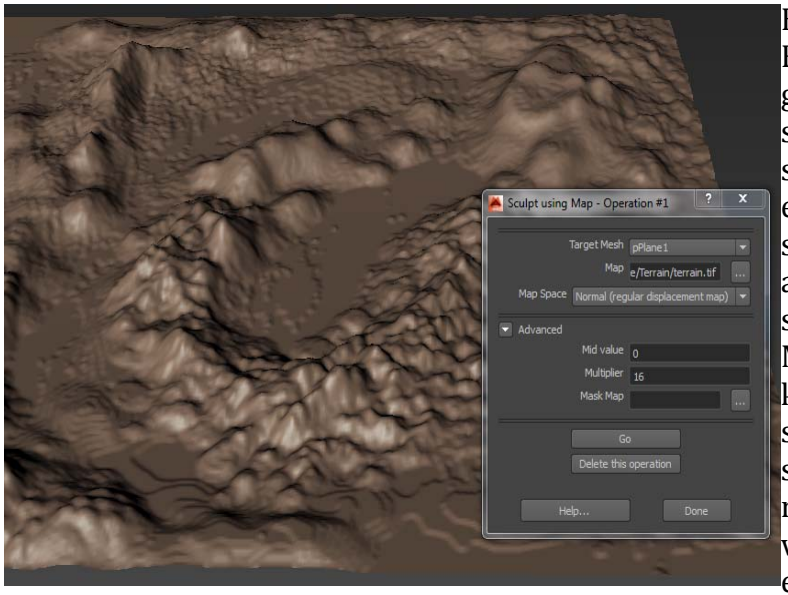


Bild 16:
Funktionen förskjuter geometrin och skapar ett skulpteringslager. Detta skulpteringslager kan nu editeras, subdiveras och skulpteras med de mer avancerade skulpteringsverktyg som Mudbox har att erbjuda. Här kan man använda sig av stenciler, masker och stämplar för att skapa mer realistiska detaljer som kan vara svårt att fånga med en editors begränsade verktyg.

Om skulptering av dessa detaljer inte kan göras för hand så finns det verktyg och program som är specialicerade för ta dig en bit på vägen.

WorldMachine[6] är ett exempel på ett sådant program.
Efter att man har skulpterat vidare på grundgeometrin så kan man baka ut en nya heightmap från Mudbox och ta in denna i ett program som World Machines som med hjälp av filternoder lägger på önskade effekter.

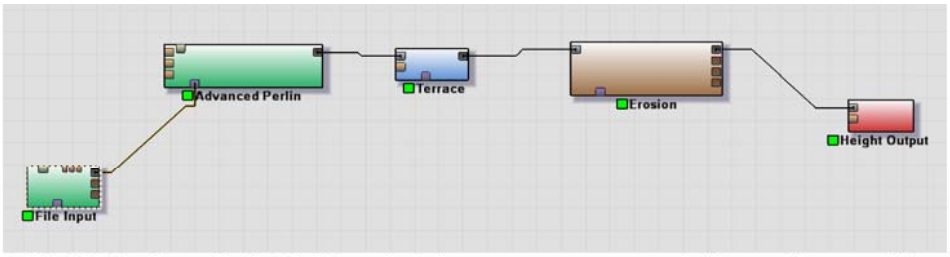


Bild 18. World Machine. Nodträd där en heightmap passerar en rad filternoder som alla adderar en önskad effekt på terrängen.

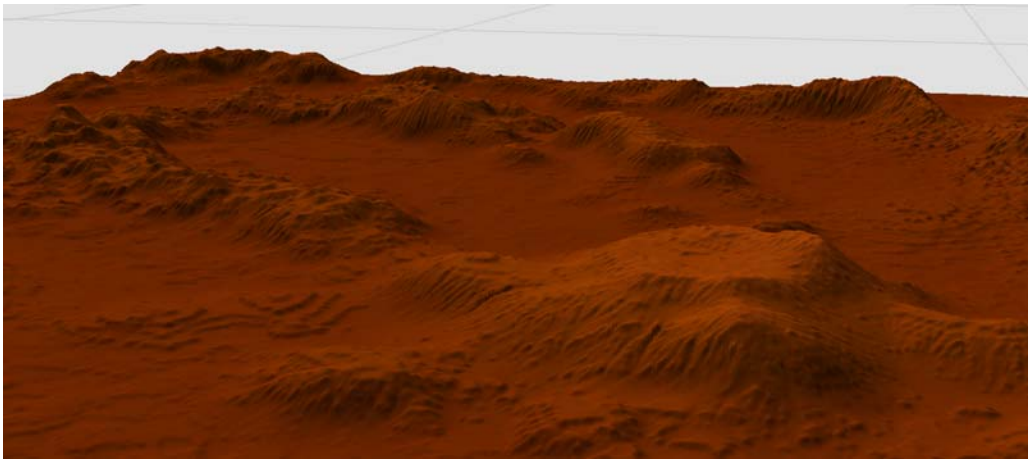


Bild 19. World Machine. Editor specialicerad för att applicera filter och snabbt och kontrollerat generera realistiska landskap.

När filter effekterna är adderade och önskat resultat är uppnått så exporterar man åter igen en heightmap i önskat format (för detta exempel: raw16-bit, till Unity4 editor)

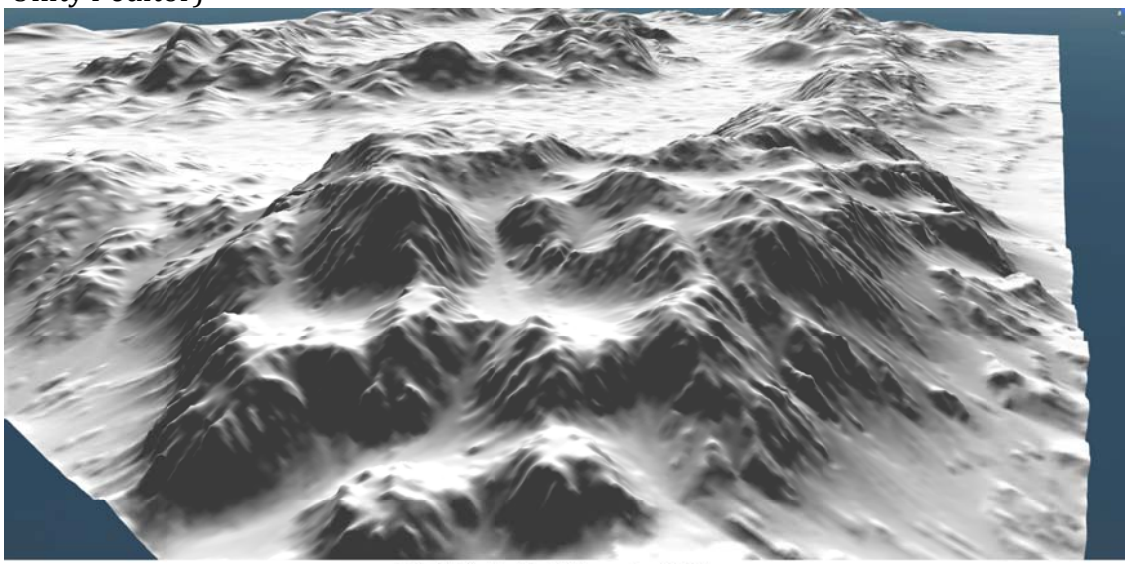


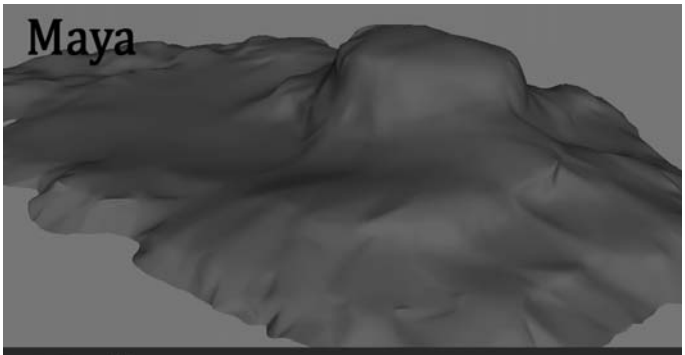
Bild 20. Unity4 Terrain Editor.

Terrängplanet förskjuts åter igen, men denna gång med en mer realistisk bergskedja med förfinade detaljer, erosion och en mer slumpmässigt genererad klippformation.

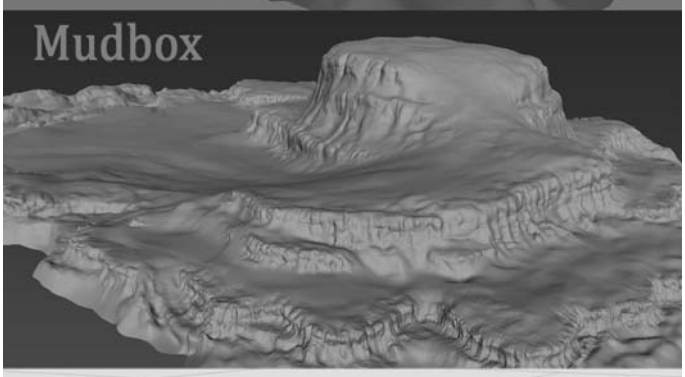
3.3.1.2 Terräng Geometrier

Detta exempel visar hur du kan arbeta med en spelmotor som saknar en terrain editor.

För att skapa en terräng i en editorlös spelmotor såsom Nebula3 så måste man nyttja sig av separata geometrier och ha tillgång till en tessellation shader med en fungerande LOD kontroll. För texturering av ett objekt i denna skala så måste shadern även ha stöd för användingen av en vertex painter eller liknande.



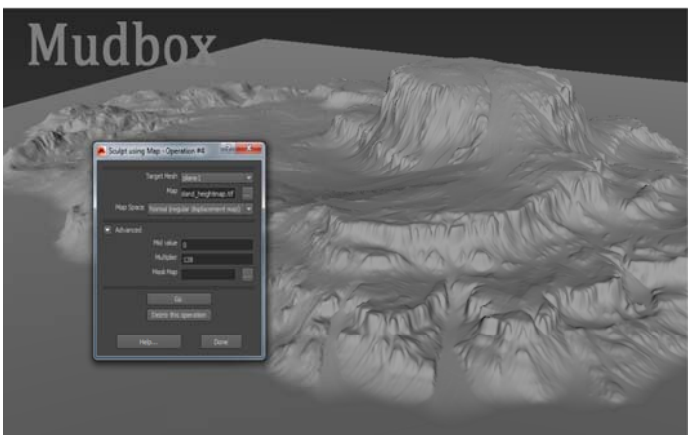
Skulpteringsarbetet är ungefär detsamma som tidigare, fast denna gång så börjar arbetet i Maya. Precis som i arbete med en terrain editor så skulpterar man först fram de större formerna i önskad skala på ett grundplan, denna gång med Mayas Sculpt Geometry Tool.



Detta grundplan skickas sedan vidare till Mudbox där mer avancerad detalj kan adderas. Redan här kan man arbeta på hur landskapets komposition och struktur bör se ut. Vilka områden som ska vara hårda/mjuka och branta/platta.

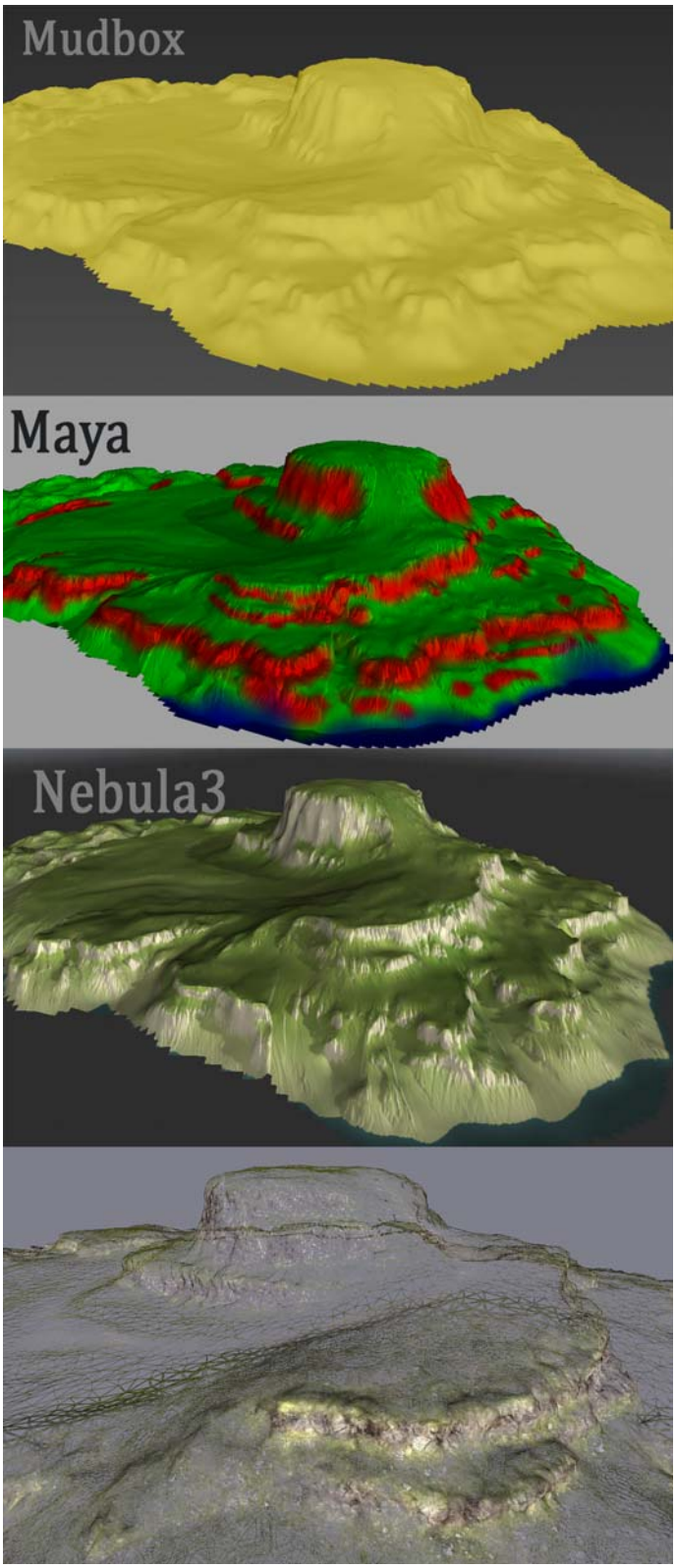
World Builder-noder adderar erosion och tar bort eventuella ”pänseldrag” från Mudbox-detaljerna.
Resultatet exporteras i form av en heightmap med passande bitformat. Vanligtvis ska denna heightmap tas in i en terrain editor i utvald spelmotor, men denna gång så skickas den tillbaka till Mudbox för att användas i Sculpt with Maps-funktionen.

Du skapar ett nytt plan i Mudbox och subdiverar denna tills du har en högupplöst geometri som är redo för displacement skulptering.



När en högupplöst version av landskapet finns på nivå 7 så kan man stega nedåt till nivå 5 (102 400 polygoner) för att få ett mer överkomligt polygonantal.

Använd sedan Mudbox face selection tool för att välja ut den del av planet som du har användning av och duplicera Mesh>Duplicate Selected Faces till en egen separat geometri.



Baka sedan ut displacment och normalmap mellan nivå 7 och 5 för att kunna omvandla den lågupplösta versionen av landskapet till en högupplöst version i spelmotorn.
Denna terräng är nu redo för displacement förskjutning och texturering.

Nebula3 kan läsa in vertex färger och applicera texturer på geometrier därefter.
Detta ger oss möjligheten att lägga in en textur per färgkanal och få mjuka övergångar mellan dessa.

Färgkanaler och deras texturmotsvarigheter:
Grön:
grästextur + gräs normalmap
Röd:
klippor + klipp normalmap
Blå:
sand + sand normalmap
Alpha:
terrain displacement + terrain normalmap

Resultatet blir en terräng på ca 300 kvadrat enheter eller *units*som kan textureras med ett material.

Tesselering av terränggeometri i nebula3 med lod inställd på ca 50 units. (observera gränsen mellan låg och hög tesselering.)

3.3.2 Karaktärer

Testerna i detta kapitel utfördes på två olika karaktärer. En karaktär med realistisk design som består av flertalet geometrier och en med stiliserad design som består av en geometri. Den realistiska karaktären är skapad av Paul Tosca[12] medan den stiliserade karaktären är vår egen *Kapten Gscept*.
Tester på dessa karaktärer genomfördes i Maya 2014, viewport 2.0 och i Nebula3.

Tesselering av karaktärer är ofta fördelaktigt då de vanligtvis befinner sig nära spelarens kamera. Det är i dessa närbildssituationer som detaljrikedomen ibland inte räcker till. Kantiga silhuetter och platta detaljer kan dock motverkas med smooth tessellation såsom PN-triangles vilket är en av de vanligaste teknikerna bland som används för detta ändamål i marknadens spelmotorer[7][8].

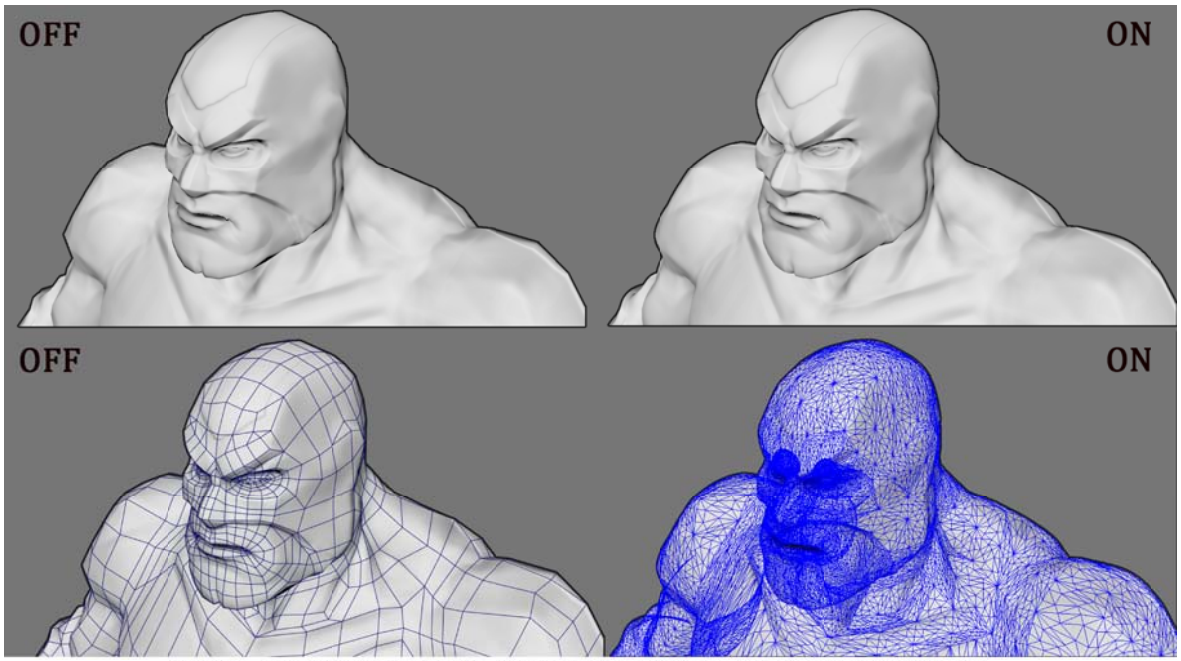


Bild 25. Tesselering med PN-triangles kan förbättra karaktärens silhuette.

Bild 26. Tesslering med PN-triangles förbättrar former och ökar realism.

Vid concept steget av en karaktär så är det till ens fördel om man identifierar områden som kan komma att kräva mer detalj än andra. Det vill säga, områden på en karaktär som kommer synas mest i bild, till exempel huvud och överkropp. Detta tankesätt bör även tillämpas gällande tesselering då vissa kroppsdelar kräver olika mängd tesselering för ett lyckat resultat. (bild 27.)

Bild 27. Karaktär med gradient som markerar områden som kan tänkas kräva en högre detaljnivå med grönt.

Som det nämns i tidigare kapitel *3.2.2 Lowpoly Topologi* så är en jämn topologi avgörande för ett bra förskjutningsresultat från din displacement map. Dock så är en jämn topologi redan en viktigt del i skapandet av realtids karaktärer eftersom animationer kräver strategiskt placerade edgeloops för att undvika dåliga deformationer. Detta betyder att arbetsflödet som redan existerar för karaktärsskapande är relativt anpassat för geometriförskjutning.

Bild 28 t.h.
En snygg silhuett kan uppnås med hjälp av en relativt låg tesseleringsfaktor.

För att uppnå en snygg silhuett på en karaktär som denna krävdes inte en hög tesselerings faktor då optimeringen har bevarat silhuetten mycket väl. (Bild 28) Dock så finns det en del hard surface geometrier på denna karaktär som vid tesselering med smooth tessellation får, precis som resten av karaktären, en avrundad och mjukare form. Om detta inte är det önskade resultatet så är man tvungen att motverka detta med hjälp av flat tessellation.

Flat tessellation ger inte en förfinad silhuett på samma sätt som smooth tessellation då den endast adderar polygoner innanför de som redan existerar på den lågupplösta modellen. Det är dock här displacement mapping kan appliceras. Istället för att endast använda sig av smooth tessellation för att förfina silhuetten så kan förskjutningen från en displacement map skapa en ny, förfinad silhuett (bild 30).

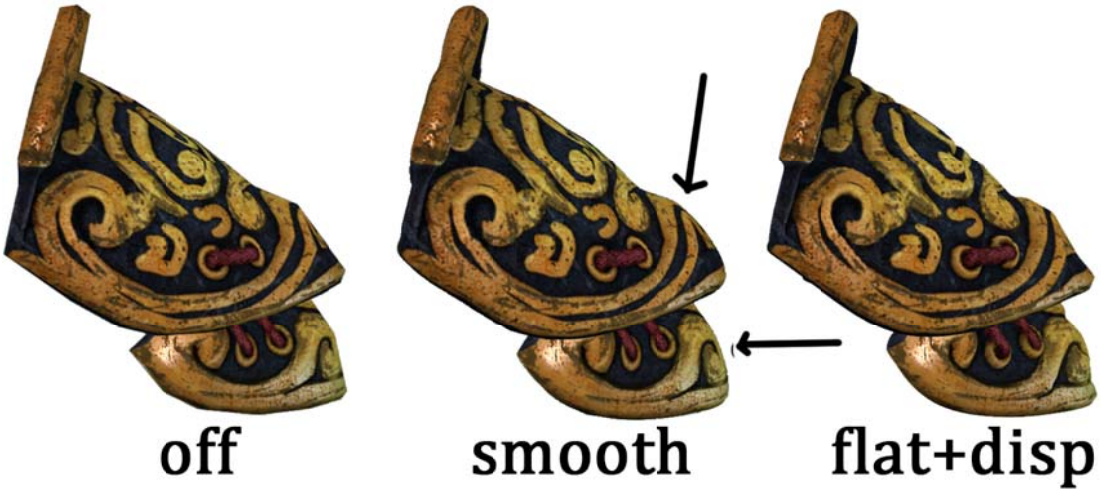


Bild 30. Smooth tessellation skapar avrundade kanter och silhuetter. Ett mer korrekt resultat uppnås med flat tessellation och displacement mapping.

Att applicera smooth tessellation på vissa geometrier kan även få dem att expandera mer okontrollerat vid förskjutning med en displacement map. Att applicera flat tessellation håller istället tillbaka dessa utsvävningar som annars kan leda till geometrikrockar och okorrekta detaljer.

Bild 31. Pilar pekar på områden där smooth tessellation displacement kolliderar med närliggande geometri och får en oönskad förskjutning.

För att uppnå smooth tessellation på de organiska delarna av en karaktär och flat tessellation på delar som är hard surface kan kräva flertalet shaders, vilket kräver fler drawcalls, så en kompromiss kan oftast vara den mest optimerade lösningen. Att applicera flat tessellation över hela karaktären med en resonabel tesselation factor och sedan förskjuta geometrin till ett önskat resultat med en låg displacement height inställning gav bättre resultat än smooth tessellation.

När man skapar en karaktär till ett spel så vill ofta att den ska ha ett så brett användningsområde som möjligt. Det är vanligt att man vill ge sin karaktär variation i form av nya kläder (*skins),* addera slitage, uppgraderingar, till och med förändra dess proportioner på olika sätt. Med displacement mapping kan man förskjuta sin geometri till den grad att man har möjligheten att addera dessa variationer i textur istället för att ersätta karaktären med en helt ny modell. Man kan utgå från en grundversion av karaktären och sedan använda denna som mall då man vill addera nya detaljer.

Man kan ta in denna grundversion eller *basemesh* till ett skulpterings program såsom Mudbox eller Zbrush där med hjälp av olika verktyg en grafiker kan lägga till eller ta bort form från karaktären.

Bild 32. Tesselering och displacement. Exempel på hur man kan ge en superhjälte ett mer aggressivt utseende med hjälp av ett texturbyte.

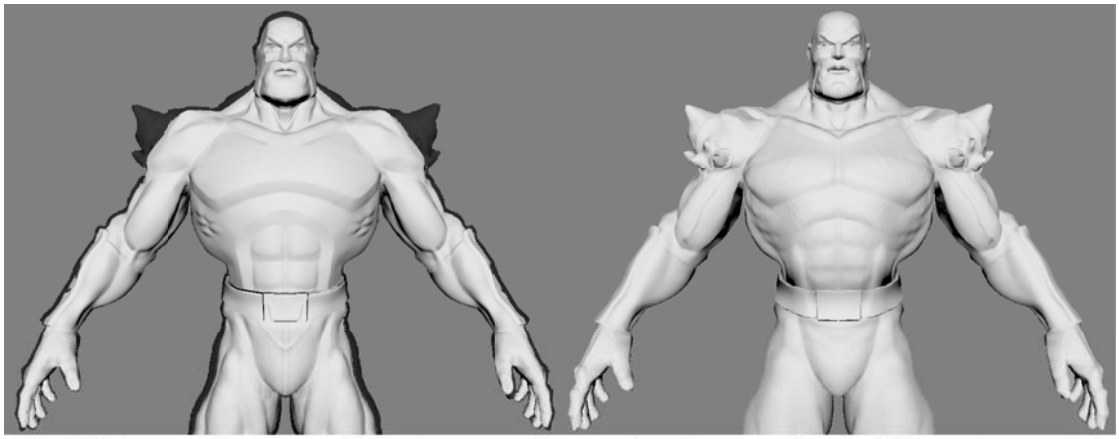


Bild 33. Exempel hur du kan förändra proportioner och addera nya detaljer både med geometri och textur.

3.3.2.1 Animationer

Att kunna mjuka upp silhuetten och detaljer på en karaktär med hjälp av smooth tessellation påverkar inte animationspiplinen på något större sätt, men att addera en displacement map på det kan få oväntade resultat.

Om animationerna är anpassade för den lågupplösta versionen av karaktären så medför appliceringen av en displacement map alltid risken att utskjutande detaljer kolliderar med annan geometri på karaktären. Detta leder till att det antingen kommer krävas mer animationsarbete eller att modelleraren måste visa hänsyn till karaktärens alla olika rörelser vi skapandet av nya detaljer.

Bild 34. Begränsad rörlighet för armarna på grund av axlarnas nya displacement geometri.

3.3.2.2 Previsualization

Att arbeta på en karaktär med displacement maps så måste en grafiker som sagt visa hänsyn till animationer och eventuell deformation av geometri. Eftersom en animatör oftast arbetar med en lågupplöst version av karaktären så är det svårt för honom/henne att se den displacement som senare kommer adderas inne i spelmotorn.

Autodesk Maya, som ofta används i en animationspipeline för karaktärer, är ett av de program som har introducerat ett stöd för DirectX 11 och OpenGl 4.x shaders i sina viewports.

Dessa shaders har samma inställningar som går att finna i shaders inuti de flesta spelmotorer som finns på marknaden, vilket ger grafiker möjligheten att upptäcka eventuella problem tidigt samt få en korrekt previsualisering.

3.3.3 Props

När man ville ha kontrol över geometri renderingen under tidigare spelgenerationer så blev man tvungen att ha flera versioner av samma modell med olika detaljnivåer och sedan byta mellan dessa beroende på avståndet till kameran. Detta kunde leda till en *poppande* effekt då ett objekt plötsligt fick mycket mer detaljrikedom och mjukare form. Tesselering löser det *poppande* problemet ganska effektivt men optimeringsfrågan kvarstår, vilka objekt bör man använda tesselering på?

Antalet props som behövs för att få en spelvärld att kännas trovärdig kan variera beroende på vilken typ av spel det är och vilken nivå av realism spelutvecklaren siktar på. Trots tekniska framsteg så kan man än så länge inte tesselera alla spelets geometrier utan man måste reservera användningen av tesselering till objekt där det gör mest nytta. Så det första planeringssteget är identifiera vart man är berädd på att offra detaljnivå och avrundade silhuetter för att kunna placera på ett fåtal utvalda.

Att tesselera och applicera displacement på alla dessa objekt är ofta onödigt då billigare tekniker såsom bump och Normalmapping klarar av att ge de flesta objekt en realistisk matrialkänsla.

Dörr exempel: Ytor som är inramade av annan geometri behöver oftast inte använda sig av displacement mapping då de inte har någon silhuett eller detalj som behöver förbättras på samma sätt som själva ramen.

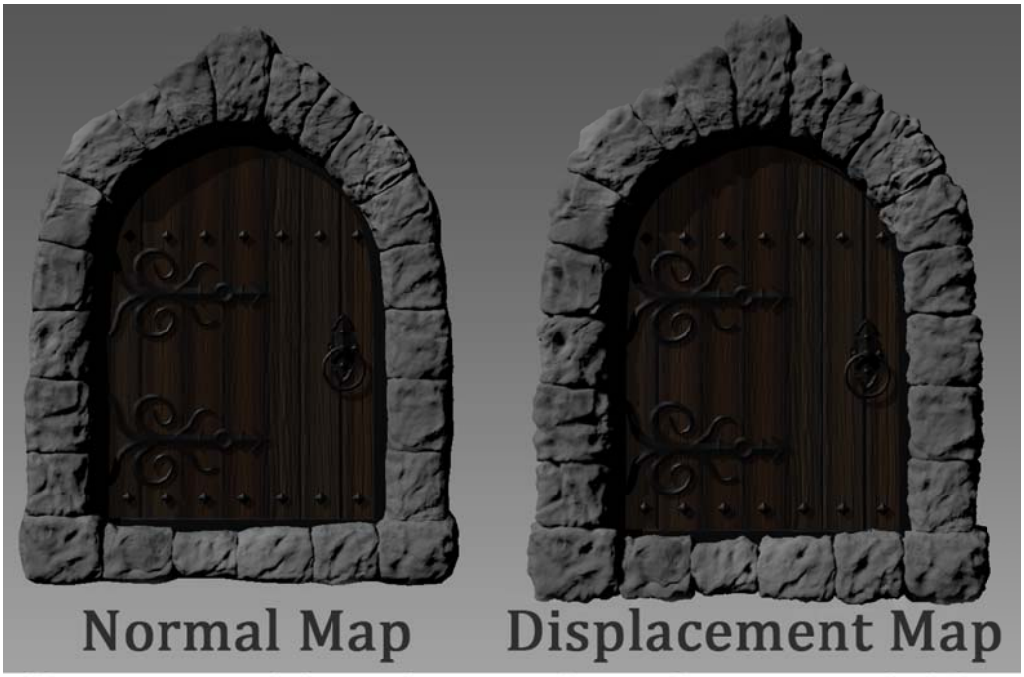


Bild 35. Dörrgeometrin behöver endast en normal map medan stenarna runt den behöver förbättrad silhuett och detaljnivå då den kan betraktas från flera vinklar.

Att dela upp denna modell till olika shaders kan också vara bra då det blir onödigt tungt att tesseleringa dörrens geometri eftersom den inte behöver använda sig av displacement mapping för att få ett bra resultat.

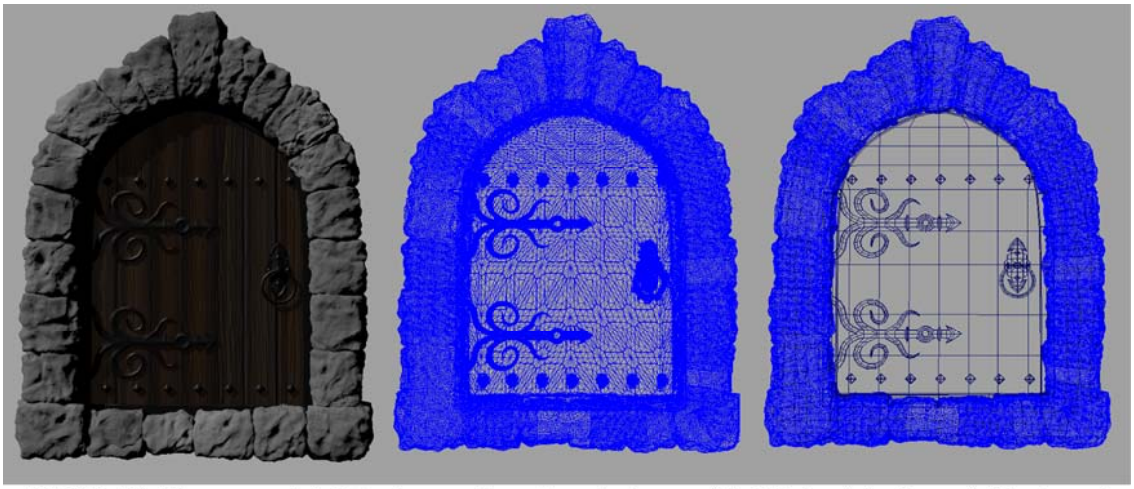


Bild 36. Då dörr geometrin inte drog nytta av tesseleringen så behövdes inte denna inkluderas i tesselerings shadern med gav fortfarande ett bra resultat.

Att tesselera och använda displacement maps på miljöobjekt som t.ex. klippor kan vara lönsamt om du är säker på att dessa kommer vara nära kameran och om du vill ha en mer intressant silhuett (bild 37).

Bild 37. Klippblock. Mer intressant silhuett och förbättrad ytkänsla.

Detta är ett bra exempel på när Displacement och Normal Maps kompletterar varandra. Normalmappen gör ett bra jobb med att skapa en detaljrik och hård materialkänsla på den lågupplösta klippan till vänster (bild 32), men den har dock en alldeles för ointressant silhuett i kontrast med det detaljrika texturen. Det är inte förrän man lyckats förskjuta geometrin som på klippan längst till höger (bild 32) som den känns mer trovärdig. Att återskapa den högra klippan med liknande lågupplöst geometri skulle kräva många fler polygoner än vad den har nu (1100 trianglar), vilket gör tesselerad displacement fördelaktigt för det här ändamålet.

Men vid användningen av klippgeometrier i spel så används de sällan ensamma - de slås ofta ihop med andra liknande geometrier för att bilda en ny, större form som har en egen silhuett. Då måste man undersöka om helheten och silhuetten av den nya formen håller måttet och se ifall de geometrier som ”ramas in” inuti den nya formen behöver tesseleras alls (se bild 33).

Bild 38. Flera klippgeometrier skapar en större helhet och ramar in vissa stenar.

Bild 39. Exempel från Battlefield 3 - Frotstbite3 engine där flertalet klippgeometrier bildar en större form.

Det kan vara onödigt att tesselera vissa stenar om de ingår i en större formation tillsammans med flertalet andra stenar. (bild 39) Det är oftast inte förrän man har placerat ut geometrierna i ett spel som man kan avgöra vilka geometrier som bör tesseleras för att få ett så optimerat resultat som möjligt.

3.4 Metodkritik

Jag har inte haft möjlighet att mäta hur de olika tesselerade geometrierna har påvärkat prestandan i varken Maya Viewport 2.0, Nebula3 eller Unity på ett exakt sätt och kan därmed inte peka ut vilken mängd av tesselering som är bättre/sämre för de olika geometrierna. De slutsatser som tagits i kapitel 4 baseras på hur jag uppfattat programmens hantering av de olika geometrierna vid testtillfällena.

4. Resultat

I denna rapport har jag gått igenom flertalet sätt att skapa och hantera displacement maps på tesselerade geometrier för olika användningsområden och kunnat ge grafiker en grundläggande syn på hur de bör planera och utföra sitt arbete. Jag har med enkla experiment hittat för och nackdelar med användningen av displacement mapping på tesselerade geometrier. Jag har påvisat skillnader i resultaten mellan normal mappade och displacement mappade geometrier samt kombinationen av de två.

4.1 Displacement Maps.

Jag utförde ett flertal framgångsrika bakningar på enkla geometrier och fick ut enkla displacement maps och fick tack vare mina resultat en större förståelse för de problem som kan uppstå.
Du bör alltid baka ut en så detaljerade displacement map som möjligt och sedan avgöra vilket format och filstorlek som är nödvändig för den motor du ska använda för att få en så bra förskjutning som möjligt. Bakning av maps med stort bitdjup och detaljrikedom ger dig också möjligheten att undvika eventuella artefakter.

4.2 Tesselering av Terräng.

Det mest självklara användningsområdet för tesselering då landskapsgeometrier är stora och deras detaljrikedom begränsas oftast av detta.
Att applicera tesselering på en terräng bör göras på ett så optimerat sätt som möjligt. Detta uppnås bäst med att skulptera en högupplöst terräng och sedan tesselera och förskjuta fram dessa detaljer med hjälp av den dynamiska LOD som en tesselerings shader kan erbjuda

Efter tester med olika motorer och deras editors så har jag kommit fram till att användningen av tesselering, displacement och normal maps gör det mycket lättare för en grafiker att skulptera fram detaljrika och övertygande geometrier.

4.3 Tesselering av Karaktärer.

Ett grundligt förarbete och bra optimering av din karaktär kan ge dig möjligheten till en optimerad tesseleringsfaktor. Ju bättre man lyckas bevara en silhuett destom mindre krävande tesselering krävs för att förfina formerna. För att bevara en bra silhuett så krävs fler polygoner, att addera dessa polygoner görs bäst på ett grafikkort istället för att läggas på din lågupplösta modell.

Du kan förskjuta fram nya detaljer eller öka realismen på de som redan existerar med hjälp av displacement mappingen. Men att göra detta kräver dock att du visar hänsyn för evenentuella animationer din karaktär har.

Att få en previsualisation av de nya detaljerna bör göras så tidigt i pipelinen som möjligt istället för att backtracka fram och tillbaka mellan modellerings program och spelmotor.

4.4 Tesslering av Props.

Trots tekniska framsteg så kan man än så länge inte tesselera alla spelets geometrier utan man måste noggrant välja ut vilka objekt som tjänar mest på användandet av denna teknik då optimering fortfarande är väldigt viktigt.

Inramade geometrier som saknar en silhuett behöver oftast inte tesseleras och detta gäller även om din geometri ingår i en större form med en större siluett.

Att addera detaljrikedom innanför silhuetten kan anses vara onödigt då normal maps klarar av detta jobb väldigt bra och kräver inte lika mycket datorkraft för att generara.

Avvägningen mellan vilka objekt som ska och inte ska tesseleras är en process som troligtvis kommer pågå under hela spelprojektet då slutresultaten kan jämföras på ett bättre sätt.

5 Diskussion

Att arbeta med Displacement maps inom realtidsgrafik är en aning mer komplicerat än arbete med andra bumpmap-tekniker . Jag ville därför identifiera viktiga riktlinjer som man ska kan följa för att undvika eventuella problem som kan uppstå vid bakningen och geometriförskjutning i en spelmotor. Om, mot förmodan, ett problem uppstår så hittade jag även ett par enkla tips för att åtgärda dem.

Jag hoppas att resultatet i denna rapport ska kunna underlätta och tydliggöra för andra hur de bör gå till väga för att själva kunna skapa optimerade men samtidigt realistiska spelmodeller.

Det finns mängder av arbetsexempel som jag inte kunde täcka i denna rapport, t.ex Vector Displacement som till skillnad mot vanlig displacement sparar ut en XYZ-vektor som möjliggör geometriförskjutning i fler riktningar än en. Men jag tror jag lyckats få fram en grundläggande bild av vad framtiden inom modellering, texturering och optimering har att erbjuda.

De tester jag har utfört må vara enkla men med kunskapen man kan lära sig från dem bör vara tillräckligt för att skapa sig en egen uppfattning om hur användbart tesselering och displacement mapping kan vara inom de olika grafiska arbetsområderna.

Att tesslerea objekt är, trots tekniska framsteg, krävande och bör endast appliceras på objekt som kommer synas på nära håll och endast öka *tessellation levels* och *factors* till den grad som anses vara tillräcklig och att inte överdriva tessleringen för att kompensera brister i modellering, bakning eller texturkvalitét.

Framtiden för tesselering och displacement mapping inom realtidsgrafik är ljus då det endast är hårdvaran som sätter gränserna, och hårdvarornas kraft ökar exponentiellt. Displacement mapping inom realtidsgrafiken är ett stort steg mot ökad realism på alla plan men det kommer nog inte ersätta normal mapping helt och hållet då en viss form av optimering alltid kommer förekomma.

En grafiker måste hålla sig uppdaterad inom nya tekniker och arbetsflöden då dessa utvecklas konstant från år till år. Detta är bara början av den ljusa framtiden som tesselering och displacement mapping går till mötes, och jag ser fram emot att få göra fler tester i fler program och spelmotorer i framtiden.

6 Slutsatser

Tesselering och Displacement Mapping har öppnat upp nya möjligheter för hur moderna spel kommer att tillverkas och hur slutresultatet kommer att se ut.
Under mitt arbete så har jag lärt mig tillräckligt för att kunna fördjupa mig inom mer avancerade displacement tekniker såsom Vector Displacement, Displacement Blending och Displacement Animations. Områden som också är relativt nya inom realtidsgrafiken. Att utforska nya tekniker, att tänja på gränser och att utmana sig själv och sina geometrier är det som för industrin framåt.

Tesselering och displacement maps är bara ett exempel på industrins olika tekniker för att ge spel en mer realistisk rendering och det finns många som är intresserade av deras utveckling. Snabbare datorer, kraftigare grafikkort och större RAM minne utvecklas konstant och tesselering och displacement kommer följa med i denna utveckling som ett återkommande alternativ för spelutvecklare världen över som vill få ut högsta möjliga bildkvaliteten.

Det är många programmerare och grafiker där ute som söker efter den perfekta balansen mellan prestanda och estetisk tilltalande geometrier och det känns som att det bara är hårdvaran som håller oss tillbaka och att vi endast har skrapat på ytan av de möjligheter som framtidens hårdvara kommer kunna låsa upp.

7 Referenser

[1] Curved PN Triangles. Vlachos A., Peters J., Boyd C., and Mitchell J. 2001. Symposium on Interactive 3D Graphics 2001
http://alex.vlachos.com/graphics/CurvedPNTriangles.pdf

[2] Phong Tessellation. [Boubekeur and Alexa 2008] – Subdivision Shading.
http://perso.telecom-paristech.fr/~boubek/papers/SuSh/sush.pdf

[3] Barycentric Interpolation.
http://classes.soe.ucsc.edu/cmps160/Fall10/resources/barycentricInterpolation.pdf

[4] View-Dependent Adaptivity Metric For Real Time Mesh Tessellation. Tamy Boubekeur
http://perso.telecom-paristech.fr/~boubek/papers/VDTess/VDTess.pdf

[5] Heightmap http://en.wikipedia.org/wiki/Heightmap
Displacement Map http://en.wikipedia.org/wiki/Displacement_mapping

[6] World Machine – 3D Terrain Editor.
http://www.world-machine.com/

[7] UDK Tessellation
http://udn.epicgames.com/Three/TessellationDX11.html

[8] CryEngine Tessellation
http://freesdk.crydev.net/display/SDKDOC2/Tessellation+and+Displacement

[9] Terrain Tessellation, Iain Cantlay.
https://developer.nvidia.com/sites/default/files/akamai/gamedev/files/sdk/11/TerrainTessellation_WhitePaper.pdf

[10] OpenGL 4.0 Tessellation For Professional Applications,
GPU Technology Conference - San Jose, CA 2010
http://www.nvidia.com/content/GTC-2010/pdfs/2227_GTC2010.pdf

[11] Nvidia DirectX 11 Tessellatoin.
http://www.nvidia.com/object/tessellation.html

[12] Paul Tosca Character.
Model: http://www.paultosca.com/tt1/Nyra_ma_fbx.rar
Texures: http://www.paultosca.com/tt1/Nyra_TGAs.rar
http://www.paultosca.com/tt1/Nyra_head_d_PSD.rar
video: http://www.youtube.com/watch?v=L5fOwSmSaW8