UPPSALA
UNIVERSITET

TVE 15 037 juni
Examensarbete 15 hp
Juni 2015

Framtagning av ritplattestyrd penselsimulation för GIMP

David Ramnerö

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Development of a Tablet Controlled Brush Simulation as an Extension of GIMP

David Ramnerö

The project was intended to produce a brush simulation plugin to the image processing program GIMP which would use a 6DOF drawing tablet for user input. Too ambitious goals coupled with compability issues between the resources used resulted in that no meaningful software was produced. Experience was however obtained, and this paper is intended to give an overview of relevant concepts for- and the process of creating a brush simulation.

Handledare: Anders Hast
Ämnesgranskare: Christoffer Karlsson
Examinator: Martin Sjödin
ISSN: 1401-5757, TVE 15 037 juni

Populärvetenskaplig sammanfattning

Digital bildbehandling är ett område som funnits med ända sedan datorer först utvecklades, om inte ännu längre, då oscilloskop användes för att generera bilder redan innan det. Idag är digital bildbehandling ett mycket stort fält och digitalt ritande likaså. Det är en självklar del av modern produktframtagning, byggteknik och annan ingenjörsverksamhet. Digitalt ritade konceptbilder är även en grundsten i dagens film- och spelindustrier.

En grundläggande del i digitalt skapande är givetvis hur användare och dator interagerar. Den standardiserade metoden för att ge en dator input är via datormus och tangentbord, men då dessa visat sig i många fall inte vara optimala för intuitiv bildbehandling har alternativa lösningar länge funnits tillgängliga. Den digitala ritplattan, med en yta som registrerar rörelser hos en pekpenna, är ett vanligt sådant datortillbehör. Dess design är tänkt att emulera fysiskt ritande och till följd av detta är dess användande intuitivt förutsatt att användaren är bekant med fysiskt ritande eller skrivande.

Detta projekt behandlar dock främst digitalt målande. Målsättningen för projektet var att skapa en applikation som med indata från en ritplatta simulerar en pensel och skickar ett utifrån simulationen genererat avtryck till ett bildbehandlingsprogram. Medan fysiskt ritande är relativt enkelt att simulera - en pennas avtryck kan oftast approximeras med en homogen linje - så erbjuder de fysiska egenskaperna hos färg och penslar en helt annan utmaning. Detta visar sig främst i icke-realistiskt målande där penseldrag kan skönjas. Att simulera sådant målande digitalt är väldigt komplext då penseln kan ändra form, vara blöt eller torr och användas på olika sätt för olika avtryck. De fysiska egenskaperna hos färg spelar även de in: färgen kan rinna, blandas med annan färg, ge en transparent effekt då den appliceras tunt osv. Att simulera dessa egenskaper är ett komplext problem och något som har börjat behandlas först på senare år då en fysiksimulation med många element, vilket en pensel med tusentals strån i hög grad har, kräver mycket processorkraft.

Innehållsförteckning

1 Inledning

1.1 Bakgrund

Fysisk simulation av komplexa system, till exempel en pensel med hundratals strån, är ett ungt forskningsfält på grund av att det först på senare år funnits teknologi med tillräcklig datakraft för tillfredsställande simulationer. Just penselsimulationer är, med rätta, inte heller fokus för en särskilt stor del av arbetet inom fältet. Penselsimulationer har dock, förutom en kommersiell efterfrågan, angränsningar till simulationer av kontinuum och mera generella hårsimulationer med tänkbara användningsområden inom medicin och ingenjörsvetenskap.

En pensel kan ofta ha hundratals eller tusentals strån och det oftast inte optimalt att simulera alla dessa som individuella kollisionsobjekt. I arbetet Industrial-Strength Painting with a Virtual Bristle Brush (DiVerdi 2010) tas dock denna approach och modeller med upp till 100 penselstrån simuleras under körtid på ett satisfierande sätt. Detta är dock knappt tillräckligt och i dagsläget är det en mer sannolik lösning att approximera hela penselhuvudets beteende från ett mindre antal simulerade kontrollstrån. Det kan göras antingen genom att skapa ytterligare strån som själva inte ingår i fysiksimulationen genom interpolation av våra kontrollstrån, eller genom att skapa en fysisk kropp kring våra kontrollstrån som deformeras av dessa. I det senare fallet skapas avtrycket av den deformerade kroppen snarare än stråna själva. Avvägningar man måste göra i dessa fall går oftast att bryta ned i fidelitet mot hastighet och det är en del där mycket experimenterande kan förväntas komma att behöva göras för att få en tillfredsställande simulation.

1.2 Målsättning

Syftet med detta projekt är att ta fram en penselsimulation i form av ett tillägg till Open Source-bildbehandlingsprogrammet GIMP[9]. I det fall att projektet ej resulterar i en meningsfull produkt skall fokus istället riktas mot att redogöra för omständigheterna kring skapandet av en simulation av den här typen samt tänkbar potential hos denna. Den tänkta simulationen skall i första hand styras av data från en ritplatta med sex frihetsgrader: 2D positionsangivelse, 2D vinkelangivelse, rotation (kring axeln längs penselns skaft) samt tryck. Ritplattan vi använder oss av är en Wacom Intuos5. Wacom är den ledande producenten av ritplattor[2] och de har en öppen API för utvecklare. Windowsversionen av denna API heter Wintab.

2 Teori

I detta avsnitt redogörs för bakomliggande teori relevant för de koncept som projektet berör.

2.1 Hårsimulation

Penselsimulation angränsar direkt till hårsimulation och kan ses som ett specialfall av den senare, vilket är ett område det då naturligt också gjorts mer forskning inom. De delar av simulationen som är av störst intresse kan dock skilja sig mellan hår och penselsimulationer, i Simulating the Structure and Dynamics of Human Hair: Modelling, Rendering and Animation (Rosenblum, 1991) till exempel, som tillsammans med ett par andra arbeten kan anses ha lagt grunden för området, så läggs fokus på att få en hårsimulation som ser så verklig ut som möjligt. Författarna diskuterar bland annat självskuggning av hårstrån och liknande egenskaper som enbart har kosmetisk funktion. Dessa egenskaper är för oss helt ointressanta.

Figur 1: Hårsimulation används flitigt inom bl. a. animerad film. Till exempel för en av huvudkaraktärerna från Disneys 'Monsters Inc.'.

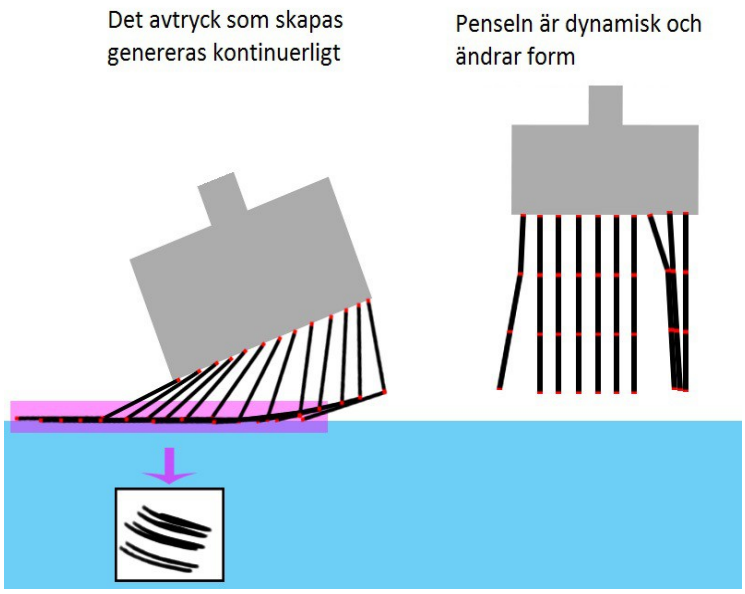
Den grundläggande dynamiken är däremot identisk, där precisheten av att simulera varje hårstrå för sig vägs mot processeringseffektiviteten av att interpolera eller klumpa ihop strån till en enhetlig massa. Avgörande i denna avvägning är att vår simulation krävs att beräkningarna kan göras under körtid och därmed behöver vara mycket effektiva, medan vid många andra användningsområden av hårsimulation så som i animerad film eller för-renderade animationer till datorspel tillåts beräkningarna ta mycket längre tid.

2.2 Penselsimulation

I denna sektion kommer relevanta koncept relaterade till penselsimulation behandlas och avvägningar gällande fidelitet mot beräkningshastighet diskuteras. Mycket av teorin i denna sektion är informerad av kapitlet A Brush Stroke Synthesis Toolbox från boken Image and Video-based Artistic Stylisation (DiVerdi, 2013)som ger en översikt över ämnet penseldragssyntes, vilket inkluderar penselsimulation.

En typisk penselsimulation består av tre delar: målaryta (canvas), penselskaft samt penselhuvud, som är uppbyggd av en mängd penselstrån eller en kropp som emulerar en mängd penselstrån. Det finns flera tänkbara sätt en penselsimulation kan styras: med tangentbord, datormus, ritplatta eller automatiserat. Det är dock osannolikt att en simulation som styrs av mus eller tangentbord skulle vara av speciellt intresse då dessa metoder för input är oprecisa och inte har någon uppenbar fördel över en ritplatta. En

automatiserad penselsimulation där någon form av artificiell intelligens tillåts styra penselhuvudet är ett intressant koncept med stor potential, men att utforska detta ligger utanför projektets ramar. Detta projekt fokuserar på en penselsimulation som styrs med en ritplatta.



Figur 2: Demonstration över hur penselns dynamik är tänkt att fungera

Datan från ritplattan används för att flytta det simulerade penselskaft i vilket penselhuvudet sitter fast. Penselhuvudet modelleras i det generella fallet av ett antal strånodeller (så kallade splines), som är uppbyggda av ett antal stela, linjära segment sammankopplade i leder med en viss spänstighet. Då stråna kolliderar med det plan som motsvarar den canvasen, samt då de kolliderar med varandra, deformeras de. Utifrån denna dynamik skapas sedan ett penselavtryck, tänkbart genom någon form av projektion av de spline-segment som är tillräckligt nära canvasen. Detta avtryck är simulationens utdata och i vårat fall är det tänkt att denna sedan ritas på den digitala canvasen i GIMP (se figur 2.). Målarfärg är som bekant flytande med varierande viskositet och arbeten som Real-time simulation of watery paint (Van Laerhoven, Van Reeth 2007) har skapat modeller för att simulera detta. De flesta penselsimulationer stödjer dock inte denna egenskap hos färgen utan gör en enkel approximation genom att t. ex. variera transparens och tjocklek hos avtrycket simulationen ger. Detta är vad som ämnas göra i detta projekt.

För att digitalt målande skall kunna konkurrera med fysiskt målande krävs att det är responsivt, intuitivt samt ger feedback till användaren. Modellen med en simulerad pensel som styrs av en ritplatta anses vara intuitiv nog. Responsivitet har helt och hållet med implementationen att göra, och det finns en rad applikationer för digitalt målande som är responsiva nog för enkelt användande. Feedback är den kanske den punkt som skiljer digitalt målande mest från fysiskt målande: det är svårt att återskapa känslan av

en pensel mot canvas. Om man, som vi, använder en ritplatta finns det inga möjligheter för taktil feedback, utan vi får i nuläget nöja oss med visuell feedback i form av en grafisk representation av den simulerade penseln. Det finns dock arbeten, som DAB: Interactive Haptic Painting with 3D Virtual Brushes (Baxter, 2004) vilka använder en specialutvecklad pekpenna som ger motstånd beroende på penselsimulationen. På detta sätt känner användaren penselns tryck mot canvasen vilket ger en högre grad av feedback.

Att ta fram ett penselavtryck att rita upp från simulationen och att genomföra själva ritandet i GIMP ses som ett område som potentiellt kan komma att ta mycket arbete, om det ens är utförbart inom projektets ramar. GIMP:s ritfunktion utgår från en lång rad parametrar som tillsammans definierar det avtryck som skapas. Detta system är relativt kraftfullt men bedöms ej vara dynamiskt nog för att få ut ett satisfierande avtryck från simulationen endast genom att låta simulationen ändra på dessa parametrar. En mer sannolik lösning är att göra större modifieringar i källkoden i GIMP för att stödja att rita direkt från simulationen. Implementationen av detta hjälps enormt av att GIMP är Open Source, dels då det därför är möjligt att modifiera källkoden och dels då det därför finns en aktiv mod-community från vilken hjälp möjligtvis kan fås.

2.2.1 Canvasen

Canvasen i simulationen behöver i de flesta fall bara vara en plan, stationär yta som tillåter kollision med friktion med de andra delarna.

2.2.2 Penselskaftet

Penselskaftet är det objekt penselstråna sitter fast i och det är penselskaftet användaren styr för att interagera med simulationen. I vårat fall skall penselskaftet följa ritplattans pekpenna så precist som möjligt. Penselskaftet kan bäst manipuleras genom att helt enkelt i varje tidssteg sätta dess positions- och rotationsparametrar till de hos pekpennan. Det kan vara värt att tänka på att detta kan resultera i konstiga interaktioner i simulationen då penselskaftet i praktiken "hoppar" till dess nya position och då kan råka hamna inuti andra kollisionsobjekt eller få lederna som håller fast penselstråna att sträckas ut till den grad att simulationen blir ostabil. Om detta är ett problem kan det möjligtvis tänkas att man istället låter krafter verka på penselskaftet på ett sådant sätt att det rör sig mot att vara positionerat som pekpennan. Att få detta att ge en responsiv och precis simulation är dock troligen mycket svårt så det är antagligen inte ett verkbart alternativ. De flesta moderna ritplattor anses dock vara så precisa i sin positionsangivelse att det bör gå att skapa en simulation som manipuleras absolut utan konstigt beteende.

2.2.3 Spline

Spline är ett essentiellt begrepp inom hårsimulation. En spline är en strånodell. Denna modell kan antingen vara kontinuerlig, genom att utgå från till exempel en Bézier-kurva, eller diskret. Att implementera dynamik och kollision för kontinuerliga modeller är långt mer komplicerat än för diskreta (DiVerdi 2013) och till följd av detta behandlar det här projektet främst en simulation med diskreta splines.

Den typiska diskreta modellen består av ett antal linjära, stela segment sammankopplade med spänstiga leder. Ju fler segment varje spline består av ju bättre approximeras ett penselstrå, men flera segment innebär också givetvis ett mer komplext system och därmed långsammare beräkningar.

De linjära segmenten har rimligtvis en mycket liten radie i förhållande till deras längd, och kan ha kollisionsobjekt som är rätblock, cylindrar eller ellipsoider. Rekommenderat är att använda ellipsoider då dessa kräver minst beräkningskraft i ODE. [5] Det är även önskvärt att friktion uppstår mellan segmenten och canvasen då de är i kontakt. När kroppar i ODE kolliderar skapas en tillfällig led, kallad kontaktled, mellan de två kropparna i kollisionspunkten. Kontaktledsstrukturen har stöd för friktion där riktning på friktionskraften samt dess styrka finns som variabler.

Lederna som kopplar ihop de linjära segmenten kan vara av olika typer, men för en spline är en led med många frihetsgrader önskvärd. De två för detta praktiska alternativen som finns i ODE är en boll-i-hylsa-led (ball-in-socket joint) och en universalled (universal joint). En universalled kan tänkas som två gångjärn, vinkelräta mot varandra och mot de segment de sammankopplar. Detta ger två frihetsgrader. En boll-i-hylsa-led har en ytterliggare frihetsgrad i form av rotation, dvs. de segment som sammankopplas kan rotera i förhållande till varandra. Detta är önskvärt i normala hårsimulationer då denna effekt visar sig ha stor inverkan på hårets dynamik enligt kapitlet A Brush Stroke Synthesis Toolbox från boken Image and Video-based Artistic Stylisation (DiVerdi, 2013), men då penslar oftast har korta, spänstiga strån är det för penselsimulationer generellt tillräckligt att använda universalleder.

En annan diskret modell som används av bl. a. (Baxter, 2004) utgår från en rad partiklar sammankopplade med spänstiga fjädrar. Denna metod ger en modell som approximerar en kontinuerlig spline bättre. I dess vanliga implementationsform stödjer den dock ej kollision emellan strån, utan endast kollisioner mellan partiklarna och canvas. Om kollision strån emellan ej önskas är denna metod att föredra då den naturligt kräver mindre processorkraft. Implementationen i (Baxter, 2004) styr dessa splines deformationen hos en mesh och kollision strån emellan

2.2.4 Penselhuvudet

Penselhuvudet är den mest komplexa delen av penseln då ett verkligt penselhuvud består

av tusentals penselstrån och att simulera dessa på ett bra sätt är det essentiella problemet inom penselsimulation. I dagsläget är det orealistiskt att tänka sig simulera alla tusentals individuella penselstrån som enskilda splines (trotts att detta i någon mån åstakommits i (DiVerdi 2010) då simulationens beräkningar måste vara snabba nog för att kunna göras under körtid. Det finns två vägar runt detta: interpolation utifrån kontrollsplines eller approximation med en deformerbar kropp.

2.2.4.1 Interpolation utifrån kontrollsplines

I denna modell består penselhuvudet av ett mindre antal splines, förslagsvis med en i varje "hörn" av den form penselhuvudet önskas ha, och med övriga utplacerade däremellan med jämna avstånd. De övriga tänkta penselstrånas beteende kan sedan interpoleras från dessa kontrollsplines beteenden. Denna interpolation kan göras på många sätt och att optimera denna ligger utanför ramarna för detta projekt, men den enklaste tänkbara modellen vore att använda en viktad superposition, dvs. om penselstrå X ligger mellan kontrollsplines Y och Z med ett relativt avstånd 0,25 till Y och 0,75 till Z skulle X:s position bestämmas enligt:

$$X_{position} \sim (1\text{-}0{,}25) * Y_{position} + (1\text{-}0{,}75) * Z_{position}$$

2.2.4.2 Approximation med deformerbar kropp

Denna modell går ut på att representera penselhuvudet med en eller flera triangel-mesher som är bunden till en skelettstruktur. Detta är ett vanligt system inom 3D-modellering och animation och inget detta projekt går in närmare på. Vi låter i denna modell en eller flera kontrollsplines utgöra skelettet för den deformerbara kroppen (alt. kropparna), vilken själv inte är ett kollisionsobjekt. Penselavtrycket skapas genom en projektion av kroppen på canvasen. Denna metod för penselsimulation användes bland annat i arbetet DAB: Interactive Haptic Painting with 3D Virtual Brushes (Baxter, 2004), vilket var ett av de första arbeten inom fältet som använder sig av en tredimensionell simulerad pensel.

Det som denna modell har svårt att simulera är de mer speciella deformationer en pensel kan undergå. Tag en större pensel med platt huvud till exempel, dess penselhuvud kan dela på sig så att penselstråna sitter i klungor med tomrum emellan dem. Beteende av den typen är svårt att simulera med en deformerbar kropp då dessa vanligtvis inte kan delas upp i flera mindre kroppar. Att använda flera kroppar löser detta i någon mån, men det finns då ändå begränsningar på hur pass mycket penseln kan deformeras beroende på hur många kroppar som används.

2.3 Open Source

Mjukvara tillskrivs ettiketten Open Source om den uppfyller en rad principer gällande

licensering och tillgänglighet. Idén är att mjukvarans källkod skall vara offentlig och tillgänglig för alla att använda, kopiera, modifiera och härleda vidare versioner från. Konceptet introducerades och förespråkas av stiftelsen Open Source Initiative (OSI) baserat på det snarlika konceptet Free Software. OSI har fastställt en rad kriterier som en programvarulicens måste uppfylla för att enligt deras definition kunna benämnas som Open Source[3].

Utifrån denna definition kan man även se Open Source som en utvecklingsmodell för mjukvara där tanken är att alla utvecklare skall ha möjlighet att lära sig av och obehindrat bygga vidare på det som andra utvecklare har åstadkommit. Allt arbete i detta projekt rättar sig efter Open Soure-principen och alla resurser i form av mjukvara är licenserade på så sätt att de kan benämnas som Open Source.

2.4 Resurser

Projektet utgår från en lång rad resurser i form av hårdvara och mjukvara. I detta avsnitt ges en kortfattad genomgång av dessa samt motivering till varför de valts att användas.

2.4.1 Ritplattan: Wacom intuos5

En ritplatta är ett datortillbehör med en yta som registrerar beröring av en pekpenna eller ett eller flera fingrar. Typiskt representerar ritytan ett fönster, eller hela skärmen, på datorn och datormusens position styrs av positioneringen av pekpennan (eller fingrar) på plattans rityta. Många moderna ritplattor har mer avancerade funktioner än detta så som möjlighet att registrera olika grader av tryck eller rotation av pekpennan. Ritplattor används i stor utsträckning inom digital bildframtagning då de ger mer precis kontroll jämfört med en datormus, samt att de i högre grad emulerar icke-digital ritande.

Ritplattan som används är en Wacom intuos5 som styrs med en pekpenna. pekpennan har i det här fallet sex frihetsgrader: 2D positionsangivelse, 2D vinkelangivelse, rotation (kring axeln längs pekpennans skaft) samt tryck. Wacom är den ledande tillverkaren av ritplattor[2] och har en öppen API som fungerar för alla deras produkter. Att utveckla utifrån en Wacomprodukt täcker därför den största delen av användare, vilket är önskvärt.

2.4.1.1 Wintab

Wintab är Wacoms API för Windows operativsystem och har kommit att bli Windows standardinterface för ritplattor. Wintab distribueras med alla Wacoms produkter och finns även fritt tillgängligt på internet med officiell dokumentation och stöd för utvecklare.

2.4.2 ODE

Open Dynamics Engine (ODE) är en Open Source fysikmotor utvecklad av Russel Smith m.fl. Motorn är skriven i C/C++ men går även att använda med Python genom tillägget pyODE. ODE är välkänd på grund av dess robusthet och användarvänlighet och har bland annat använts till en rad datorspel. ODE valdes över andra alternativ utifrån detta, samt utifrån att det används i kapitlet A Brush Stroke Synthesis Toolbox från boken Image and Video-based Artistic Stylisation (DiVerdi, 2013), som på många vis var utgångspunkten för projektet.

2.4.2.1 DrawStuff

DrawStuff är ett program som distributeras tillsammans med ODE och används för att visualisera simulationer skapade i det.

2.4.3 GIMP

GNU Image Manipulaton Program (GIMP)[9] är ett Open Source bildredigeringsprogram i stil med det kanske kändare, men proprietära programmet Photoshop. GIMP är ett nämnvärt åstadkommande av Open Source-gemenskapen i hur det utvecklas av en öppen grupp volontärer och trots detta lyckas i alla fall i någon mån konkurera med kommersiella alternativ för professionellt användande. GIMP har stöd för tilläggsprogram med en implementationsmall kallad GIMP Plugin Template, men dessa är begränsade i vad de kan göra så det är sannolikt att man vid implementation av något som ritfunktionsinput från en fysiksimulation behöver modifiera själva källkoden till GIMP.

3 Metod

3.1 Förstudie

Innan projektet påbörjades studerades kapitlet A Brush Stroke Synthesis Toolbox från boken Image and Video-based Artistic Stylisation (DiVerdi, 2013). I detta kapitel, som i skrivande stund finns tillgängligt online som PDF, går DiVerdi igenom forskning som gjorts på syntesisering av penseldrag och redogör för de grundläggande algoritmer som behövs för simulation av penslar. Penseldrag syftar till de irreguljära avtryck som kan ses som atomära byggstenar av icke fotorealistiskt målande. Dessa penseldrag efterliknas digitalt genom att skapa en fysiksimulation där en simulerad pensel interagerar med en canvas och ett avtryck projiceras och ges som utdata av simulationen. DiVerdi går igenom denna process och redogör för avvägningar kring optimering av simulationen.

Ett problem som även tas upp är att det inte i dagsläget är tänkbart att simulera en pensel

med tusentals penselstrån där varje strå är ett simulerat fysiskt objekt. Två tillvägagångssätt läggs fram som möjligheter runt detta: antingen kan man utifrån ett mindre antal simulerade strån interpolera hur de andra stråna skulle bete sig eller så kan man approximera penseln som en deformerbar kropp och då alltså inte simulera penselstrån för sig.

3.2 Implementation

I denna sektion redogörs kortfattat och översiktligt för vad som krävs för att implementera de olika resurser som används i projektet.

3.2.1 Wintab

För att använda Wintab i ett program måste headerfilen `wintab.h` inkluderas, vilken installeras med alla Wacom-ritplattor som en del av drivrutinerna. För att en applikation sedan skall kunna användas med en ritplatta måste den först definiera ett så kallat context-objekt (typnamn HCTX) där applikationsfönstret samt adressen till Logcontext-strukturen (som innehåller variabler relaterade till contexten) ges som indata. Detta är nödvändigt för att ritplattan skall veta vilken applikation data skall skickas till. Applikationen behöver även en funktion som mottar Windows-meddelanden precis som vilken Windowsapplikation som hellst. Typen av meddelande läses av och det intressanta fallet är då meddelandet är ett datapaket från ritplattan (betecknat WT_PACKET). Det finns även ett meddelande som signalerar att contexten bör förrändras, som till exempel när en ny ritplatta kopplas in, och en del andra meddelanden relaterade till Wintab som inte är intressanta för oss. Dessa känns igen genom att de alla har prefixet WT. Övriga Windowsmeddelanden, med prefixet WM, rör standardfunktioner för applikationsfönster så som förrändring av fönstrets storlek, eller fönstrets skapande (vilket man i det generella fallet vill skall trigga skapandet av en context för fönstret).

I det fall att det mottagna meddelandet är ett WT_PACKET kan önskvärd data extraheras från detta och behandlas av programmet. Wacom erbjuder licensfri exempelkod för alla utvecklare som visar hur dessa koncept används och som kan approprieras obehindrat.

3.2.2 ODE

ODE, då det är Open Source, distributeras i form av källkod men inkluderar premake4 som hjälper användaren att konfigurera ODE för sitt system och sin tänkta IDE och installerar libraries och includes så att de enkelt kan länkas till. ODE är i Windows oberoende av externa program eller bibliotek, förrutom givetvis de som ingår i Windows systemfiler och dessa hittas och länkas till på egen hand. Efter installation

implementeras ODE enkelt genom att bara inkludera `ode.h`, samt definiera om enkel eller dubbelprecissionsmatematik skall användas för beräkningarna (genom `#define dSINGLE` till exempel). Dubbel precission ger högre fidelitet men tar mer beräkningskraft så vilken av dessa som används är en viktig avvägning. Det går givetvis att enkelt byta emellan de två genom att ändra vad som definieras, men det finns risk att simulationer som är stabila med den ena kan ge konstigt beteende med den andra.

De grundläggande komponenterna i en simulation i ODE är en värld i vilken kroppar existerar, några av dessa möjligtvis sammankopplade med leder, och med potential att krafter verkar på dem. Hur dessa och övriga koncept i ODE fungerar, samt vilka funktioner som kan kallas på dem finns väl dokumenterat i den officiella ODE-manualen. Kroppar i ODE är alltid av en viss geometrisk kollissionstyp som till exempel ett rätblock eller en cylinder och de har en massa som avgör hur de påverkas av krafter som verkar på dem. Följande exempelkod visar hur man i ODE enkelt kan skapa en låda med med en viss massa, fördelad på ett visst sätt:

```
body[i] = dBodyCreate(world);
dBodySetPosition(body[i], XPOS, YPOS, ZPOS);
dMassSetBox(&m, density, WIDTH, DEPTH, HEIGHT);
dMassAdjust(&m, MASS);
dBodySetMass(body[i], &m);
box[i] = dCreateBox(space, WIDTH, DEPTH, HEIGHT);
dGeomSetBody(box[i], body[i]);
if(i>0){
	joint[i-1] = dJointCreateUniversal(world, 0);
	dJointAttach(joint[i-1], body[i-1], body[i]);
	dJointSetUniversalAnchor(joint[i-1], XPOS, YPOS, ZPOS);
}
```

Notera att de sista fem raderna kod skapar en universalled som sammankopplar den låda som precis skapats med en annan, tidigare skapad kropp. Denna exempelkod efterliknar koden för att skapa de splines som modellerades i prototypsimulationen.

3.2.2.1 DrawStuff

DrawStuff kan användas genom att inkludera `drawstuff.h` samt definiera vilka ritfunktioner som skall användas. Exempel:

```
#define dsDrawBox dsDrawBoxD
```

DrawStuff initieras enkelt med ett funktionskall där storlek på det önskade fönstret anges, och ritfunktioner placeras i programmets stegfunktion. DrawStuff är enkelt och intuitivt. Följande exempelkod ritar en blå låda på en specifik plats med specifika dimensioner:

```
dReal BoxDimensions[3] = { BoxWidth, BoxDepth, BoxHeight };
dsSetColor(0, 0, 255);
dsSetTexture(DS_WOOD);
dsDrawBox(dBodyGetPosition(Box), dBodyGetRotation(Box), BoxDimensions);
```

(Notera att den första raden deklarerar en dReal-variabel, vilken definieras i ODE och inte DrawStuff)

3.2.3 GIMP

The GIMP Team erbjuder en kodmall för utveckling av tilläggsprogam (plugins) till

GIMP vid namn GIMP Plug-in Template. Denna innehåller kod som gör att din plugin kommer, efter den installerats, visas i GIMP under filter-menyn, samt lite andra fuktioner med syfte att man skall kunna utveckla plugins för GIMP utan att behöva sätta sig in i hur källkoden fungerar helt och fullt. Denna kodmall kräver att GTK+ 2 development bundle är installerad (obs! inte GTK+ 3). GTK+ är en verktygslåda för att skapa fönster och andra gränssnittselement och GIMP är uppbyggd utifrån detta.

GTK+ har en lång rad externa library-beroenden så som Glib, cairo, Pango, ATK, gdk-pixbuff och X11 som också måste installeras. I Linux-miljö görs detta enkelt med *apt-get -install*, men för Windows krävs att ett Unix-emulationsprogram, som t.ex. Cygwin används. Under projektets arbetsgång var detta en stor problemkälla som ledde till att utvecklingen flyttades från Windows-miljö till Linux-miljö.

Kodmallen kräver även att GIMP development package, som innehåller header-filer och lib-filer för GIMP. Installationen av denna är dock relativt intuitiv.

4 Resultat

På grund av otaliga problem som uppkom under projektet är resultat i form av mjukvara i det närmaste obefintlig, det blev helt enkelt inte mycket tid över till utvecklande. Det som främst åstakommits är en evaluering av processen att skapa ett program av den här typen utifrån öppna resurser (se implementationsdelen för en redogörelse kring detta), samt erfarenhet i de miljöer som använts, och av för penselsimulation relevanta koncept.

Projektet utfördes under en tio-veckorsperiod av en person, dock med tillgång till en mentor samt en handledare med expertis inom området visuell information och interaktion. Efter förstudie lades projektet upp utifrån att tanken att först sammanfoga resurserna, så att en GIMP Plugin-bas skapas med ODE integrerat och med möjlighet att läsa av data från ritplattan, och från detta utveckla programmet så långt som skulle hinnas med under projektets gång. Projektets framåtskridande hindrades dock i flera steg på grund av svårigheter med att sammanfoga resurserna som det var tänkt.

4.1 Problem med att sammanfoga resurserna

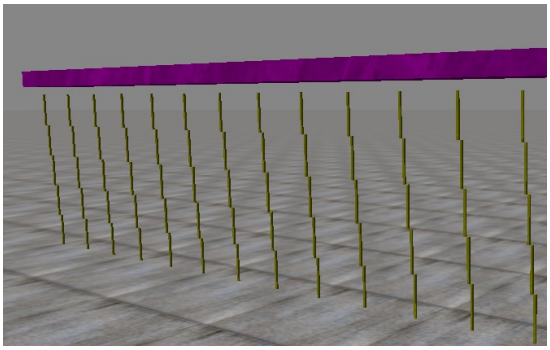
Arbetet påbörjades i Windows-miljö i vilken otaliga problem uppstod kring att sätta upp GIMP Plugin Template med rätt resurser för utveckling. Denna kodmall kräver GTK+ development bundle som har en rad externa library-beroenden vilka visade sig vara svåra att installera korrekt i Windows-miljö (detta redgörs för mer utförligt i 3.2.3). Dessa kompileringsproblem lyckades aldrig kringgås vilket var den huvudsakliga anledningen till att projektets målsättning ej uppfylldes.

Arbete gjordes för att istället utveckla i Linux-miljö, då i operativsystemet Ubuntu. Detta underlättade enormt för att sätta upp GIMP Plugin Template på rätt sätt och i denna miljö kompilerade tilläggsmallen korrekt och var därmed redo för att

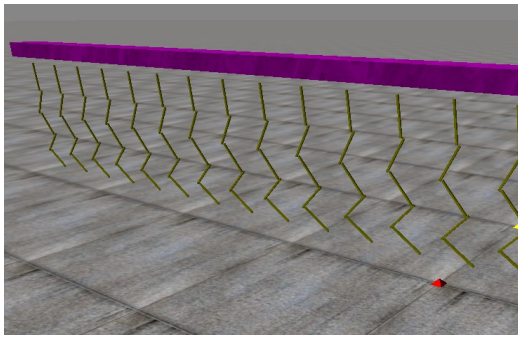
programmera till.

I Linux-miljön uppstod istället problem med att sätta upp ODE på korrekt sätt. Problemet var att ODE vid installationen inte skapade alla libraries på rätt sätt utan ett par bibliotek skapades för 32-bits-Linux istället för det 64-bitar-system som användes. Detta problem visade sig svårt att komma runt och lyckades inte kringgås utan att nya problem uppstod. Det visade sig även i Linux-miljön vara svårare att utveckla för Wacom-ritplattan då ingen direkt motsvarighet till Wintab fanns med samma dokumentation och rersurser för utvecklare.

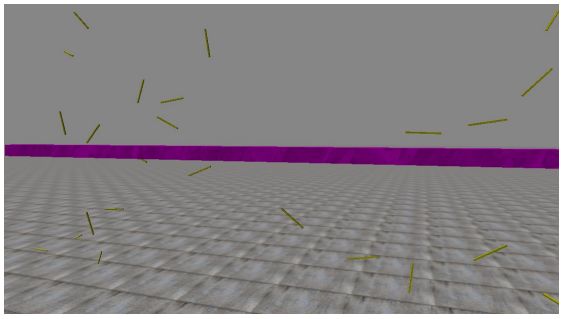
4.2 Prototypsimulation



Figur 3: Penselmodell i ODE



Figur 4: Splinesens beteende kräver vidare optimering för att efterlikna penselstrån



Figur 5: Simulationen är ej stabil i dess nuvarande stadie

En del jobb gjordes dock på en simulationsprototyp i ODE. En modell för kontrollsplines skapades med linjära stela segment sammankopplade med spänstiga universalleder. Ett penselskaft modellerades med ett antal splines kopplade till det (Se figur 3). Integration av ritplattestyrning visade sig vara svårare än förrutspått, och andra problem som uppstod under arbetsgången tog upp mycket utav utvecklingstiden vilket resulterade i att detta aldrig implementerades framgångsrikt. Istället styrdes prototypen med hjälp av tangentbordet.

Denna styrning var långt ifrån optimal och som figur 4 visar optimerades aldrig splinesen för att trovärdigt simulera penselstrån då simulationen inte kunde testas på önskvärt sätt utan att ritplattestyrning först implementerades. Från detta, och från att tangentbordsstyrningen var mycket oprecis följde också att simulationen var mycket instabil (se figur 5). Som diskuterats i 2.2.2 leder detta till instabilitet då penselskaftet "hoppar" till dess nya position och då kan råka hamna inuti andra kollisionsobjekt eller få lederna som håller fast penselstråna att sträckas ut för mycket.

5 Diskussion

5.1 Reflektion över resultat

Den huvudsakliga anledningen till att väldigt lite konkret åstakommits under projektet är de allt för ambitiösa mål som sattes. Projektet utfördes under 10 veckor av en person. Utföraren av projektet hade vid projektets påbörjan dessutom en väldigt begränsad erfarenhet av de resurser och de miljöer som användes, vilket förklarar varför målsättningen var, tycks det i efterhand, minst sagt orealistisk. Det som kan ses som positivt med detta är den erfarenhet som nu har erhållits av de diverse resurser som använts. Det ter sig sannolikt att i det fall att ett liknande projekt skulle utföras så skulle dessa erfarenheter kunna vara till stor hjälp, dels under utvecklingen, dels vid projektplaneringen, och resultera i en mer realistisk målsättning då bedömningen av arbetsåtgång och potentiella problemkällor skulle vara långt mer välinformerad.

Den simulationsprototyp som redovisas i resultatdelen bedöms inte vara mycket av ett åstadkommande. Då ritplattestyrning inte lyckades implementeras gick det inte att testa simulationen på ett meningsfullt sätt så väldigt lite tid spenderades på att utveckla denna. Till följd av detta efterliknar inte penselmodellen i simulationen en verklig pensels beteende på ett satisfierande sätt, och hur mycket arbete som krävs för att optimera detta beteende har inte utforskats. Bedömningen är dock att detta bör vara relativt enkelt att genomföra, förutsatt att ritplattestyrning implementerats.

5.2 Identifiering av problemkällor

Den huvudsakliga problemkällan bedöms vara att man försökte sammanfoga så många resurser för att skapa en mjukvara. Att få GIMP, ODE och ritplattan att arbeta tillsammans som utgångsläge ses i efterhand som ett icke-optimalt tillvägagångsätt. En bättre lösning hade troligen varit att fokusera på en implementation, till exempel att integrera ritplattestyrning i fysiksimulationen utan att blanda in GIMP, som utgångsläge. Detta i sig hade gott och väl kunnat vara hela vidden av projektet, möjligtvis hade till och med det varit väl ambitiöst för ett tioveckorsprojekt. Oavsett bedöms det som ett avsevärt mycket sundare tillvägagångsätt då man efter potententiell lyckad framtagning av något sådant kan implementera det i GIMP, snarare än att försöka göra det från utgångsläget, och därmed få ett mera skalbart projekt.

En annan uppenbar problemkälla var brist på erfarenhet av de miljöer projektet utfördes i. En vecka av projektet spenderades på att installera och lära sig Linux, vilket sedan visade sig mindre lämpligt än Windows för utförande av projektet. Att denna bedömning inte kunde göras i ett tidigare stadie utan att behöva investera en veckas tid på detta härör från, liksom den väl orealistiska projektplanen, bristande kunskap och erfarenhet.

6 Sammanfattning

Projektet ämnade skapa en ritplattestyrd penselsimulation som var tänkt kunna ge ritavtrycksdata till bildbehandlingsprogrammet GIMP. Detta försöktes åstakommas genom att skapa en GIMP-plugin, implementera fysikmotorn ODE i denna, samt implementera Wintab som tillåter programmering av ritplatta. Problem uppstod i sammanfogandet att dessa resurser och detta resulterade i att ingen mjukvara som närmar sig de utsatta målen skapades. Trotts detta erhölls erfarenhet av processen att skapa ett program av denna typ, samt av de miljöer och bibliotek som användes och fördjupad kunskap om bakomliggande teori för penselsimulation.

7 Förslag för framtida arbete

För framtida projekt med syfte att ta fram penselsimulationer rekommenderas en mer fokuserad målsättning än den som sattes upp för detta projekt. Det tycks logiskt att fokus läggs på att först ta fram själva simulationen, innan arbete görs på att få ut data ur denna och skicka det till ett bildbehandlingsprogram. Det som då står ut utöver allt annat som en förutsättning för att skapa en penselsimulation av den typen detta projekt behandlar är att stöd ritplattestyrning implementeras i ODE.

7.1 Implementation av ritplattestyrning i ODE

Simulationsprototypen som skapades i detta projekt använder DrawStuff för visualisering av simulationen. DrawStuff bygger på OpenGL och har därför själv inte en funktion som behandlar mottagna Windows-meddelanden (MainWindProc). För att implementera Wintab behöver MainWindProc skapa en context för ritplattan då meddelandet för skapandet av fönstret (WM_CREATE) mottas. MainWindProc måste även behandla åtminstonde meddelandet med datapaket från ritplattan (WT_PACKET). Det verkar troligt att det för detta krävs att modifikationer görs i OpenGL, alternativt att man använder ett annat grafikbibliotek än DrawStuff.

8 Referenser

[1] A Brush Stroke Synthesis Toolbox In Image and Video-Based Artistic Stylisation, Vol. 42 (2013), pp. 23-44, doi:10.1007/978-1-4471-4519-6_2 by Stephen DiVerdi edited by Paul Rosin, John Collomosse Retrieved May 25 2015.

[2] Summary of Financial Results for FY 3/15 by Wacom http://investors.wacom.com/media/files/investor-relations/2015-english/E-1_150430_Financial%20Results%20of%20FY0315_E_final.pdf Retrieved May 25 2015.

[3] The Open Source Definition by The Open Source Initiative http://opensource.org/docs/osd Retrieved May 25 2015.

[4] Simulating the Structure and Dynamics of Human Hair: Modelling, Rendering and Animation by Robert E. Rosenblum, Wayne E. Rosenbaum , Edwin Tripp III 1991 https://design.osu.edu/carlson/history/PDFs/hair-paper.pdf Retrieved May 26 2015.

[5] ODE Manual by Russel L. Smith and the ODE Community http://ode-wiki.org/wiki/index.php?title=Manual Retrieved May 26 2015.

[6] DAB: Interactuve Haptic Painting with 3D Virtual Brushes by Bill Baxter, Vincent Sheib, Ming C. Lin, Dinesh Manocha 2004 http://gamma.cs.unc.edu/DAB/files/DAB-small.pdf retrieved June 3 2015

[7] Industrial Strength Painting with a Virtual Bristle Brush by Stephen DiVerdi, Aravind Krishnaswamy, Sunil Hadap 2010 http://dl.acm.org/citation.cfm?id=1889889 retrieved June 3 2015

[8] Real-time simulation of watery paint by Tom Van Laerhoven, Frank Van Reeth 2005 http://www.heathershrewsbury.com/dreu2010/wp-content/uploads/2010/07/RealtimeSimulationOfWateryPaint.pdf retrieved June 3 2015

[9] GIMP by The GIMP Team http://www.gimp.org/ retrieved June 5 2015