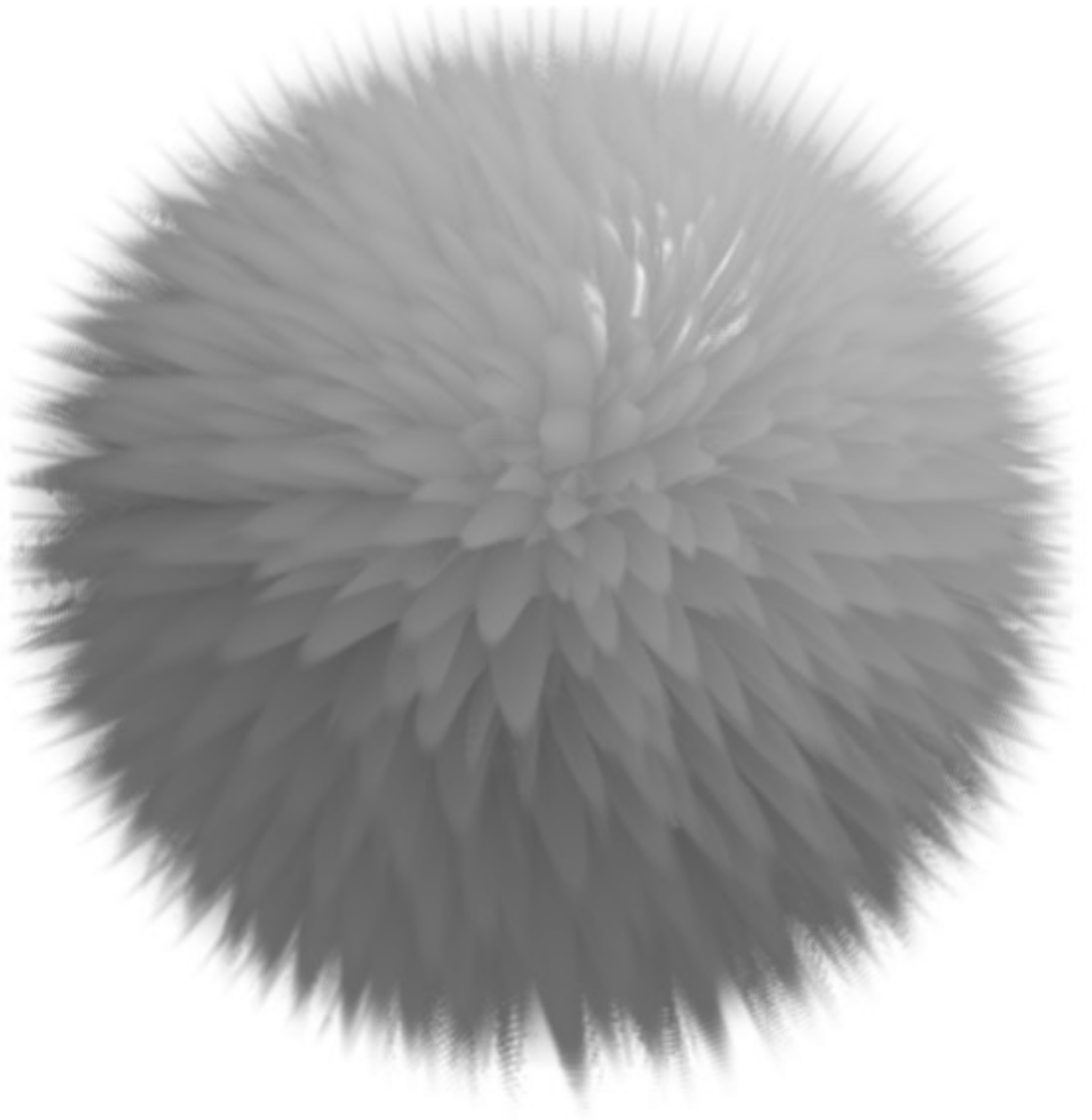
TNM022 Proceduella Bilder

Rendering av proceduell päls i realtid

Jonas Nilsson

jonni957@student.liu.se



Sammanfattning

Jag har undersökt och experimenterat med möjligheterna att rendera päls i realtid med hjälp av GLSL och ett modernt grafikkort. Jag ville generera så mycket som möjligt proceduellt och helt undvika att behöva modellera eller förändra geometri. Målet var att pälsen skulle bli fotorealistisk och att programmet skulle hålla en uppdateringfrekvens på högre än 1fps. Jag har koncentrerat mig på kort päls och hade inga krav på att programmet skulle kunna rendera realistiska långa hårstrån. Överhuvudtaget strävade jag främst efter att visualisera volymen som pälsen upptar.

Till viss del tycker jag mig uppnått mitt mål. Bildfrekvensen ligger oftast på ca 10fps utan optimeringar och ljuset reflekterats rätt från varje hårstrå. Problemen som har uppstått på grund av min metod är aliasing och fula konturer på objekten. Jag kan generera intressanta volymetriska mönster som kan se ut som päls, men metoden är inte bra för täta långa samlingar av hårstrån. Metoden är inte tillräckligt allmän för att vara riktigt användbar. Sfären kan renderas med godkänt utseende. Men försöket med den andra modelen, tekannan, har inte varit lika lyckade. Dock lyckades jag implementera en dynamisk vindanimation av pälsen vilket är mer än jag planerade från början.

Förstudie

För att få en överblick på metoder som andra använt till liknande realtids renderingar gjorde jag sökningar på google och via skolans bibliotek. Tyvärr gav sökningarna inte många metoder att välja på. Den vanligaste och som verkade användas i störst utsträckning bygger på att rendera flera lager ovanpå varandra för att skapa illusionen av en volym[1,2,3]. Jag valde den då den kändes enkel att implementera och det fanns gott om beskrivningar hur metoden kunde utvecklas. I efterhand kunde jag kanske tjänat på att söka efter icke realtids metoder då vissa sådana metoder kanske kan köras i realtid på moderna grafikkort.

Programmeringsspråk

Jag valde att skriva programmet i OpenGL och dess shader språk GLSL eftersom jag haft mycket erfarenhet av det under flera kurser och privata projekt den senaste tiden. Dessutom hade jag en färdig grund att bygga programmet på sedan tidigare.

Grundmodell

Modell

Jag började med att tillämpa en metod som går ut på att rendera olika skal på objektet där varje skal är genomskinligt förutom i de områden som utgör hårstrån. Jag angrep problemet att skapa lager naivt genom att helt enkelt rendera samma objekt flera gånger på samma ställe med olika storlek. Modellen som används är en enkel sfär, samt den klassiska tekannan. Sfärens vertexar, normaler och texturkoordinater genereras i programmet, medan tekannan renderas av en metod i GLUT.

Hitta pälsfunktionen

Det första försöket att skapa en textur som gav ett rimligt pälslikt utseende gjorde jag genom att använda lägga samman cos värdet för texturkoordinat t, och sin värdet för texturkoordinat s. Båda koordinaterna multipliceras först med ett frekvensvärde. Ett tröskelvärde beräknas också med hjälp av en parameter som skickas från programmet och talar om vilket lager man befinner sig i. Resultatet kan sen användas för att beräkna alphavärdet i punkten. Mitt första försök blev illa genomtänkt och resulterade i små justeringar av funktionerna genom försök och misslyckanden. Jag använde då en if-sats för att slänga bort (genom *discard* metoden) fragment som var större än ett visst tröskelvärde, vilket resulterade i tydliga kanter mellan lagren. Funktionen var tillsist onödigt komplicerad och endast anpassad för just den frekvens som jag använde, vilket gjorde att jag funderade lite och skrev om den på ett mer genomtänkt vis.

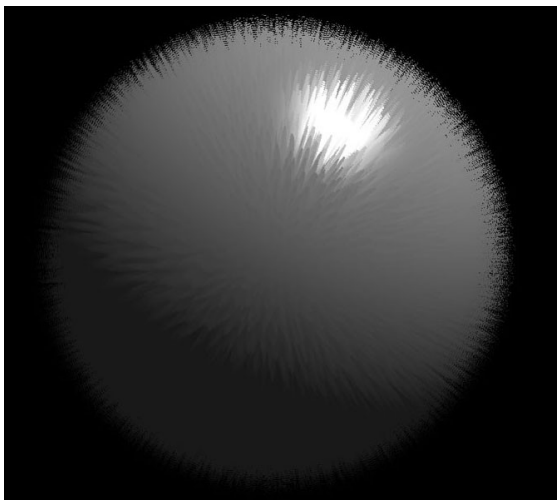


Illustration 1: Ett första försök

Funktionen ”*value”* som varieras beroende på texturkoordinaterna ser ut som följer:

*value = abs(sin(frekvens*2*texurcoord.s)+sin(frekvens*texturcoord.t))*

Tröskelvärdet för lagret beräknas sedan enligt:

tröskel = (2/ sqrt(frekvens))((layer-1) /tot_layers)*

Det slutgiltiga alphavärdet för fragmentet fås sedan genom en smoothstep funktion.

*alpha = smoothstep(tröskel,1.4*tröskel, value / sqrt(frekvens))*

Värdet 1.5 kan ändras till ett mindre för att få skarpare hårstrån och ett större för att få mjukare hårstrån.

Metoden ger ett bra resultat för varierande antal lager med varierande frekvens. Men det kan vara önskvärt att skapa ännu brantare lutning mellan lagren. Kanske en sin funktion inte egentligen passar.

Ljussättning

Per pixel ljussättning

För att kunna skapa en realistisk ljussättning måste ljusberäkningen göras på varje pixel. Jag använder både ambient, diffus och spekulär ljussättning för att skapa så mycket realism som möjligt. Metoden har jag hittat på websidan Lighthouse 3D[7] som är full med nybörjar exempel för GLSL¨och tillämpar en standard lösning för problemet. Man beräkna den diffusa färgen beräknas med hjälp av vinkeln mellan normalen och ljuskällan. Den spekulära beräknas med hjälp vinkeln mellan halvvektorn (vektorn mitt emellan kameran och ljuskällan) och normalen.

Jag har implementerat metoden precis som den stod, men ändrat mindre detaljer. Styrkan på den beräknande diffusa, spekulära och ambienta färgen avtar i de lägre lagren, för att skapa en enkel illusion av volym.

Förändring av normaler

Utan några förändringar är normalen i varje punkt motsvarande interpolerade normal från närliggande vertices. Därmed blir ljusättningen av varje lager precis likadan som om bara ett lager skulle renderas. Inga enskilda hårstrån kan därför ses.

För att beräkna nya normaler behövs tangenten och binormalen (dvs kryssprodukten av tangenten och normalen). Tangenten finns tillgänglig precis som normalen genom ett attribut, kallat vTangent, och går att komma åt i vertexshadern. Den kan därför skickas med som en variabel till fragmentshadern och användas precis som normalen. Det verkar som att vTangent inte är tillgänglig på alla grafikkort. Hittills har jag bara fått det att fungera på nyare nVidia kort. Om inte vTangent används är man tvungen att specificera tangenten och spara den för varje vertex när man skapar modellen. Det finns antagligen någon annan lösningen, men då vTangent fungerar bra för mig så har jag inte undersökt de andra möjligheterna.

Den nya normalen beräknas sedan genom att addera en vektor i tangentens riktning och en vektor i binormalens riktning till den gamla normalen[6].

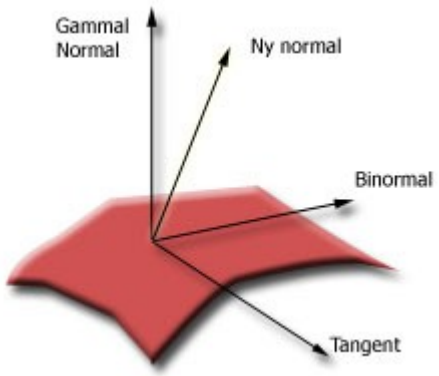


Illustration 2: Normal, binormal och tangent

Det intressanta problemet är då hur stora dessa vektorer skall vara. Det gällde att hitta ett värde på hur mycket normalen skulle lutas i varje ritkning. Här kommer GLSLs funktioner dFdx/dFdx. Funktionerna ger förändringen i x eller y led för en variabel, vilken som helst som shaderna använder, mellan fragmenten. Därför kan dFdy/dFdx användas för att veta hur mycket variabeln *value* förändras i x och y led. Om normalen vrids med det värdet kommer normalerna på ett ”hårstrå” att luta i samma riktning. Detta kräver att man använder en kontinuerlig funktion som är lika för varje lager. Jag använde först det beräknade alpha värdet i dFdy/dFdx, men det värdet är inte kontinuerlig då det beräknas genom en stegfunktion vilket gör att normalerna runt kanterna på en cirkel i ett lager vrids väldigt mycket. Vilket gör att kanterna på varje cirkel är väldigt tydlig. Därför använder jag dFdy/dFdx av *value* funktionen direkt och fick ett mycket bättre resultat.

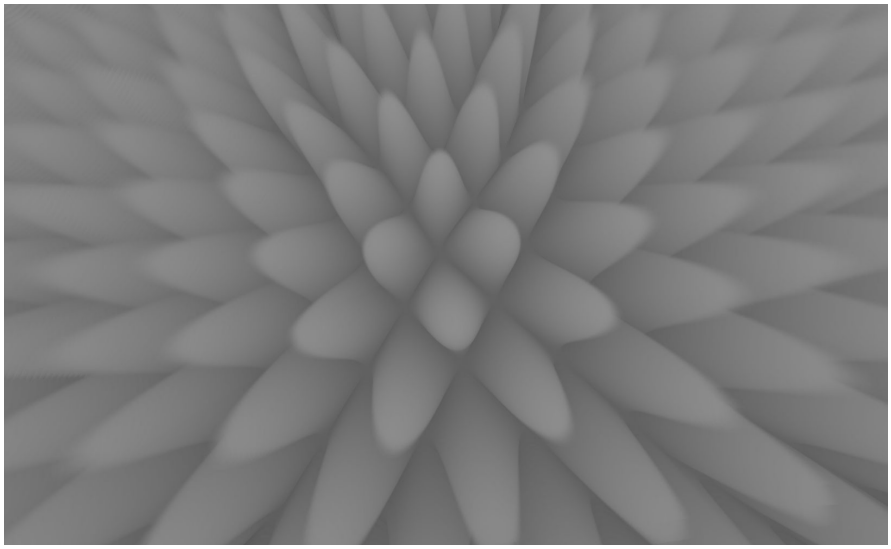


Illustration 3: Närbild på lagren sedda framifrån med per pixel ljusättning och ändrade normaler

Öka realismen

Perlin Noise

För att gör pälsen oregelbunden behövs någon noise funktion. Men då perlin noise är väldigt tungt att köra i shadern och noisefunktionen inte behöver variera i tiden, bara i två dimensioner så kan en förberäknad noisefunktion användas. Den beräknas innan programmet startar renderingsloopen. Och sparar de genererade värdena i en textur. Den texturen skickas till shadern som snabbt kan ta fram rätt texturvärde för att använda noise. Texturen kan användas till att flytta texturkoordinaterna i förhållande till texturvärdet, vilket gör att pälsfunktionen får ett mer oregelbundet utseende. Det är också möjligt att använda texturen för att ändra frekvensen som används i pälsfunktionen. Storleken på den textur som genererats av programmet är mycket viktig och måste anpassas för att matcha frekvensen som pälsfunktionen använder. Annars kommer funktionen variera för mycket mellan texturkoordinaterna och brytas upp till ett fult mönster. Därför skapar programmet en ny noise-textur varje gång frekvensen ändras av användaren med en större storlek för högre frekvenser och mindre för lägre frekvenser.

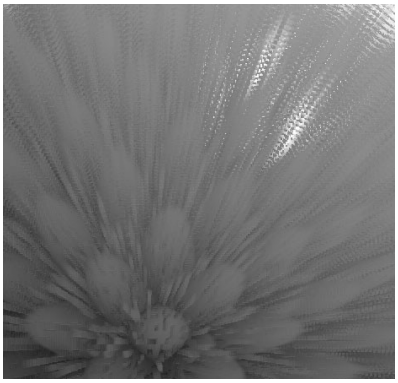


Illustration 4: Stor noise textur

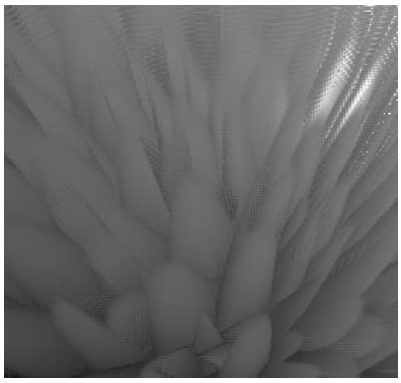


Illustration 5: Liten Noise textur

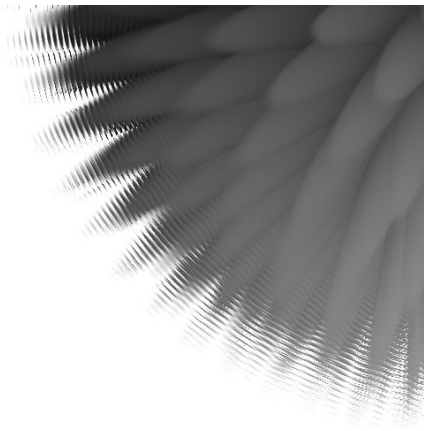
Vindfunktionen

Den funktion jag hittade för vind var beskriven i ett kort paper och behandlade vind på grässtrån[4]. Men tillvägagångsättet var likadant med olika lager av varierande transparens. Dock utgick metoden från att alla lager från början inte var translaterade. Jag tyckte att det var en mycket liten förändring och ändrade min metod så att alla modeller ritas ut med samma storlek. Tyvärr gör depth testet i Opengl att de undre sfärerna klipps bort innan renderingen. Jag försökte stänga av depth testet och dylikt, men det förstör istället renderingen av lagren och skapar mycket artefakter.

Sen slog det mig att jag bara behövde flytta vertexen i normalens riktning så många lager som behövdes för att först få ned den till en utgångspunkt. Väl där kan jag direkt implementera algoritmen som beskrevs.

Algoritmen går ut på att flytta vertexen i normalens och vindens riktning, men göra det på ett sätt som behåller avstånden mellan lagren.

Kanten, problem och lösningar



Den stora bristen och det svåra, kanske fatala, problemet med metoden är att illusionen av en volym bryts då lagren syns tydligt i silhuett. Jag har försökt använda en lösnings idé som beskrivits i flera dokument[1,2,3] men försöker göra den så proceduell som möjligt.

Bakgrunds polygon

I de dokument jag läste som beskrev metoden med skal ingick också lösningar på de uppsprukna kanterna. Den vanligaste var att rita polygoner vända mot kameran ut från kanten med en textur som smälter in i pälsens mönster (referens). Jag ville föröka göra något liknande rent proceduellt och gjorde därför en polygon i mitten av sfären som renderas med en shader som försöker härma shadern som används på sfären. Men jag har inte fått fram en funktion som helt smälter in i objektet. Shaderna beräknar avståndet till mitten av objektet och vinkeln mellan texturkoordinaten och en upp vektor. Alpha värdet på konturen bestäms sedan av avståndet till mitten och vinkeln med en smoothstep och en sinus-funktion. Normalerna på polygonen förändras också och lutas bort i från centrum. För att simulera rotationer av sfären ändras fasen på sinus-funktionen beroende på värdet på de två rotations variablerna i programmet. Det ger ett utseende av rörelse som inte stör, så länge man inte tittar närmare på kanten.

Metod ser bara ibland bra ut och kan bara användas med den ”hårboll” som renderas nu. Det skulle vara intressant att hitta en generell metod som kunde användas på alla objekt. Metoden fungerar inte heller när objektet deformeras för mycket, av till exempel en kraftig vind.

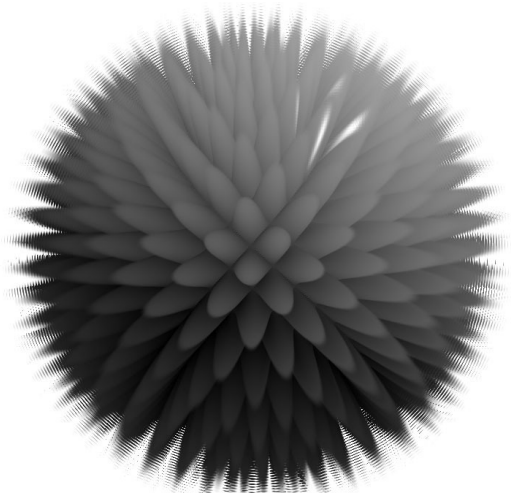


Illustration 7: Utan bakgrundspolygon

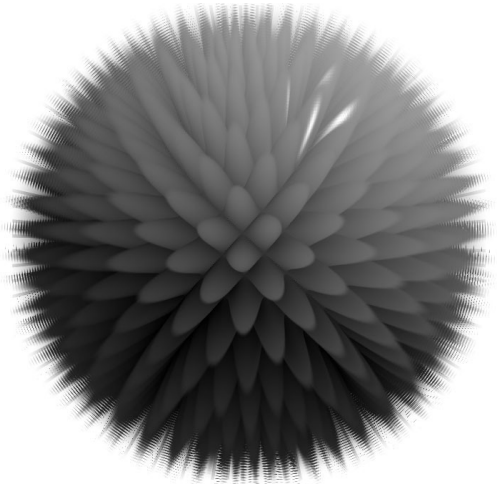


Illustration 6: Med bakgrundpolygon

Blur

Ett sista, kanske desperat, tillägg för att ge ett mjukare utseende på renderingen var att helt enkelt använda en metod från bildbehandling för att minska värdet av de höga frekvenserna och släta ut bilden. Framebuffern kopieras till en textur efter en rendering och kan därmed skickas till ett shader program. Programmet använder texturen och ett gauss filter för att beräkna de nya filtrerade värdena för fragmentet. En polygon som täcker hela skärmen renderas sedan med programmet och ger en slutgiltig filterad bild. Metoden hjälper till att minska bruset i kanterna av objektet. Men gör hela bilden lite för suddig. Det skulle vara bättre att hitta en metod som endast filtrerar kanterna. Shadern som används kommer ifrån ett tidigare projekt och borde nog anpassa för att ge ett bättre resultat.

Resultatet

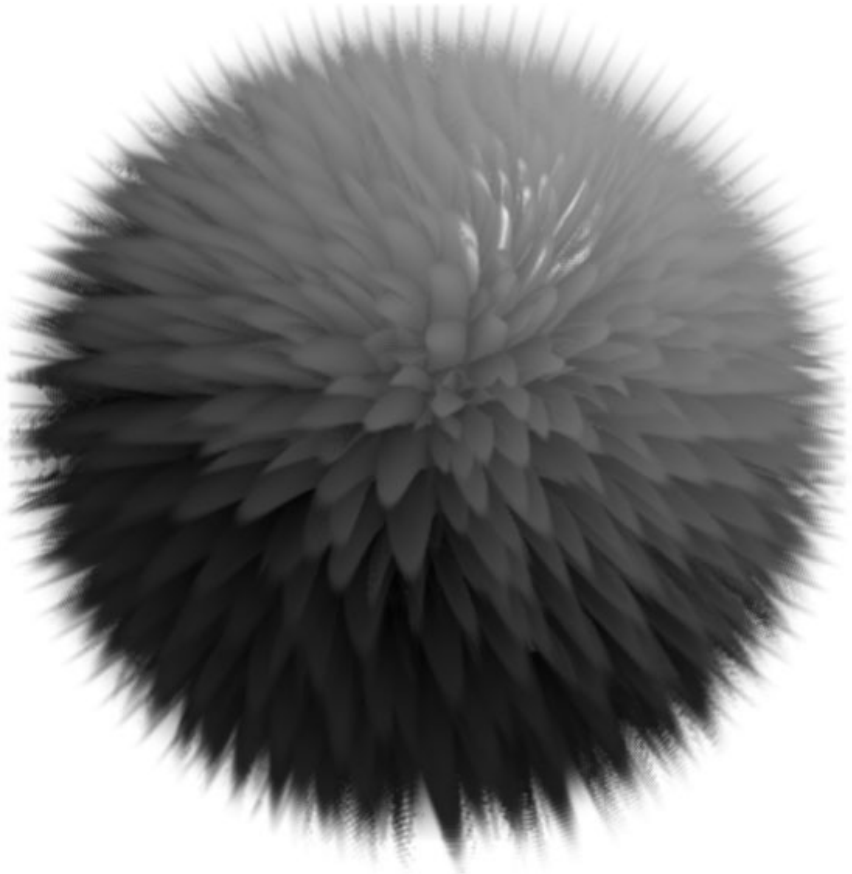


Illustration 8: Sfären renderad med 30 lager, bakgrundpolygon, vind och all ljussättning

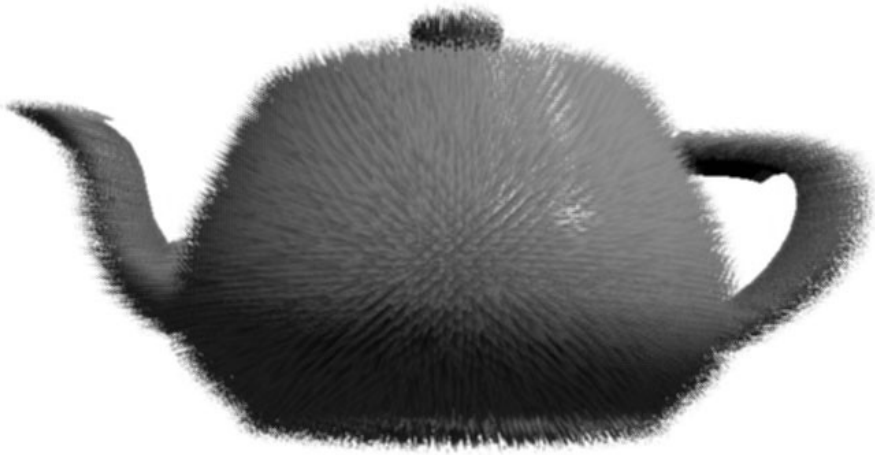


Illustration 9: Tekannan renderad på samma vis som sfären ovan

Prestanda

Testdatorn har varit en AMD64 3500+ med 1 gb ram och ett Nvidia 6800LE kort. Programmet körs med 30 renderade lager, vind beräkning, bakgrundspolygon och blur i ett 640*480 pixlar stort fönster. Med dessa inställningar hinner ungefär 8 bilder renderas per sekund, vilket känns på gränsen till interaktivt. Antagligen finns det många möjligheter till optimeringar att göra, både i renderingsloopen och i shader programmen som kan ge en viss prestandaökning.

Problem, förbättringar

Aliasing

Då ett hårstrå i många fall är mycket mindre än en pixel stor räcker inte de beräkningar jag nu gör för att få ett bra utseende. Steget mellan två fragment i shadern blir så stort att tröskelberäkningen och vridningen av normalen inte längre kommer att stämma. Jag har fått en föreslagen lösningen från Stefan Gustavsson[5] där funktiones gradient och förändringen av texturkoordinaten mellan pixlarna skall används för att beräkna värdet. Jag har gjort experiment med metoden men lyckas inte använda den för att lösa aliasing problemet. Gradient verkar korrekt beräknad men problemet ligger att använda den i en smoothstep funktion för att få fram värdet. Jag har ändå lagt med den shader koden i programmet för att någon annan kanske skall upptäcka vad jag gör fel.

Utveckling

Perlin Noise skulle kunna beräknas i shadern för att ge en bättre distortion av texturkoordinaterna. Men det var inte rimligt på det grafikkort jag nu använde. Kanske kommer perlin noise vara en del av hårdvaran i senare grafikkort vilket löser många problem.

Vind och dynamik kan implementeras mer komplett för att att simulera gravitation och rörelser av objektet.

Det finns parametrar som skulle kunna förändras för att bättre kunna påverkan utseendet av håret och göra det möjligt för användaren att till exempel påverka hårets täthet och liknande.

Sen kan man välja andra mer intressanta färger än grått på pälsen också.

Andra metoder

Den bästa metoden för att visualisera volymen bör vara genom någon volymrenderingsteknik. Men det fungerar antagligen bäst för korta täta hårstrån.

Pälsen kan också skapas med en punkter (exempel GL_POINTS) men det innebär att varje hårstrå tar upp en vertex, vilket inte är ett bra val för tätare päls. Däremot kan långa enskilda hårstrån lättare skapas. Kanske skulle en kombination av volymrendering och punkter kunna fungera.

Överhuvudtaget känns det kanske som att en helt ny metod är det bästa valet för att lösa alla problem, det går inte att komma undan det faktum att man endast simulerar en volym med min metod.

Referenser

Papers:

1. Real-Time Rendering of Fur – Sheppard
2. Real-Time Fur over Arbitrary Surfaces – Lengyel, Praun, Frinkelstein, Hoppe
3. Interactive Fur Modeling Based on Hierarchical Texture Layers – Yang, Sun, Wang, Wu 2006
4. Real Time Animated Grass – Bakay, Lalonde, Heidrich
5. Beyond the pixel: towards infinite resolution textures – Gustavson

Böcker:

6. Texturing & Modeling: A Procedural Approach, Third Edition – Ebert, Musgrave, Peachy, Perlin, Worley

Websida:

7. http://www.lighthouse3d.com/opengl/glsl/