

TSBK03

Teknik för avancerade datorspel

Badhairday

Carl Dehlin, carde650@student.liu.se
John Styngberg, johst529@student.liu.se

30 december 2017

Sammanfattning

Detta är en rapport som detaljerar tekniker som kan användas för att generera, animera och belysa hår på 3D-modeller. För generering av geometri används tessellering för fler vertexar och geometry-shaders för att generera linsegement för hårstrån. Animering sker genom att i varje tidssteg att generera hår med påverkan av tidsvarierande krafter. Belysning av håret beräknas enligt Kajiya-Kay-modellen och skuggor beräknas med en voxeliseringssmetod.

Innehåll

1	Introduktion	3
2	Metod	4
2.1	Geometrigenerering	4
2.1.1	Tessellering	4
2.1.2	Bezierytor	4
2.1.3	Hårgenerering	7
2.1.4	Hårtyper	8
2.1.5	Animation	10
2.2	Ljusmodell	10
2.2.1	Diffus komponent	10
2.2.2	Spekulär komponent	11
2.3	Skuggmodell	12
2.3.1	Voxelisering	12
2.3.2	Skuggmappning	14
3	Resultat	16
3.1	Tessellering	16

3.2	Hårtyper	16
3.3	Shading	16
4	Diskussion	22
4.1	Tessellering	22
4.2	Hårgenerering	22
4.3	Animation	22
4.4	Skuggmodell	22

1 Introduktion

Hair is the first thing. And teeth
is the second. Hair and teeth. A
man got those two things he's got
it all.

James Brown, 1986

Att rita hår på 3D-modeller ger dem liv. Modeller av människor, djur och andra levande kreaturer har mycket att hämta i hårgenerering för att komma närmare målet att vara realistisk. Detta projekt fokuserar på hår, men att generera geometri på ytan av en modell kan användas för både fjädrar och fjäll för att förverkliga fåglar och reptiler.



Figur 1: Konceptuell bild av en prototypisk modell med olika sorters hår.

2 Metod

Under denna rubrik beskrivs metoder som användes för projektet. Först beskrivs hur generering av geometri har gjorts och sedan hur denna ska belysas och skuggas.

2.1 Geometrigenerering

I denna sektion kommer genereringen av vertexar på 3D-modellen diskuteras, samt hur linjesegment skapas utifrån informationen som är tillgänglig. Efter detta diskuteras hårtyper och animering.

2.1.1 Tessellerering

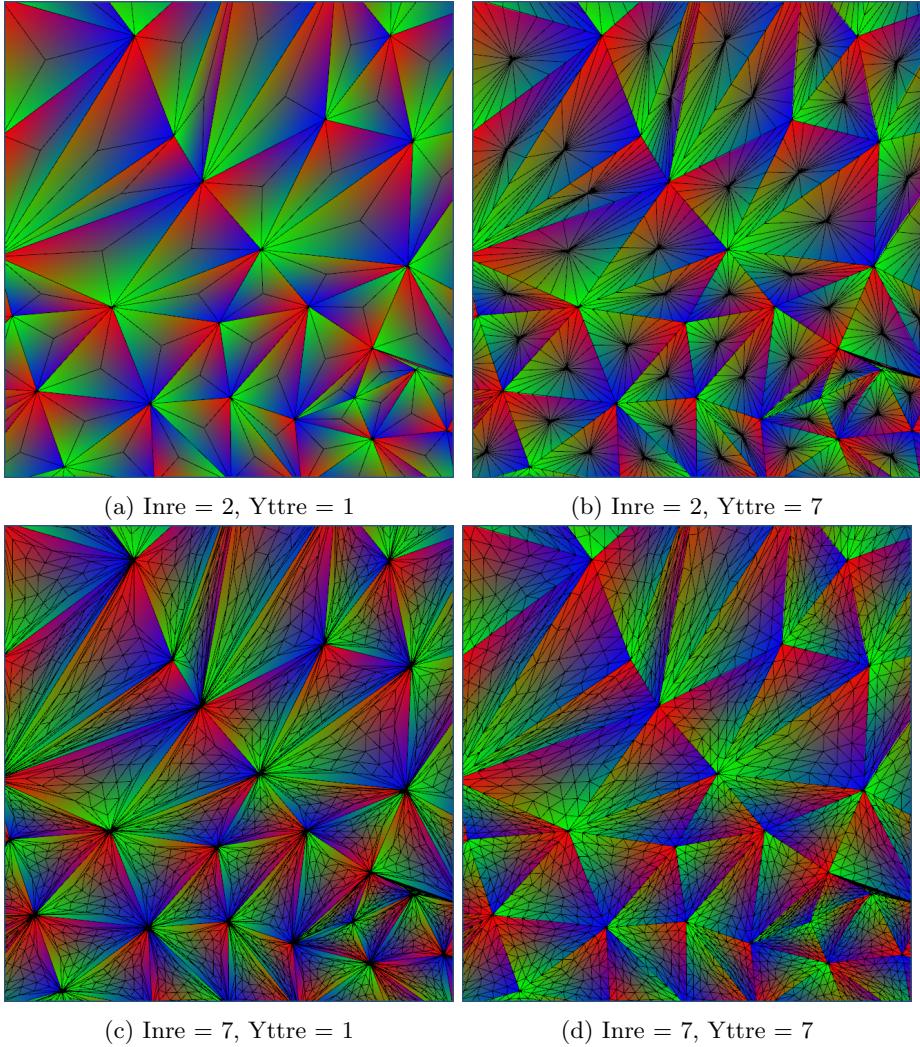
Innan hårstråen börjar skapas utanpå modellen finns ett tesselleringssteg för att öka antalet trianglar som finns att arbeta med. För varje triangel skapas ett strå, vilket innebär att om en modell har lågt antal polygoner så blir modellen gleshårig.

Tessellerering görs utefter två parametrar för varje kant, den inre och den yttre tesselleringsnivån. Den sistnämnda av dessa är enkel att tyda, den anger hur många segment som kanten ska delas upp i. Den förstnämnda är lite mer ointuitiv och svår att beskriva. Den intresserade hänvisas till [1] för utförlig beskrivning om hur tesselleringen sker.

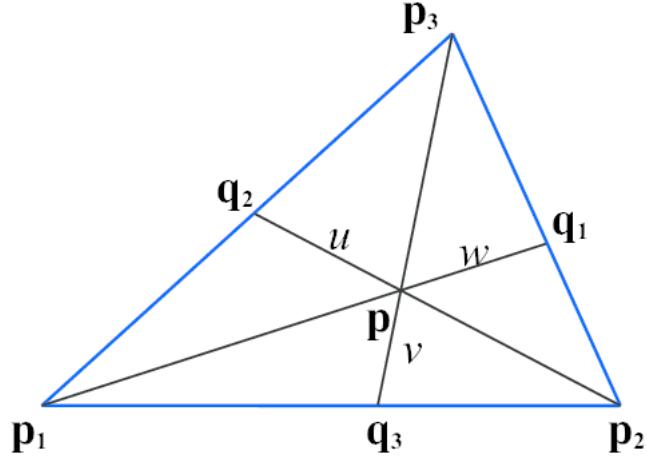
Efter att både yttre och inre trianglar har skapats, länkas triangelnivåer ihop. Olika inre och yttre nivåer kan ses i figur 2. I figur 2b tillsammans med 2d syns det att de inre trianglarna ser likadana ut oavsett vilken yttre nivå som sätts. Men med en yttre nivå som är lika stor den inre så syns ”uträkningarna” som skapade de inre nivåerna.

2.1.2 Bezierytor

Bezierinterpolation är en teknik som används som ett alternativ mot att representera en kurvad yta som resultatet av många platta polygoner. Istället används behandlas de inkommande polygonerna som en kurvad yta och använder hörnpunkterna och deras normaler för att interpolera fram ytan.



Figur 2: Visualisering av olika tesselleringsnivåer. I dessa exemplen används samma tessellering för alla kanter.



Figur 3: Domänen för en bezier-triangel. Bildkälla [2]

Positioner på ytan interpoleras från kontrollpunkter¹ på ytan. Dessa kontrollpunkter kan ses i figur 4a. För interpoleringen så används barycentriska koordinater, som visas i figur 3. w är för enkelhetens skull definierad som $w = 1 - u - v$. Formeln för den kubiska bezier-ytan är:

$$\begin{aligned} \mathbf{p}(u, v) = & u^3 \mathbf{p}_{300} + v^3 \mathbf{p}_{030} + w^3 \mathbf{p}_{003} + 3u^2v\mathbf{p}_{210} + 3uv^2\mathbf{p}_{120} + \\ & + 3u^2w\mathbf{p}_{201} + 3v^2w\mathbf{p}_{021} + 3vw^2\mathbf{p}_{012} + 3uw^2\mathbf{p}_{102} + 6uvw\mathbf{p}_{111} \end{aligned}$$

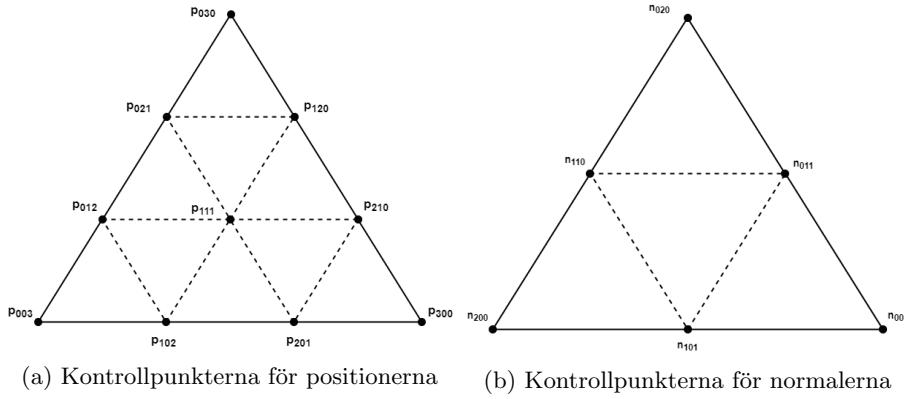
Normalerna interpoleras enligt normalerna i punkterna som anges i figur 4b, på en kvadratisk yta:

$$\begin{aligned} \mathbf{n}(u, v) = & \mathbf{n}_{200}u^2 + \mathbf{n}_{020}v^2 + \mathbf{n}_{002}w^2 + \\ & + \mathbf{n}_{110}uv + \mathbf{n}_{101}uw + \mathbf{n}_{011}vw \end{aligned}$$

För tydlighetens skull bör det nämnas att texturkoordinater interpoleras linjärt över ytan, här är kontrollpunkterna de ursprungliga hörnen och texturkoordinaterna som hörde till dem. Användningsområdet för dessa koordinater i vårat projekt påverkas inte av några artefakter av denna enkla uträkning:

$$\mathbf{t}(u, v) = u\mathbf{t}_{300} + v\mathbf{t}_{030} + w\mathbf{t}_{003}$$

¹Även kallade kontrollkoefficienter



Figur 4: Illustration av kontrollpunkter/koefficienter för en bezieryta. Positioner använder sig av en kubisk interpolation och kräver då 7+3 punkter, medan den kvadratiska normalinterpolationen kräver 3+3.

2.1.3 Hårgenerering

Innan hårgenereringens lösning presenteras måste problemet begränsas. För att för varje triangel i en 3D-modell generera en godtycklig mängd punkter som kopplas ihop till hårstrån i real-tid, krävs det att vi utnyttjar de parallella krafterna i GPU:n. Att utföra dessa beräkningar på CPU skulle tillåta mer information under genereringen av hår, men vara mycket långsammare än alternativet med GPU.

I favören av realtid fortsätter diskussionen med fokus på GPU-alternativet. Lösningen som användes använde sig av information om position och normalriktning för varje triangel. Utöver detta, används också en uniform tyngdkraft för all generering. Detta betyder att ingen hänsyn ges till andra hårstrån, vilket försämrar realism till en viss grad. Ingen hänsyn ges heller till resten av 3D-modellen, vilket innebär att hår kan växa igenom modellytan. Lyckligtvis kan dessa fel döljas med att generera mer hår. Nedan ses pseudokod för algorithmen som körs i geometry-shadern. För varje triangel som kommer in i så görs följande:

Inparameterar :

```
vec3 inPosition [];
vec3 inNormal [];
vec3 forceVector;
int numSegments;
```

inPosition blir positionen för hörnen i triangeln, inNormal blir normalerna för de tre hörnen, forceVector är resultanten av alla krafterna som verkar på håret

och numOfSegments antal segment som ska genereras. Vidare förklaring för hur algoritmen fungerar finns efter funktionskroppen.

Funktionskropp :

```
// Mitten på triangeln och den genomsnittliga normalen
vec3 midPosition = vec3(0,0,0)
vec3 midNormal = vec3(0,0,0)

for i = [1:3]:
    midPosition += inPosition[i]
    midNormal += normalize(inNormal[i])
end

midPosition /= 3
midNormal /= 3

vec3 segmentStartPosition = midPosition
vec3 segmentEndPosition = vec3(0,0,0)
vec3 currHairDirection = midNormal
for i = [1:numOfSegments]:
    gl_Position = segmentStartPosition
    emitVertex()

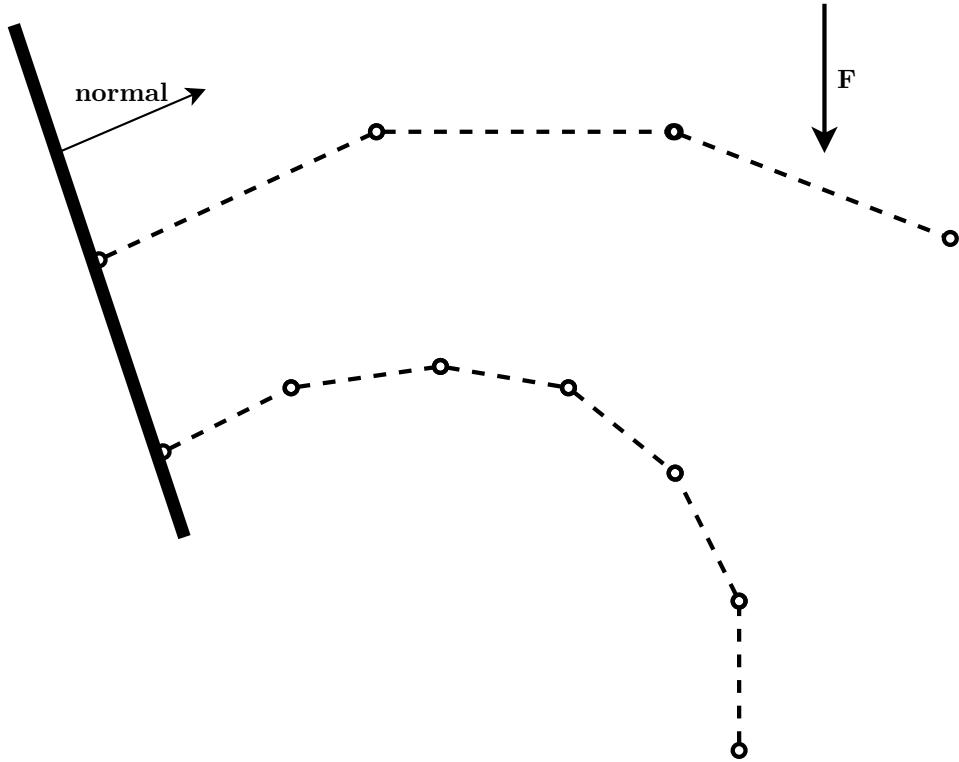
    segmentEndPosition =
        segmentStartPosition + currHairDirection
    emitVertex()
    endPrimitive()

    segmentStartPosition = segmentEndPosition
    currHairDirection += forceVector
    currHairDirection = normalize(currHairDirection)
```

Notera här att kraften som används är densamma för alla noder i håret. Skillnaden mellan hur denna kraft påverkar olika noder är hur tidigare noder har påverkats av dessa krafter. Det första segmentet, hårröten, påverkas endast av den genomsnittliga normalen och varje efterföljande normal faller successivt mer mot den utomstående kraftriktningen. Denna egenskap är något som används tillsammans med segmentlängd för att skapa olika typer av hår.

2.1.4 Hårtyper

Genom att utnyttja egenskapen att segmentens påverkas mer av utomstående krafter beroende på hur många segment som genererats innan, kan vi inkorpora-



Figur 5: Demonstration av hur hår med samma totala längd, men olika antal segment påverkar hur håret böjer sig efter utomstående krafter.

rera segmentlängd för att skapa hår av samma längd, men med olika egenskaper.

Som kan ses i figur 5, så kan segmentantalet ses som ett mått på hur styvt håret är. Mjukt hår simuleras bäst med fler segment och för att få taggigt hår används färre. Viss akning får ges till antalet segment här, ty för många leder till att håret faller snabbt och kan då falla in i modellen. Vill man nödvändigtvis använda sig av ett stort antal segment, bör man vikta summeringen att inte påverkas lika snabbt av kraftresulttanten. Exempelvis:

```
forceWeight = 0.5
currHairDirection += forceWeight * forceVector
```

Fler segment leder till bättre approximation av en kurva, men blir även tyngre beräkningsmässigt.

2.1.5 Animation

I detta avsnitt så nämns vår lösning för animation, eller vår avsaknad av animation. Vi genererar nytt hår i varje frame och förändrar endast kraftresultanten kontinuerligt och långsamt. Detta leder till håret rör sig mjukt. Ifall kraften rör sig snabbt så märks det tydligt hur onaturligt håret rör sig. Men för att simulera vind fungerar denna metod acceptabelt.

2.2 Ljusmodell

Ljussättning av hår skiljer sig från de konventionella metoderna då det inte är tvådimensionell geometri som hanteras, utan endimensionella linjer.

Vi implementerade enklaste möjliga ljussättning enligt Kajiya-Kay-modellen[3]. Det existerar betydligt mer avancerade modeller som såsom Marschner[4], vi tittade dock inte på denna närmare.

Kajiya-Kay-modellen är motsvarigheten av Blinn-Phong för endimensionell geometri med en diffus samt en spekulär komponent. Den diffusa komponenten implementerades helt enligt Kajiya-Kay-modellen och den spekulära komponenten ges av beräknas enligt en *ad-hoc* implementation med inspiration från modellen för den diffusa komponenten.

I de kommande delarna så använder vi följande definition på sinus mellan vektorer.

$$\sin(a, b) = \|a \times b\| = a^T c \quad (1)$$

där c är a ortogonalprojicerat på normalplanet till b .

2.2.1 Diffus komponent

Då det är svårt att få tag på original-pappret där Kajiya och Kay beskriver ljusmodellen så återges en härledning av den diffusa komponenten nedan.

Ekvationerna kan härledas utifrån att ta en cylinder och låta radien gå mot noll. Vi kallar cylindern riktningen för tangent riktningen $\hat{\mathbf{t}}$ och normalen till ytan på cylindern för normalen $\hat{\mathbf{n}}$.

Cylindern parametriseras enligt x, ϕ , där x är position längst med cylindern och ϕ är vinkelns. Låt $s(\phi)$ vara det diffusa tillskottet av ett litet område på cylindern som pekar med normalen $\hat{\mathbf{n}}$ utåt. Vi har då att

$$s(\phi) = \max(0, \hat{\mathbf{l}}^T \hat{\mathbf{n}}) = \max(0, \hat{\mathbf{l}}^T (\cos(\phi)\hat{\mathbf{a}} + \sin(\phi)\hat{\mathbf{b}})) \quad (2)$$

där $\hat{\mathbf{a}}$ och $\hat{\mathbf{b}}$ utgör en bas för normalplanet till tangentriktningen. Vi kan välja basvektorerna för normalplanet godtyckligt, vi väljer att sätta $\hat{\mathbf{a}}$ till ortogonal projektionen av ljusvektorn på normalplanet. Vi låter sedan $\hat{\mathbf{b}}$ vara en basvektor som är ortogonal till både $\hat{\mathbf{t}}$ och $\hat{\mathbf{a}}$.

Detta integreras runt hela cylindern. Gör vi detta för ϕ från $-\frac{\pi}{2}$ till $\frac{\pi}{2}$ så kan vi stryka max-operatorn.

$$s = \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} s(\phi) d\phi = \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \hat{\mathbf{l}}^T \hat{\mathbf{a}} \cos(\phi) d\phi + \underbrace{\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \hat{\mathbf{l}}^T \hat{\mathbf{b}} \sin(\phi) d\phi}_{0} \propto \hat{\mathbf{l}}^T \hat{\mathbf{a}} \quad (3)$$

Där konstanten 2 från integralen kan bakas in i den diffusa konstanten k_d . Uttrycket kan med hjälp av Lagrange's identitet skrivas som

$$f_d(\hat{\mathbf{l}}, \hat{\mathbf{t}}) = k_d \sin(\hat{\mathbf{l}}, \hat{\mathbf{t}}) \quad (4)$$

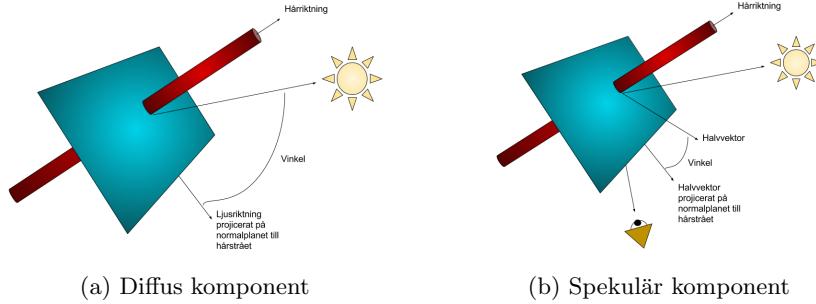
Den diffusa komponenten är helt enkelt storleken på ljusvektorns projektion på normalplanet till linjen. Komponenterna i ljusmodellen har visualiseras i figur 6.

2.2.2 Spekulär komponent

Den spekulära komponenten är dock svår att härleda från dom ursprungliga Bling-Phong ekvationerna. Kajiya och Kay ger endast en *ad-hoc* modell för denna komponent som liknar den spekulära komponenten i Blinn-Phong. Vi implementerade en spekulär komponent utifrån medelvektorn mellan vyriktning och ljusriktning.

Vi tar tolkningen från den diffusa komponenten, men byter ut ljusriktningen mot halvvektorn och applicerar sedan en potens.

$$f_s(\hat{\mathbf{t}}, \hat{\mathbf{l}}, \hat{\mathbf{v}} | \alpha) = k_s \sin\left(\hat{\mathbf{t}}, \frac{\hat{\mathbf{l}} + \hat{\mathbf{v}}}{\|\hat{\mathbf{l}} + \hat{\mathbf{v}}\|}\right)^\alpha \quad (5)$$



Figur 6: Diffus och spekulär komponent i Kajiya-Kay-modellen.

2.3 Skuggmodell

Utöver ljussättning av hår så behövs någon form av skuggning för att det ska se realistiskt ut. Hår är transparent vilket gör att hård skuggning inte kommer att ge något bra resultat. Istället krävs någon form av mjuk skuggning för att ge ett någorlunda bra resultat.

Det finns mycket arbete på skuggning av glesa modeller. Vi tog ursprungligen inspiration från ett papper där håret renderas i flera pass beroende på avståndet från ljuskällan och skrivs in i olika lager i en tredimensionell textur [5]. OpenGL stödjer inte rasterisering i 3D vilken innebär att detta måste skötas i mjukvara, vilket leder till en del komplikationer vid implementering. Den metod som istället implementerades utnyttjar *random access*-metoder så som *imageLoad* och *imageStore* i fragmentshadern för att skriva direkt in i en 3D-textur.

2.3.1 Voxelisering

För att modellera skuggor mellan hår så användes en voxeliseringsmetod. Håren voxeliseras i en textur för att få ett mått på densitet i varje rymdpunkt. Håren voxeliseras utifrån ljusets perspektiv, d.v.s. texturens tredje komponent varierar i z-led sett från ljuset.

Modellen för ljusets propagering genom håret ges enligt ekvation 6.

$$L_{x,y,z} = \alpha^{V_{x,y,z}} L_{x,y,z-1} \quad (6)$$

där $L_{x,y,z}$ är ljusnivå i voxel med spatiellt index x, y, z , $0 < \alpha < 1$ är en parameter på hur mycket ljus som tar sig igenom hår och $V_{x,y,z}$ är någon form av voxelisering av modellen.

Expanderas uttrycket för ljussättningen i en voxel fås

$$L_{x,y,z} = \prod_{k=0}^z \alpha^{V_{x,y,k}} = \alpha^{\sum_{k=0}^z V_{x,y,k}} = \alpha^{C_{x,y,z}} \quad (7)$$

Där $C_{x,y,z}$ är voxeliseringstexturen summerad i z-led.

Voxeliseringen (d.v.s. $V_{x,y,z}$) kan göras på olika sätt med variation i hur bl.a. uppdatering och interpolering av voxlarna görs. Beroende på hur det sker så representerar $V_{x,y,z}$ olika saker.

Vi har testat två metoder för uppdatering av texturen. För varje fragment så kan följande två variationer testats

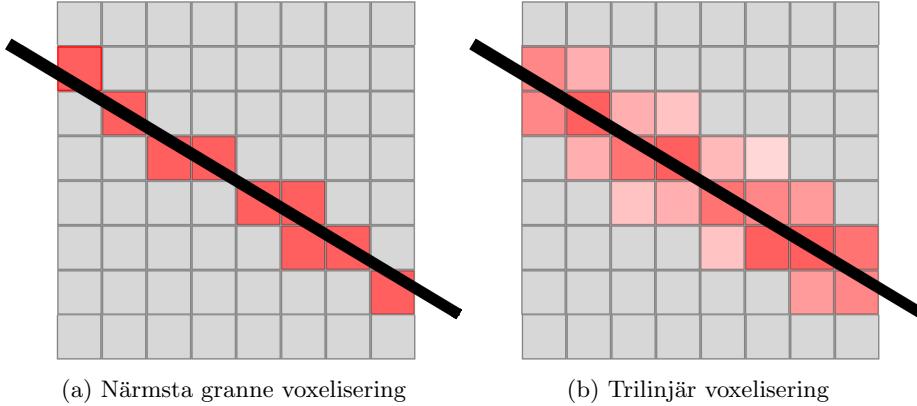
1. Binär voxelisering: Sätt voxel till 1, V representerar ett binärt existenstest
2. Uppräknande voxelisering: Addera 1 till voxel, V representerar ett densitetsestimat

En implementationssvårighet med alternativ 2 är att uppdateringen måste ske atomärt. Detta ger utslag på prestandan då GPU-trådar måste synkroniseras. En annan implementationssvårighet är att i OpenGL-standarden finns stöd för atomär addering, men är endast effektivt implementerat för heltal. Önskas addering av flyttal så måste detta göras genom heltals-operationer (med *imageAtomicAdd*) för att sedan översättas till flyttal senare.

Vi har även testat två metoder för interpolering av fragment in i voxeliseringen, nämligen

1. Närmaste granne: Uppdatera närmaste voxel
2. Trilinjär: Ge bidrag till varje granne, vikta med distans till voxelcentrum

Trilinjär filtrering påminner om *anti-aliasing*-metoder, men är inte riktigt samma sak. Metoderna illustreras i figur 7.



Figur 7: Olika metoder för interpolering vid voxelisering. Bilden illustrerar hur en linje ger bidrag till olika voxlar.

Närmaste granne-interpolering är trivialt implementerat. Trilinjär interpolation är svårimplementerad av samma anledningar som med uppräknings-varianten i voxeliseringen: För att det ska göras korrekt så måste texturen först läsas och sedan skrivas till, vilket kräver tråd-synkning vilket ger utslag på prestandan. Metoden lämpar sig inte heller om binär voxelisering används: Om två fragment skriver till en textur med interpolation, vilket fragments värde ska räknas? Detta blir endast vettigt om fragmentets bidrag till voxeln adderas.

De kombinationer av voxeliseringsmetoder som testats är alltså

1. Närmsta granne binär
2. Närmsta granne addering
3. Trilinjär addering

När håret har voxeliseras så summeras texturen i z-led. Detta gör att när texturen läses i fragmentshadern så fås ett mått på hår många fragment som ligger mellan nuvarande fragment och ljuskällan.

När håret renderas läses den summerade texturen från lämplig voxel för att få ett mått på hur mycket ljus som når fram till fragmentet.

2.3.2 Skuggmappning

Voxeliseringsmetoden kan modellera icke-genomskinliga material genom att sätta voxelvärdet till något väldigt stort. En enklare, mer effektiv och robustare metod är att göra en skuggkarta för dessa material. Detta kombineras med voxeliseringsmetoden för att avgöra den slutgiltiga skuggnivån i varje voxel. Skuggmappningen är implementerad på enklast möjliga sätt genom att skriva djupet för varje fragment (sett från ljuskällans perspektiv) till en textur. Under

renderingsfasen läser fragmentshadern pixlar i ett närområde runt nuvarande fragment från texturen. Skuggvärdet för fragmentet räknas ut enligt ekvation 8.

$$S_{x,y,z} = \frac{1}{|\Omega|} \sum_{i,j \in \Omega(x,y)} (z > D_{i,j}) \quad (8)$$

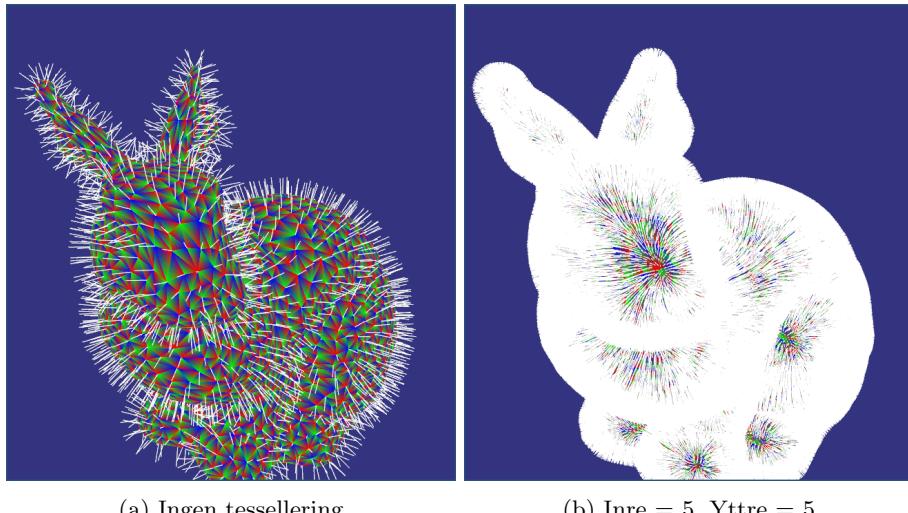
Där S är skugginvån, x, y, z är pixel-koordinaterna för ett fragment, Ω är ett område kring fragmentet i xy -planet och D är djuptexturen (Z-buffern).

3 Resultat

I denna del presenteras bilder av de olika områdena som presenterats under metoder.

3.1 Tessellerering

I figur 8 visas effekten av tessellerering. Med en inre och yttre tesselleringsnivå på 5 så blir mängden trianglar 37 gånger fler.



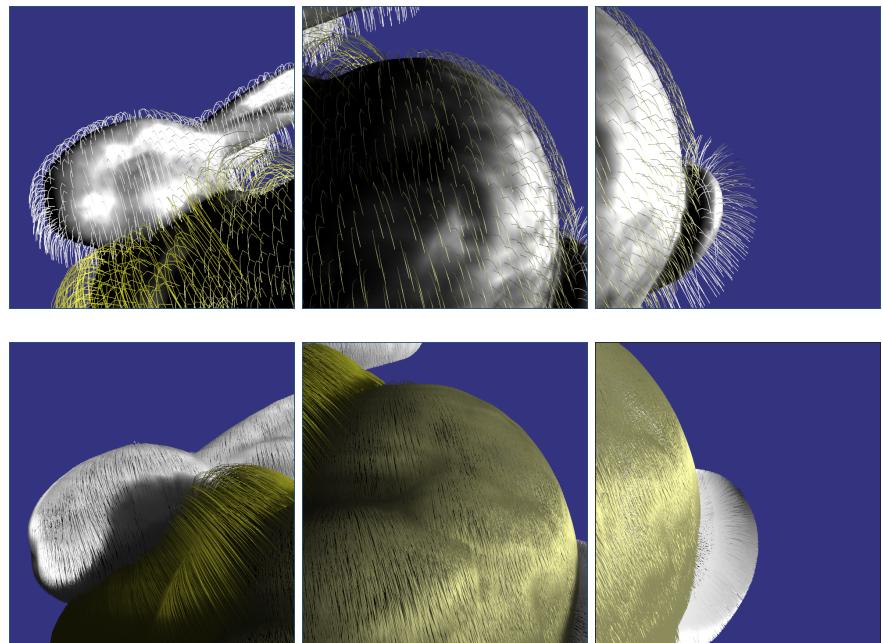
Figur 8: Tesselleringsjämförelse

3.2 Hårtyper

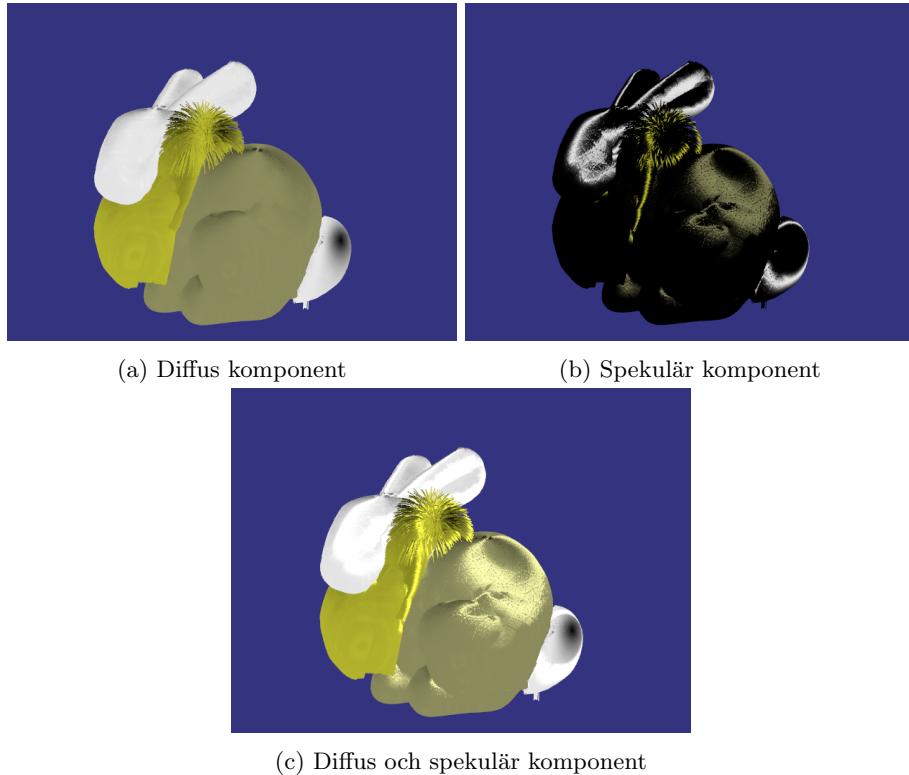
Här användes olika hårtyper över modellen med hjälp av en textur, beroende på vilket värde som lästes ur texturen valdes hårfärg, segmentantal, segmentlängd och viktning på krafterna. I exemplet nedan ser vi vitt kort hår, en luftig man, mörkgult kroppshår och en fluffig svans. För att enklare se hur hårtyperna ser ut så visas den med och utan tesselleringssteget.

3.3 Shading

Effekten av de olika komponenterna i Kajiya-Kay-modellen ses i figur 10. Resultatet av diffus shading ses i figur 10a, spekulär i figur 10b och båda komponenterna tillsammans ses i figur 10c.



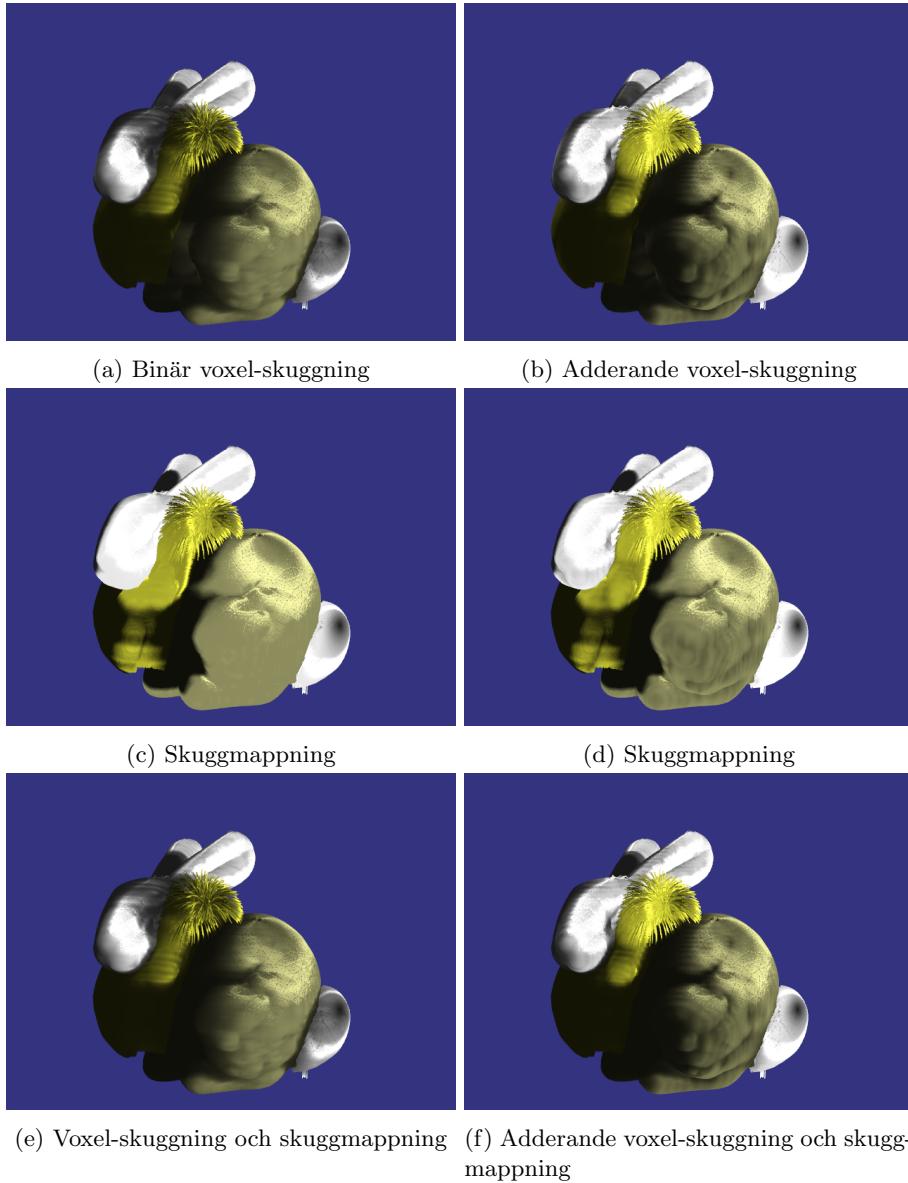
Figur 9: Hårtyper visade på den ursprungliga geometrin (övre) och den tessel-
lerade (undre).



Figur 10: De olika komponenterna i Kajiya-Kay-modellen.

Effekten av de olika skuggkomponenterna ses i figur 11. Resultatet för skuggning med binär voxelisering (med och utan skuggmappning) ses i figurerna 11a, 11c samt 11e. Motsvarande resultat för adderingsmetoden i voxeliseringen visas i figurerna 11b, 11d samt 11f.

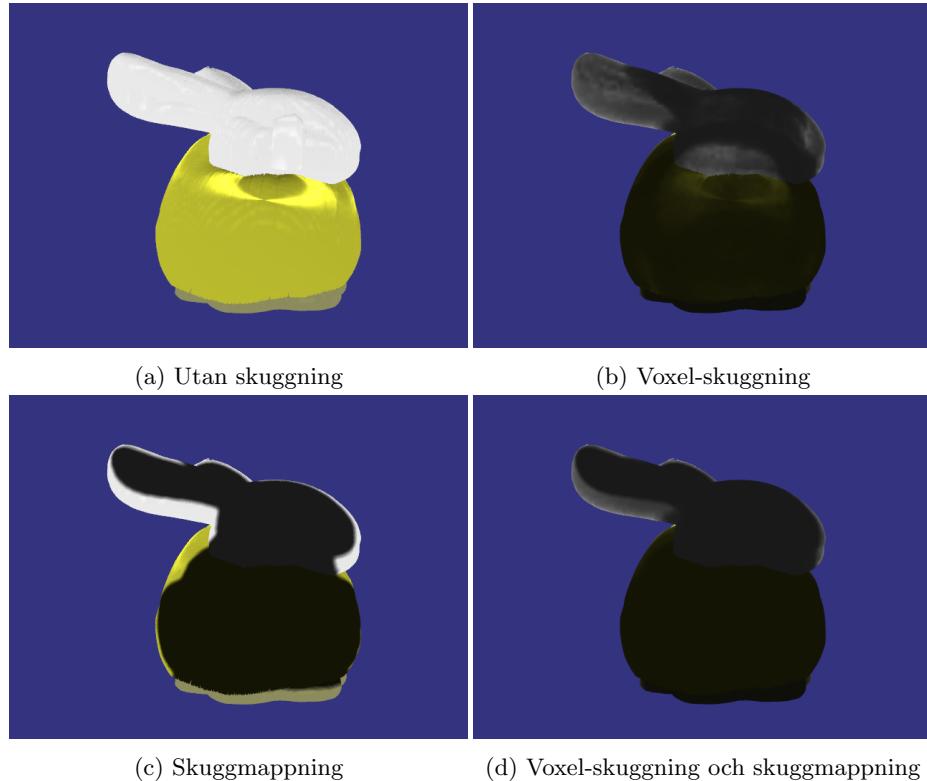
Observera att voxeliseringen inte helt stängts av i illustrationen av skuggmappningen, endast summeringen i z-led är avstängt. Detta illustrerar tydligt skillnaden mellan voxelisingsmetoderna och att adderingsmetoden fungerar bättre som en densitetsestimator (se områden med hög kurvatur, t.ex. veck). Vi ser också tydligt på huvudet hur binär voxelisering ger tydligare hår-till-hår-skuggning. Nackhåret är för glest för ge tillräckligt högt bidrag i voxeliseringen med adderingsmetoden vilket gör att det knappt blir några skuggor på huvudet.



Figur 11: Jämförelse mellan komponenter i skuggmodellen och skillnaden mellan uppdateringsmetoder av voxeliseringen.

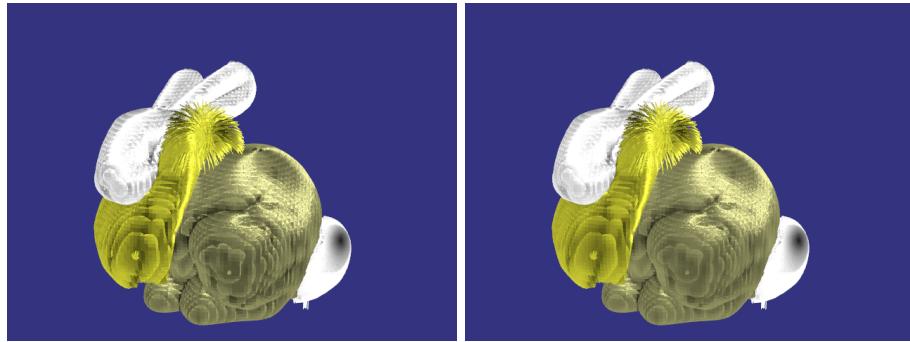
Illustration av artifakter som uppstår med de olika skuggkomponenterna ses i figur 12. Håret har voxelisrats för mjuka skuggor medans kroppen har använts för skuggmappning får den hårdna skuggningen. I figur 12a visar vi resultatet utan skuggning. En tydlig artefakt skapas då ljusmodellen för den spekulära kom-

ponenten blir odefinierad när kameran pekar i motsatt riktning som ljuskällan. I figur 12b visar vi att artefakten kvarstår med endast voxeliserings-skuggning på. Man kan här även se att olika områden blir olika skuggade på ett väldigt onaturligt vis. I figur 12c ser vi att den spekulära artefakten försvinner, men håret är inte skuggat. I figur 12d visar vi resultatet då bågge skuggkomponenter används. Resultatet är relativt fritt från artefakter.



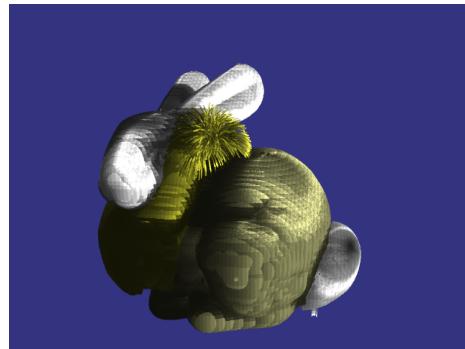
Figur 12: Ljus- och skuggartifakter när kameran tittar rakt emot ljuskällan.

En jämförelse mellan tre olika kombinationer av voxeliseringsmetoder ses i figur 13. Observera att figur 13c har gjorts med summering i z-led för att tydligt illustrera voxeliserings-artefakterna, vilka inte syns annars. Metoderna som använder uppräkning av texturen ses i figurerna 13a samt 13b där bilderna har skapats utan summering i z-led då artefakterna är tydliga nog.



(a) Uppräckande närmsta granne

(b) Uppräckande trilinjär



(c) Binär närmsta granne

Figur 13: Jämförelse mellan närmaste granne och trilinjär filtrering för adderande voxelisering samt närmsta granne och binär voxelisering med binär filtrering

4 Diskussion

4.1 Tessellerering

Tessellerering för trianglar är väldigt bra för att ge mer geometri att rita hår på. Detta är en nödvändighet för att göra bezierinterpoleringen av ytorna gav inte märkvärdiga resultat då det är mycket hår i vägen.

4.2 Hårgenerering

Hårgenereringen är simpel och lämpar sig bra för att växa hår på djur. För att utvidga till människor som har mer intressanta hårtyper så behöver hårtyperna utvidgas. Krulligt hår kräver spiraler och långt hår får gärna ha lite vågor.

Det finns även hår som inte riktigt beter sig som enskilda hårstrån utan som en större enhet. Exempel på detta är afro:t och mustaschen på figur 1. Hår lämpar det sig nog att använda en annan typ av lösning. Kanske man genererar en mesh och använder sig av texturer för att färglägga och manipulera geometrin av meshen.

4.3 Animation

Som nämnts i animationssektionen av hårgenerering så är metoden för animering väldigt begränsat. För att nå upp till mer realistisk animation krävs en tillståndsvektor för hårstråna. *Transform feedback* är ett verktyg i OpenGL som verkar lämpat för att skicka tillbaka information från genereringssteget in i en animeringscykel.

Implementationer för håranimation har gjorts i CUDA, se här en video som använder NVIDIA:s teknologi [6].

4.4 Skuggmodell

Binär voxelisering ger skuggor från hårstrån på modellen.

Uppräcknande voxelisering ger hög densitet i områden med hög kurvatur på modellen. Detta gör att hårsegment som sticker iväg inte avger någon tydlig skugga på andra hårsegment.

Voxeliseringsmetoden fungerar i teorin men ger för stora artefakter för att användas som den fungerar nu.

Det bör även nämnas att metoden vi utgick ifrån gav upphov till mycket artefakter, speciellt när hårsegment gränsar mellan olika lager i 3D-texturen. En slutsats är därför att det underlättar mycket att utnyttja random-access-metoder

och jobba direkt mot en 3D-textur när man experimenterar med liknande problem in grafik.

Referenser

- [1] Dave Shreiner Grahan Sellers John Kessenich Bill Licea-Kane. I: *OpenGL Programming Guide. The Official Guide to Learning OpenGL*. 2013. Kap. 9. Tesselation Shaders. ISBN: 978-0-321-77303-6.
- [2] Xunnian Yang. “Shape aware normal interpolation for curved surface shading from polyhedral approximation”. I: (2013). accessed 25 Dec, 2017. URL: https://www.researchgate.net/The-domain-of-a-Bezier-triangle_257406710.
- [3] J. T. Kajiya och T. L. Kay. “Rendering Fur with Three Dimensional Textures”. I: *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '89. New York, NY, USA: ACM, 1989, s. 271–280. ISBN: 0-89791-312-4. DOI: 10.1145/74333.74361. URL: <http://doi.acm.org/10.1145/74333.74361>.
- [4] Stephen R. Marschner m.fl. “Light scattering from human hair fibers”. I: 22 (juli 2003), s. 780–791.
- [5] Tae-Yong Kim och Ulrich Neumann. “Opacity Shadow Maps”. I: *In Proceedings of the 12th Eurographics Workshop on Rendering Techniques*. Springer-Verlag, 2001, s. 177–182.
- [6] Nvidia CUDA Hair demo 1920x1080 GTX 560 Ti Direct CuII [HD]. Nvidia. 2012. URL: <https://www.youtube.com/watch?v=53Z3VHGQjE8>.