Non-Photorealistic Rendering and Animation of Gaseous Phenomena

Dries HERBOTS

promotor :
Prof. dr. Frank VAN REETH

Universiteit Maastricht

Eindverhandeling voorgedragen tot het bekomen van de graad
Licentiaat in de informatica afstudeervariant multimedia

universiteit hasselt

Non-photorealistic rendering and animation of gaseous phenomena

Dries Herbots

Promotor: Prof. dr. Frank Van Reeth
Begeleider: dr. Fabian Di Fiore

Thesis voorgedragen tot het behalen van de graad van master in de informatica/ICT/kennistechnologie.

2005 – 2006

Abstract

'Gaseous phenomena' zijn alomtegenwoordig in het dagelijkse leven, en dus ook in animatie. Het zijn grillige en complexe fenomenen. Een animator wil een scène eenvoudig en naar eigen smaak kunnen manipuleren. Technische hulpmiddelen zijn dus wenselijk bij de animatie van dit soort fenomenen.

We bekijken technieken waarmee we dit kunnen doen voor 'stylized' animatie. Eerst zoeken we naar de eigenschappen van stylized animatie, welke factoren van belang zijn, en hoe die kunnen toegepast worden. Traditionele 2d animatie vormt daarbij een inspiratiebron. Controle is een belangrijk criterium, zowel voor de simulatie als voor de manier van renderen.

Een aantal simulatietechnieken voor fluïda wordt besproken. We onderscheiden fysische en niet-fysische methoden. Fluid solvers zijn moeilijk te controleren, maar vertonen wel realistisch gedrag. Andere technieken laten de animator vrij om zelf een pad te maken. Veel hangt af van het beoogde resultaat.

We beschrijven ook een aantal mogelijken rendertechnieken, waaronder 2.5d primitieven, billboards en texture advectie. Toon shading komt aan bod en kan op rook worden toegepast. We bespreken ook hoe men self-shadowing via shadow volumes kan implementeren. Vaak hangt de manier van renderen samen met welk soort simulatie er is gebruikt. Tot slot bevat dit eindwerk een bespreking van de implementatie.

Voorwoord

Ik wil iedereen bedanken die heeft geholpen bij het tot stand komen van dit eindwerk. Mijn promotor Prof. dr. Frank Van Reeth, die mij een boeiend onderwerp heeft aangereikt. Mijn begeleider dr. Fabian Di Fiore voor zijn creatieve input en praktische hulp tijdens het werk aan mijn thesis. De assistenten die mij hebben geholpen met mijn code. En natuurlijk mijn familie en vrienden voor hun geduld en onvoorwaardelijke steun.

Inhoudsopgave

Lijst van figuren

Hoofdstuk 1

Inleiding

Onder 'gaseous phenomena' verstaan we vormeloze substanties zoals rook, vuur of water. Dit soort fenomenen is niet weg te denken uit de onze dagelijkse leefomgeving. Ze gedragen zich zeer chaotisch en zijn nogal reactief, maar geven ook aanleiding tot prachtige visuele effecten. Het is dus niet verwonderlijk dat men die effecten wil nabootsen in computer graphics. Vaak denkt men daarbij aan een fotorealistische weergave, bijvoorbeeld voor de special effects in films, of voor computerspelletjes die een levensecht uitzicht nastreven. Vooral dit uizicht is belangrijk, je zou kunnen zeggen dat er een subjectieve norm is, nl. "ziet het er echt uit".

Niet-fotorealistisch renderen betekent dat het niet van belang is of het resultaat er echt uitziet. Er is niet echt sprake van een algemene norm, het beoogde uiterlijk hangt van de toepassing af. De toepassingen zijn uiteenlopend. In animatie streeft men soms een *cartoonachtig* uiterlijk na, waar een wetenschappelijke rendering vooral data wil verduidelijken.

In dit eindwerk bekijken we de mogelijkheden om gaseous phenomena te renderen op een niet-fotorealistische manier. We kijken vooral hoe we dit kunnen doen voor animatiedoeleinden. Zowel bij 2d als 3d animatie is er nood aan mogelijkheden om makkelijk gestileerde gaseous phenomena aan een scène toe te voegen. We geven een overzicht van de beschreven technieken om dit te doen. We proberen criteria te vinden die van belang zijn bij *stylized* animatie, en vergelijken de verschillende technieken.

De meeste technieken bestaan grofweg uit een *simulatie* - en een *visualisatie* stap. Die zijn vaak van elkaar afhankelijk, maar er zijn ook combinaties mogelijk. Zo maken bijvoorbeeld veel rendertechnieken gebruik van particles. Als de simulatie dus particle data genereert, kunnen die daarna op eender

welke manier gerendered worden. Simulatie en renderen bespreken we daarom in aparte delen. Deze thesis is als volgt georganiseerd:

In hoofdstuk 2 bekijken we de eigenschappen van traditionele 2d animatie. We zien welke criteria van belang zijn voor de animatoren, hoe bepaalde taken kunnen geautomatiseerd worden, en bespreken enkele concrete algoritmes. In hoofdstuk 3 bekijken we een aantal modellen om fluïda te simuleren en hoe ze geschikt zijn voor niet-fotorealistische animatietoepassingen. Hoofdstuk 4 bespreekt de manieren voor 'stylized' renderen. Hoofdstuk 5 gaat dieper in op de implementatie.

Hoofdstuk 2

Traditionele animatie en automatisering

Sinds hun ontstaan zijn tekenfilms almaar populairder geworden. Mede dankzij de eerste Disney cartoons uit de jaren 30 hebben ze een steeds groter wordend publiek weten te bereiken. Vandaag is de animatiefilm niet meer weg te denken uit het dagelijkse leven. Tekenfilms zijn niet enkel meer voor kinderen, maar komen in uiteenlopende vormen voor: reeksen voor volwassenen, in de reclame, langspeelfilms, videoclips, ...

Animatiefilms maken is echter erg arbeidsintensief. Geen wonder dus dat sinds zijn introductie de rol van de pc bij de ontwikkeling van animaties enkel is toegenomen. Tegenwoordig kunnen animaties volledig in 3d gemaakt worden dankzij ontwikkelingen in grafische hardware. Hoewel 3d animatie een subdomein van de animatiefilm is geworden, betekent dit zeker niet dat de 2d tekenfilm zal verdwijnen. 'Gewone' tekenfilms bieden de ontwerper nog altijd veel meer expressiemogelijkheden en creatieve controle. Dit komt omdat 3d animaties vaak te realistisch zijn. Veel effecten in zulke films zijn gegenereerd op basis van algoritmes die proberen de werkelijkheid te benaderen. Animatoren maken in traditionele tekenfilms vaak gebruik van overdrijvingen en abstracties, die niet realistisch zijn, maar wel herkenbaar. Dit is moeilijk met de computer te genereren.

Toch kan de computer ook een rol spelen in 2d animatiefilms. Bepaalde tijdsintensieve aspecten kunnen gedeeltelijk geautomatiseerd worden, soms worden ook 3d effecten gebruikt in 2d films. Het is echter belangrijk altijd een evenwicht te zoeken tussen controle voor de animator enerzijds, en automatisering anderzijds.

2.1 Traditionele animatie

Een traditionele animatiefilm is een reeks aaneengeschakelde tekeningen die snel achter elkaar worden getoond. Die beelden verschillen maar een heel klein beetje van elkaar. Omdat het menselijk oog slechts een eindig aantal beelden per tijdseenheid kan verwerken ontstaat de illusie van beweging en lijkt het voor de kijker alsof hij naar een getekende film kijkt. Al vanaf 10 frames per seconde ontstaan bewegende beelden, maar om flikkeringen te vermijden wordt een hogere framerate aangehouden. De meeste moderne films bevatten 24 beelden per seconde. Dit maakt van tekenfilms een zeer arbeidsintensieve kunstvorm. Omdat er vaak door verschillende tekenaars aan gewerkt wordt, moet opgelet worden dat proporties en volumes bewaard blijven doorheen de animatie, wat het productieproces niet vergemakkelijkt.

Om in handgetekende 2d tekenfilms een scène te maken gaat men als volgt te werk:

1. De hoofdtekenaars zetten de belangrijkste acties op papier: de begin- en eindfase en significante momenten. Dit noemt men de *extreme frames*.

2. Assistenten tekenen zogenaamde *key frames*: tussenframes die de timing van de scène bepalen en meer detail toevoegen.

3. Andere tekenaars mogen de resterende *in-between frames* tekenen.

Een van de meest tijdrovende bezigheden aan traditionele 2d animatie is het tekenen van die in-between frames. Maar in zo'n serie frames verschilt een bepaalde tekening meestal maar een klein beetje van zijn buren. Dit maakt het in-betweening proces interessant voor automatisering. De volgende sectie gaat over automatische in-betweening door gebruik te maken van 2.5d modelleren zoals beschreven in [1].

2.2 In-betweening met behulp van 2.5d modelling

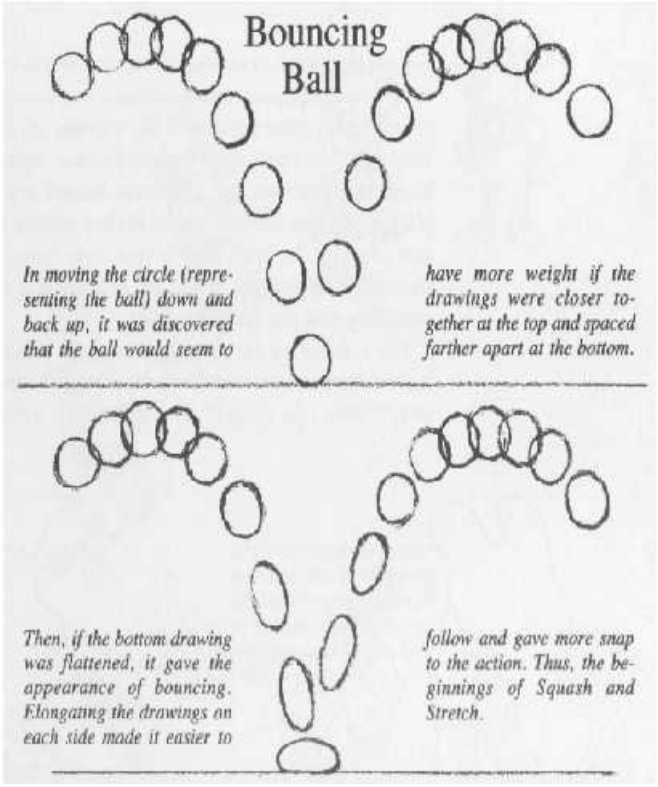
In 2d animatie zijn 2 categoriën transformaties te onderscheiden. Transformaties binnen een vlak evenwijdig met het tekenvlak, zoals translaties in dat vlak of rotaties rond de z-as. Deze transformaties vormen de eerste categorie en zijn niet zo'n probleem voor in-betweening. De 2d categorie bevat de transformaties die buiten zo'n vlak vallen, en die geven veel meer problemen. Dit komt omdat de tekenvolgorde van de verschillende delen van een object

kan veranderen in de loop van zo'n transformatie. Er is dan nood aan informatie over de ruimtelijke structuur van het object om de overgang in goede banen te leiden, maar die is niet aanwezig in 2d tekeningen. Bij traditionele animatie maakt de animator een 3d beeld in zijn hoofd.

Als de computer instaat voor het in-betweening proces moet die ook op de hoogte zijn van deze informatie. 2.5d modelling kan hiervoor zorgen. 2.5d objecten kan je beschouwen als 2d primitieven waaraan ruimtelijke informatie is toegevoegd. In [1] implementeert men 2.5d via een gelaagd systeem. Men werkt met volgende levels:

- *Level 0* bestaat uit de basisprimitieven waaruit een object is opgebouwd, in dit geval subdivision curves.
- *Level 1* zorgt voor de inpassing van 2.5d. Een object wordt gezien als een verzameling primitieven, geordend in functie van zijn rotaties rond de x -en y-as. Voor een aantal combinaties van (x,y) rotaties wordt een verzameling geordende primitieven getekend, zoals extreme frames in 2d animatie. Voor welke (x,y)'s dit gebeurt, is te kiezen door animator en hangt van de scène af (gewenste camerahoeken e.d.).

  Om tot een animatie te komen worden eerst key frames gespecificeerd op bepaalde tijdstippen om het tijdsverloop van de animatie te bepalen. Renderen van een (key of in-between) frame komt dan neer op het zoeken naar omliggende extreme frames, en de controlepunten van hun curves te interpoleren. Ook de tekenvolgorde wordt uit de extreme frames afgeleid.
- *Level 2* gebruikt 3-dimensionale skeletten om objecten te modelleren. Een skelet is een hiërarchische structuur van met elkaar verbonden beenderen. Vooral voor grote objecten met veel sub-objecten kunnen skeletten handig zijn. Elk sub-object dat onafhankelijk moet kunnen bewegen wordt aan een bot toegekend. De controlepunten van de primitieven worden nu gespecificeerd volgens een lokaal coördinatensysteem per bot. Het maken van key frames en toevoegen van transformaties kan nu gebeuren door oriëntatie van beenderen in de ruimte. De tekenvolgorde van de beenderen kan worden afgeleid uit hun positie.
- *Level 3* biedt de mogelijkheid om high level tools bovenop level 2 te gebruiken.



Figuur 2.1: Squash and stretch voor een botsende bal.

Deze implementatie van een 2.5d systeem is een voorbeeld van een aanpak die in-betweening kan automatiseren en wordt gebruikt bij andere technieken die zijn beschreven in het vervolg van deze thesis.

2.3 Eigenschappen van traditionele animatie toegepast op computer animatie

In de paper van John Lasseter [17] met dezelfde titel als deze sectie gaat hij dieper in op de karakteristieke eigenschappen van traditionele animatiefilms, en hoe ze van toepassing zijn op computer animatie.

Squash and stretch

Levende wezens en zachte objecten veranderen van vorm als ze bewegen. Enkel harde objecten behouden hun vorm. Zo kunnen bewegingen of botsingen

duidelijk gemaakt worden. In animatie wordt dit vaak zelfs overdreven, om stilistische redenen of voor extra nadruk (zie fig. 2.1). In computer animatie kan samendrukbaarheid of uitrekking (*squash and stretch*) bekomen worden door verschillende schalering tov de X-, Y- (-en Z) as. Belangrijk is wel dat het totale volume van een object behouden blijft.

Timing and motion

De timing van een actie of een beweging heeft invloed op hoe de kijker ze waarneemt. Door meer of minder in-between frames te gebruiken kunnen met dezelfde actie verschillende emoties worden uitgedrukt. Timing heeft ook invloed op de perceptie van de massa en de grootte van objecten.

Anticipation

Een actie bestaat uit 3 delen:

1. De voorbereiding (anticipation)
2. De actie zelf
3. De beëindiging van de actie

Een goed getimede voorbereiding kan de kijker helpen de actie beter te begrijpen. De voorbereiding kan ook massa suggereren.

Staging

Met de *staging* van een idee bedoelt men dat het zo gepresenteerd wordt dat zijn betekenis duidelijk is. Een idee kan een actie, personage, gemoed, etc. zijn. Belangrijk is dat de kijker naar de plaats van de actie wordt geleid, en dat er slechts 1 idee tegelijk voorkomt. Contrast met de rest van de scène wordt vaak gebruikt. Acties die in silhouet zijn getekend (langs de zijkant) komen veel duidelijker over.

Follow-trough and overlapping action

Follow trough is het laatste deel van een actie. Niet alle onderdelen van een actie stoppen op hetzelfde moment. *Overlapping* wil zeggen dan een volgende actie al begonnen is nog voor de huidige is gestopt. Dit houdt de interesse

van de kijker vast.

Straight ahead and pose-to-pose action

Bij *straight ahead action* begint de animator spontaan bij het eerste frame te tekenen en maakt frame na frame tot een actie is afgerond. *Pose-to-pose action* betekent dat een actie wordt gepland, en in stappen afgewerkt. Eerst enkele keyframes en daarna pas alle in-betweens. Bij computer animatie kunnen dan een groot aantal in-betweens automatisch gegenereerd worden. Design van keyframes kan ook vereenvoudigd worden door hiërarchische modellen.

Slow in and out

Door het aantal keyframes bij begin en einde te variëren, kan een beweging interessanter worden. Bij computer animation kan je dit doen door splines te gebruiken voor het pad van een object.

Exaggeration

Door goed gekozen overdrijvingen kan het uitzicht van een actie verfijnd worden.

Secondary action

Een actie als direct gevolg van een andere actie verhoogt de complexiteit van een scène. Ze mag wel niet concurreren met de primaire actie.

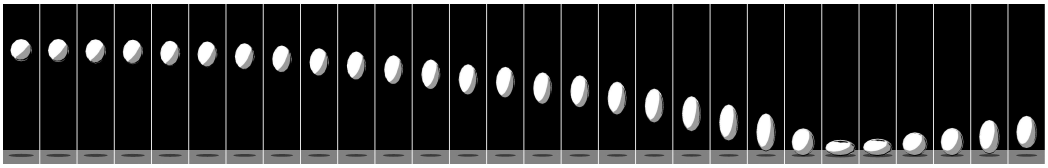
Appeal

De aantrekkingskracht en uitstraling van een scène of personage is uiteraard van belang. Asymmetrie is van nature interessanter.

Personality

De zonet besproken principes moeten gebruikt worden om de karakters in een animatie een aantrekkelijke en geloofwaardige persoonlijkheid te geven. De animator moet hierover een idee hebben voor hij begint te ontwerpen.

In theorie zou met al deze eigenschappen rekening gehouden moeten worden, wanneer animaties met de pc worden gemaakt. Dit is niet altijd mogelijk. Sommige principes lenen zich beter tot computer animatie dan andere.



Figuur 2.2: Squash and stretch effect op een procedurele, 3-dimensionale beweging, zie [11].

Een manier om squash and stretch in een procedureel model te implementeren wordt besproken in 'simulating cartoon style animation' [11]. Hierover gaat de volgende sectie:

2.4 Squash and stretch

Met de hand getekende cartoons hebben verschillende voordelen tegenover procedureel gegenereerde animaties. Zo kunnen tekenaars een abstractie maken van een beweging, rekening houdend met de perceptie van de kijker. Maar traditionele animatie is niet interactief en tijdrovend. Lasseter [17] wijst op een aantal belangrijke animatie principes. Maar het toevoegen van deze principes aan procedurele modellen stelt problemen: In animatie wordt emotie verwerkt in de bewegingen (*characterization*), en er is een verband tussen de manier van renderen en de animatie-techniek. Rekening houdend met dit alles probeert men in [11] 1 animatie principe toe te passen op computer animatie, namelijk *squash and stretch*. Volgens Lasseter anticipeert en benadrukt *squash and stretch* een botsing tussen objecten.

Dit principe is al op verschillende manieren geïmplementeerd:

- Squash and stretch kan je aan **physically-based modelling** toevoegen, maar voor stretch zijn artificiële krachten nodig. Dit is een beetje contradictoir als je van physically based modelling uitgaat.
- Technieken gebaseerd op **implicit surfaces**. Tot nu toe enkel methodes die de vorm van het object bepalen, en niet de gehele beweging.
- Bij **time-warping** worden verschillende delen van een object op andere tijdstippen geplaatst, waardoor stretch ontstaat als het hele object een beweging maakt tijdens een bepaald tijdsinterval. Het is nog niet duidelijk hoe op deze manier botsingen kunnen gesimuleerd worden.

In [11] implementeert men een model dat de bewegingen en botsingen van objecten bepaalt. De gebruiker kan de visuele output manipuleren aan de hand van een aantal parameters:

- Gravitatie g, globale parameter.
- Restitutie r bepaalt de energie die een object verliest bij een botsing en wordt per object ingesteld.
- Maximale stretch s_{max} per object.
- Minimum squash s_{min} per object.
- Stretch rate k_{str} per object.
- Squash rate k_{sq} per object.

In de simulatie bevindt elk object zich steeds in één van deze twee toestanden, namelijk *free space mode* waarbij objecten bewegen als puntmassa's, en gravitatie en snelheid hun vorm bepalen, of *collision mode*, wanneer squash en stretch regels vorm en gedrag bepalen. Elke tijdstap wordt de simulatie als volgt ge-update:

1. Update alle free-space objecten tot aan het volgende tijdstip of tot aan een eerder optredende botsing.
2. Bereken voor elke nieuwe botsing de *collision parameters*.
3. Update alle objecten die bij botsingen betrokken zijn.

Vervormingen (deformations)

Elk object heeft zijn eigen assenstelsel, en vervorming gebeurt ten opzichte van deze assen. De x-as moet steeds wijzen in de richting waarin het object zich voortbeweegt. Gesteld dat vervorming het volume moet bewaren en volgens 1 parameter s wordt gestuurd, resulteert dat in volgende schaleringsmatrix:

$$\begin{pmatrix} s & 0 & 0 \\ 0 & \sqrt{1/s} & 0 \\ 0 & 0 & \sqrt{1/s} \end{pmatrix}$$

Beweging en vervorming in free-space mode

Een object moet zich als een puntmassa volgens ballistische regels bewegen. En hoe sneller een object beweegt, hoe meer het uitgerokken moet worden. Dit laatste geeft volgende uitdrukking voor s:

$$s = \frac{k_{str}\|v\|s_{max} + 1}{k_{str}\|v\| + 1}$$

Dit resulteert in volgende update-routine voor objecten in de vrije ruimte: translateer het object naar zijn nieuwe positie, roteer het zodat de x-as van zijn assenstelsel samenvalt met de bewegingsrichting, en vervorm het volgens s.

Beweging en vervorming in collision mode

Tijdens een botsing moet de vervorming *smooth* evolueren, en als een object overgaat van free-space naar collision mode moet de vervorming continu zijn. De beweging en vervorming van een object in collision mode zijn enkel afhankelijk van een interpolatie van de beginvoorwaarden, zijnde v_{in}, s_{in} en de normale n. De snelheidsvector wordt bij het verlaten van de botsing is v_{in} gereflecteerd om n, en verminderd met een stukje restitutie r. Men kiest voor sinusoïdaal interpoleren tijdens de collision.

Een botsing bestaat uit een squash en een stretch fase. Tijdens de squash fase $t = 0 \rightarrow t_{mid}$ geldt het volgende:

$$s = s_{in} - (s_{in} - s_{mid}) \sin \omega_{in} t$$

$$s_{mid} = 1 - (1 - s_{min}) \frac{s_{in}}{s_{max}}$$

$$\omega_{in} = \frac{\|v_{in}\|}{l(s_{in} - s_{mid})}$$

$$t_{mid} = \frac{\pi}{2\omega_{in}}$$

Waarbij l de afstand is tussen het punt van het object dat het verst is verwijderd van het botsingspunt, en het botsingspunt zelf. De stretch fase $t_{mid} \rightarrow t_{out}$ verloopt analoog:

$$s = s_{out} - (s_{out} - s_{mid}) \cos \omega_{out} t$$

Tijdens de collision wordt een object geroteerd volgens een hoek die het resultaat is van interpolatie tussen de inkomende -en uitgaande richtingsvector.

2.5 Conclusie

Hoewel traditionele animatie op expressief gebied nog steeds veel krachtiger is dan computer gegenereerde animatie, kan de computer toch bij meer en meer onderdelen van het animatieproces helpen. We hebben enkele technieken gezien die wel degelijk het werk verlichten zonder daarbij de karakteristieke eigenschappen van 2d animatie te verwaarlozen.

Een animatie heeft niet enkel behoefte aan personages en actie, maar ook aan een setting waar de acties plaatsvinden. Een belangrijk onderdeel van de setting is het voorkomen van natuurlijke fenomenen, zoals water, vuur, rook, etc. Deze vertonen vaak complex gedrag, dat niet zo gemakkelijk met de hand te tekenen is. Aan de andere kant willen designers vaak geen realistisch uitziende fenomenen, maar een gestileerde, abstractere vorm van het fenomeen weergeven. De rest van deze thesis bespreekt een aantal technieken om gaseous phenomena aan animaties toe te voegen. Aandachtspunten hierbij zijn:

- Controle voor de animator; heeft hij genoeg creatieve vrijheid zowel over de simulatie als renderstijl?
- Frame to frame coherentie; op welke wijze kan dit gewaarborgd worden?
- Gebruiksgemak -en snelheid; een intuïtieve interface is belangrijk om zulke complexe fenomeen efficiënt te kunnen manipuleren.

Hoofdstuk 3

Modelleren van gaseous phenomena

Alvorens een fenomeen te renderen is het vaak handig om het te simuleren. De keuze die hierin gemaakt wordt beïnvloedt niet alleen het uitzicht van de geproduceerde animaties, maar ook het animatieproces zelf. Dat het model het gedrag van de uiteindelijke animatie bepaalt, is nogal logisch. Maar er zijn nog een aantal andere factoren die bij de keuze voor een model meespelen. Controle over de gehele animatie en het gedrag van het fenomeen, gebruiksgemak bij het ontwerpen van een specifieke animatie, snelheid van werken etc., zijn belangrijke aandachtspunten.

Grofweg kunnen we 2 methoden onderscheiden. Technieken die gebaseerd zijn op wetten uit de fysica en op die manier een realistisch model nastreven, *fysische methoden*. Andere technieken houden weinig of geen rekening met fysische eigenschappen bij de simulatie, of leggen meer nadruk op controle. Deze technieken hebben we onder *niet-fysische methoden* ondergebracht.

Particle systems

Particle systems worden al sinds 1983 [15] gebruikt voor het modelleren van talrijke amorfe fenomenen. Vanwege hun eenvoud en flexibiliteit zijn ze nog steeds heel populair. Ze worden hier apart besproken omdat ze in beide categoriën kunnen gebruikt worden. Meestal worden ze ook in een aangepaste vorm gecombineerd met andere technieken.

Bij een particle system wordt een amorf fenomeen gemodelleerd als een verzameling particles. Dit betekent dat het model niet-deterministisch en stochastisch van aard is. Elk particle heeft een aantal attributen. Positie en

snelheid zijn de belangrijkste. Een systeem maakt gebruik van een aantal krachten die de particles voortbewegen. Elke tijdstap berekent men voor elk particle aan welke krachten het onderhevig is en hoe die zijn snelheid veranderen. Met die nieuwe snelheid wordt een nieuwe positie bepaald. Verder hebben particles nog een aantal andere attributen, afhankelijk van het systeem en de simulatie. Dit zijn eigenschappen als grootte, kleur, leeftijd, etc. Nieuwe particles worden in het systeem geïntroduceerd op plaatsen die men bronnen noemt. Soms bevinden er zich al een aantal particles in het systeem in zijn begintoestand. Ook worden sommige particles verwijderd uit het systeem, op basis van bepaalde criteria (bv. ouderdom, te ver verwijderd, ...). In een klassiek particle system is er geen interactie tussen de particles onderling. Een recente beschrijving van een manier om rook te modelleren voor real-time toepassingen met particle systems staat in [3]. Het beschrijft verschillende krachten waaronder wind, drijfkracht en turbulentie, en introduceert draaikolken via onzichtbare particles.

Een *second order particle system* [16] is een uitbreiding op de traditionele variant. Hier zijn de krachten en particle bronnen niet statisch, maar speciale particles. Zij evolueren dus ook mee met het systeem, wat resulteert in een minder artificiële simulatie en meer mogelijkheden biedt. Nog een groot verschil met een gewoon particle system, is dat particles toegang moeten hebben tot andere particles. Zo kunnen bv. naburige particles het gedrag van een bron particle beïnvloeden. Men maakt een onderscheid tussen *force class*; de manier waarop de kracht zich gedraagt (bv. als draaikolk, wind, ...); en *force categorie*, die het soort kracht bepaalt (elektrisch, mechanisch, ...). Die onderverdeling in categoriën moet de animatie eenvoudiger te controleren maken. Ilmonen en Kontkanen geven in hun paper [16] een aantal voorbeelden van fenomenen die ze met een second order system hebben gesimuleerd, waaronder rook en vuur. Ze lieten behalve rook particles ook kleine en grote draaikolk (force) particles in de simulatie los.

3.1 Fysische methoden

In de fysica gebruikt men onder meer de *Navier-Stokes vergelijkingen* om het gedrag van *fluïda* te beschrijven. Het probleem is dat dit complexe differentiaalvergelijkingen zijn die enkel in een aantal eenvoudige gevallen analytisch op te lossen zijn. Voor grafische toepassingen gebruikt men relatief ongedetailleerde numerieke benaderingen. Dit omdat men rekening moet houden met het geheugen en de snelheid van de pc, en omdat de rook er niet noodzakelijk minder realistisch gaat uitzien. Foster en Metaxas [20]

[21] produceerden in 96-97 mooie resultaten met hun benadering van Navier-Stokes, die ze gebruikten om water en rook te renderen. Stam [18] [19] baseerde zich op hun werk om een stabielere en snellere solver te maken, die bovendien real-time werkte. Samen met Fedkew en Jensen ontwikkelt hij in 2001 een meer gedetailleerde solver [13]. Afhankelijk van het beoogde effect kan met 2d of 3d solvers gewerkt worden. Witting [2] gebruikt een 2d model. Het voordeel van een 3d simulatie is dat er diepte informatie is over de rook, waar bij het renderen rekening mee gehouden kan worden. Omdat fluid solvers in 3 dimensies veel meer geheugen vragen worden grote simulaties in 3d bemoeilijkt. Om dit op te lossen stelt men in [12] voor enkele 2d grids te combineren met een kolmogorov veld. Zo kunnen grote 3 dimensionale effecten voordeliger gesimuleerd worden.

Witting [2] beschrijft een animatie systeem voor speciale effecten met behulp van fluid dynamics, dat is ontwikkeld voor de tekenfilm 'The prince of Egypt'. Bij dit systeem vertrekt men van een afwijkende formulering van Navier-Stokes, die uit de meteorologie stamt.

Onderstaand volgt een samenvatting van de fluid solver van Stam [18] [19], omdat het een goed voorbeeld is van een korte solver, geschikt voor animatie doeleinden, en hij ook gebruikt wordt in de implementatie (zie hoodstuk 5).

3.1.1 De fluid solver van Stam

De *Navier-Stokes vergelijkingen* zijn een mathematisch model voor veel soorten fluid flows. De toestand van een fluïdum op een bepaald moment in de tijd wordt voorgesteld door een vectorveld. Dit veld geeft op elk punt in de ruimte een vector die de snelheid van het fluïdum op dat punt weergeeft. Navier-Stokes is een verzamelnaam voor een aantal differentiaalvergelijkingen die de evolutie van dit snelheidsveld over de tijd beschrijven. Onderstaand: Navier-Stokes voor een snelheidsveld $\mathbf{u}$ (3.1).

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} + \nu \nabla^2 \mathbf{u} + \mathbf{f} \tag{3.1}$$

Om visueel effect te bekomen moet er ook nog iets zichtbaar door dat snelheidsveld bewegen. Voorwerpen kunnen gewoon als particles het snelheidsveld volgen. Een fluïdum zou je kunnen voorstellen door een grote verzameling deeltjes, maar dit zou te veel geheugen vergen. Daarom wordt het in zijn geheel voorgesteld als een dichtheidsveld. Dat veld geeft voor elk punt in de ruimte de dichtheid van het fluïdum op die locatie. De evolutie van dit veld ρ door een (statisch) snelheidsveld $\mathbf{u}$ wordt beschreven door

onderstaande vergelijking (3.2):

$$\frac{\partial \rho}{\partial t} = -(\mathbf{u} \cdot \nabla)\rho + \kappa \nabla^2 \rho + S \tag{3.2}$$

Zoals reeds opgemerkt zijn de Navier-Stokes vergelijkingen te ingewikkeld om analytisch op te lossen. Een fysisch correcte numerieke benadering is niet mogelijk in real-time. Voor real-time toepassingen is het belangrijker dat de benadering stabiel en snel is, en geen opvallend onrealistisch gedrag vertoont.

Dichthedenveld

(3.1) en (3.2) lijken sterk op mekaar, maar (3.2) is makkelijker te benaderen omdat ze lineair is. De voor (3.2) gepresenteerde oplossing wordt daarna naar (3.1) uitgebreid. Allereerst wordt de ruimte gediscretiseerd door ze in cellen op te delen. Het fluïdum wordt gesampled in het midden van deze cellen. De dichtheid van het fluïdum en de snelheid zijn constant in elke cel. Om grensgevallen te vergemakkelijken wordt een extra rij cellen rond de ruimte gezet. Men neemt aan dat de tijdstap dt constant is gedurende de simulatie.

Het rechterlid van vgl. (3.2) bestaat uit 3 termen. Om (3.2) te benaderen worden deze termen in omgekeerde volgorde behandeld. De laatste term zegt dat de dichtheid kan toenemen door externe bronnen. Dit kan in rekening worden gebracht door deze bronnen gewoon bij het dichtheidsveld op te tellen.

De 2de term beschrijft de verspreiding van het fluïdum door de ruimte, de diffusie. κ stelt de snelheid voor waarmee dit gebeurt. In de grid komt het erop neer dat elke cel dichtheid uitwisselt met zijn buren. Zij $x0$ het huidige dichtheidsveld, x het nieuw te berekenen veld en a een aangepaste diffusiefactor. Een naïeve implementatie voor de diffusie zou zijn:

```
Doe voor elke combinatie (i,j) (en dus voor elke cel)
x[i,j] = x0[i,j] + a*( x0[i-1,j] + x0[i+1,j] + x0[i,j-1] +
                       x0[i,j+1] - 4*x0[i,j] )
```

Voor grote waarden van a echter kan de dichtheid beginnen oscilleren en uiteindelijk divergeren, wat de simulatie onstabiel maakt. Om dit te vermijden volgt men een omgekeerde redenering. Stel dat je zoekt naar de dichtheden x[i,j] die, als je zou terug diffunderen in de tijd, de begindichtheden x0[i,j] zouden opleveren. Dit kan als volgt uitgedrukt:

```
x0[i,j] = x[i,j] - a*( x[i-1,j] + x[i+1,j] + x[i,j-1] + x[i,j+1] -
                                                    4*x[i,j] )
```

Dit is een lineair stelsel in x[i,j]. Om dit stelsel op te lossen gebruikt men een numerieke methode, *Gauss-Seidel relaxation*. Merk op dat ook andere benaderingen mogelijk zijn, zo lang het stelsel maar wordt opgelost. Zo is *conjugate gradiënt* ook populair [32]. De x[i,j]'s worden zo achterhaald zonder stabiliteitsproblemen, onafhankelijk van de waarde voor a.

De term $-(\mathbf{u} \cdot \nabla)\rho$ drukt uit dat de dichtheden het snelheidsveld moeten volgen. Om deze advectiestap te berekenen beschouwt men het midden van elke grid cel als een particle. Er wordt gezocht naar de posities van deze particles, één tijdstap geleden. Anders gezegd: de plaatsen waar de particles zich dt tijd geleden moesten bevinden om nu in het midden van de grid cellen terecht te komen. De dichtheid in een cel wordt nu bekomen door te kijken naar zijn 'vorige positie' en de dichtheden van de 4 dichtstbijzijnde cellen (lineair) te interpoleren.

Snelheidsveld

Vergelijking (3.1) kan op een bijna identieke manier opgelost worden als (3.2). De termen uit het rechterlid worden in omgekeerde volgorde berekend, volgens de methoden die bij (3.2) zijn gebruikt. Er moet alleen een extra routine worden toegevoegd na de diffusie -en advectie stap. Het snelheidsveld moet namelijk behoud van massa garanderen. Om dit te bereiken gebruikt men *hodge decompositie*: elk snelheidsveld is de som van een veld dat de massa bewaart en een gradiëntenveld. Om dit massa behoudende veld te vinden kan je dus het gradiëntenveld van het originele aftrekken. Dit gradiëntenveld achterhalen komt neer op het oplossen van een stelsel, een *poisson* vergelijking. Deze stap heet projectiestap omdat het veld als het ware op zijn massa behoudende component wordt geprojecteerd. Nu moet enkel nog aan de grensvoorwaarden gedacht worden en de fluid solver is klaar.

3.1.2 Visuele Resultaten

Fluïda die met behulp van solvers gesimuleerd worden vertonen erg realistisch gedrag, met verschijnselen zoals kleine draaikolken en complexe interactie met objecten. Ze zijn uitermate geschikt voor animaties waarbij fysische nauwkeurigheid belangrijk is, wat betreft de waarneming ten minste. Daar staat tegenover dat dit realisme niet te vermijden is. Dit is nochtans de

bedoeling van sommige animaties, om een of ander effect te bereiken. Als expliciet afwijkend gedrag wordt nagestreefd zal dit met fluid solvers moeilijk te realiseren zijn.

In het systeem van Witting [2] kan een simulatie onafhankelijk van het renderen worden opgeslagen (*deferred rendering*). Later kunnen dan verschillende stijlen geprobeerd worden. Dit is een voordeel als men wil experimenteren en resulteert na verloop van tijd in een collectie nuttige simulaties, geschikt voor hergebruik.

3.1.3 Controle

Als het gaat over controle, toch een belangrijk criterium bij animatie, zijn fluid solvers absoluut niet flexibel. Enkel door het toevoegen van krachten, en het instellen van enkele algemene parameters (viscositeit, diffusie, etc.) kan de animator invloed uitoefenen op de simulatie. Voor de rest wordt de simulatie gestuurd door de solver. Het probleem is dat fluïda en dus ook hun modellen uiterst reactief zijn. Ze kunnen heel heftig reageren op een verandering in de omgeving. Het toevoegen van een relatief kleine kracht kan zo het hele systeem beïnvloeden en een totaal andere animatie opleveren. Deze onvoorspelbaarheid maakt het moeilijk voor de gebruiker om de gevolgen van zijn acties in te schatten, laat staan dat hij de gehele animatie in de gewenste richting zou kunnen leiden. Meestal moet men zich beperken tot het instellen van enkele beginvoorwaarden en daarna nemen de fysische wetten de animatie over. Het verkrijgen van de gewenste simulatie is op deze manier vaak een kwestie van trial-and-error en dus tijdrovend.

In het systeem uit [2] kunnen foto's of animaties als input voor de solver gebruikt worden. Met behulp van een aantal drempelwaarden en andere variabelen kan de gebruiker via de user interface bepalen hoe de input wordt geïnterpreteerd. Zo worden de beginwaarden voor een snelheids -en temperatuurveld ingesteld. Het systeem biedt op die manier meer controlemogelijkheden voor het opstellen van simulatievoorwaarden. Animaties en foto's zijn voor een animator ook een intuïtievere manier om dit te doen. Het probleem met deze werkwijze is dat de uitgebreide controle zich beperkt tot het creëren van de beginvoorwaarden, waarna men opnieuw aan de fluid solver is overgeleverd. In hetzelfde systeem kunnen simulaties op een efficiënte manier opgeslagen worden. Later kunnen ze hergebruikt worden. Er zijn mogelijkheden voor herschalering, instellen van ander perspectief, .. wat inpassen in andere scènes vereenvoudigt. Deze eigenschap verbetert controle en vermindert de werkhoeveelheid.

Keyframe controle

De controlemogelijkheden over fluid simulaties worden uitgebreid met de methode uit [7]. De gebruiker kan door middel van keyframes laten weten hoe hij de simulatie wil laten evolueren. Het systeem probeert dan, gebruik makend van een aantal krachten, de simulatie zo te optimaliseren dat de keyframes zo goed mogelijk benaderd worden.

Als uitgangspunt gebruikt men de solver van Stam. Een toestand $\mathbf{q}$ van de solver bestaat uit een grid van densities ρ en een snelheidsgrid $\mathbf{v}$. Een toestand $\mathbf{q_t}$ op een tijdstip t wordt berekend uit vorige toestand: $\mathbf{q_t} = \mathbf{S}(\mathbf{q_{t-1}})$. De simulatie L is dan gewoon een opeenvolging van toestanden: $L(\mathbf{q_0}) = (\mathbf{q_1}, ..., \mathbf{q_n})$.

De animator kan keyframes definiëren, zowel density keyframes ρ_t^* als snelheids keyframes $\mathbf{v_t^*}$. Hiermee kan hij aanduiden hoe de densities op een tijdstip t moeten verdeeld zijn, of hoe het snelheidsveld er dan moet uitzien. Het systeem zal de wensen van de animator zo goed mogelijk proberen te benaderen. Om de simulatie te kunnen beïnvloeden zijn krachten nodig. De controle parameters hiervan worden niet ingevuld, maar in een vector $\mathbf{u}$ gestopt. Het komt er dan op aan een waarde voor $\mathbf{u}$ te vinden die aan de wensen van de animator voldoet.

Om dit te bereiken construeert men een objectieve functie φ, die uitdrukt hoe goed de simulatie aan de gevraagde eisen voldoet. Het probleem is dan herleid tot het minimaliseren van φ voor $\mathbf{u}$. φ is de som van 2 functies: $\varphi = \varphi_k + \varphi_s$. Daarbij is $\varphi_k(L(\mathbf{u}, \mathbf{q_o}))$ het verschil tussen de simulatie en de keyframes. $\varphi_s(\mathbf{u})$ meet de hoeveelheid kracht die is toegevoegd.

Om φ te kunnen minimaliseren gebruikt men een gradiënt-gebaseerde techniek waarvoor we dus $\frac{d\varphi}{d\mathbf{u}}$ nodig hebben. $\frac{d\varphi}{d\mathbf{u}} = \frac{d\varphi_s}{d\mathbf{u}} + \frac{d\varphi_k}{d\mathbf{u}}$. φ_s hangt enkel van de vector $\mathbf{u}$ af, en is dus makkelijk af te leiden. φ_k is van de simulatie L afhankelijk, dus moet $\frac{dL}{du}$ berekend kunnen worden.

Om exacte afgeleiden te kunnen vinden voor L, maakt men een ruimte bestaande uit snelheidsveld, dichtheidsveld en hun afgeleiden. In deze ruimte laat men de simulatie plaatsvinden zodat tijdens elke stap t ook de afgeleiden meeberekend worden. Een toestand wordt dan als volgt uitgebreid:

$$\hat{q}_t = \left(q_t, \frac{dq_t}{du_1}, ..., \frac{dq_t}{du_c} \right)$$

waarbij nieuwe toestanden worden berekend aan de hand van een nieuwe stapfunctie $\hat{S}$ vertrekkende bij begintoestand $\hat{q_0}$:

$$\hat{q_t} = \hat{S}(\hat{q}_{t-1})$$

$$\hat{q_0} = (q_0, 0, ..., 0)$$

$\hat{S}$ definiëren we aan de hand van S:

$$S = M \circ A_\rho \circ P \circ A_V \circ F$$

Elk van deze onderdelen van S breiden we uit tot een functie die niet enkel de nieuwe velden maar ook de nieuwe afgeleiden berekent. bv:

$$\hat{M} = \left(M, \frac{dM}{du_1}, ..., \frac{dM}{du_c} \right)$$

zodat:

$$\hat{S} = \hat{M} \circ \hat{A}_\rho \circ \hat{P} \circ \hat{A}_V \circ \hat{F}$$

Minimalisatie

Uiteindelijk zal keyframe controle gerealiseerd moeten worden door φ te minimaliseren voor controlevector $\mathbf{u}$. Dit kan op verschillende manieren, in de paper gebruikt men een quasi-Newton methode die φ en zijn gradiënt $\frac{d\varphi}{du}$ moet kunnen evalueren voor elke $\mathbf{u}$. Dit wordt gedaan door zowel φ als zijn afgeleiden in elke stap van de simulatie mee te berekenen.

Dat de animator op deze manier keyframes kan definiëren biedt al heel wat meer mogelijkheden voor animatie met fluid solvers. De controle beperkt zich wel nog altijd tot het geven van keyframes; de simulatie zal de keyframes (min of meer) passeren, daarbuiten is er niet al te veel plaats voor controle. Het systeem heeft enkele beperkingen; het is nog niet geschikt voor grote simulaties, en de optimalisatie kan soms vastlopen in lokale minima, waardoor bijvoorbeeld een extra keyframe moet toegevoegd worden. Toch is het al een opmerkelijke vooruitgang voor animatie dat er controle is over de simulatie. Zeker omdat die uitgeoefend wordt met keyframes, bijna zoals in traditionele animatie.

3.2 Niet-fysische methoden

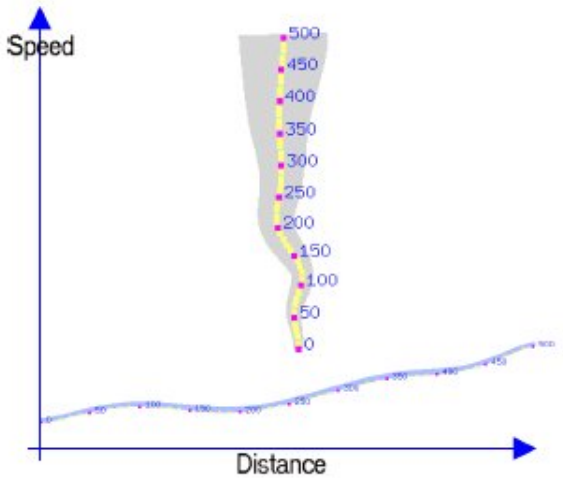
Onder niet-fysische simulaties verstaan we systemen die geen formules uit de fysica proberen te benaderen. Ze maken wel gebruik van een verzameling (versimpelde) regels om het model te updaten. De gebruikte regels hangen af van het beoogde resultaat. Zo kan beperkt realisme worden nagestreefd, of meer controle en creatieve input voor de gebruiker. Particle systems zijn veel voorkomende modellen, waarbij de update-regels het gedrag van de particles bepalen.

Een erg flexibel model gebruikt men in [4]. Men gebruikt een aangepaste versie van een second order particle system, in combinatie met 2.5D [1] technieken, om de rook te modelleren. De gebruiker speelt een hele grote rol in het bepalen van het gedrag van de simulatie. De animator moet zelf het pad tekenen dat het fluïdum dient te volgen. Hij kan een pad vanuit verschillende hoeken tekenen zodat een in-between pad kan berekend worden en 2.5d animatie mogelijk wordt. Zie figuur 3.1 voor een voorbeeld van zulke getekende input. De 2D paden worden door het systeem tot 3D paden omgezet om zo een 3 dimensionale simulatie te bekomen. Ook is de mogelijkheid voorzien om het pad tijdsafhankelijk te maken. De gebruiker tekent dan voor verschillende tijdstippen andere paden. Op tussenliggende tijdstippen wordt een in-between pad gegenereerd. Op die manier blijft het pad niet statisch in de tijd en vergroten dus de animatie mogelijkheden. Bij elk pad kan een acceleratie curve worden gevoegd, die de versnelling van deeltjes op verschillende plaatsen in het pad bepaalt.

De manier waarop dit aan de gebruiker wordt aangeboden sluit zeer nauw aan bij zowel het systeem voor automatische in-betweening en de 2.5d methode uit [1], als bij de manier waarop in traditionele animatie met extreme frames en in-betweening wordt omgesprongen. De animator kan op een voor hem bekende manier, namelijk al tekenend, bepalen hoe het fluïdum zich hoort te gedragen.

3.2.1 Visuele resultaten

De visuele output van niet-fysische modellen is veel meer divers, omdat de te gehoorzamen wetten niet vaststaan. De rook zal zich in sommige gevallen nog altijd min of meer realistisch gedragen, wanneer men de regels zo heeft opgesteld dat ze een reële situatie nabootsen. Merk op dat dit soort realisme moeilijk de vergelijking met fluid solvers kan doorstaan, het gaat eerder om gedrag dat realistisch overkomt. Met deze modellen is het ook mogelijk om



Figuur 3.1: Tekenen van een pad en acceleratie curve voor rook (zie [4]).

creatiever met de natuurwetten om te gaan en de animatie een zelfgekozen gedrag op te leggen. De manier waarop het model het uitzicht van de animatie bepaalt is veel meer afhankelijk van het concrete systeem en de gebruikte regels.

3.2.2 Controle

Qua controle voor de animator bieden dit soort modellen meer mogelijkheden. Omdat men niet noodzakelijk fysisch realisme nastreeft, en zelf de wetten bepaalt volgens dewelke de animatie zich gedraagt. Sommige modellen zoals in [4] bieden de designer zelfs een framework waarbinnen hij zelf in grote mate de simulatie kan sturen. Dit biedt meer mogelijkheden voor animatie omdat het gedrag van de rook op die manier kan aangepast worden aan de context van de animatie.

3.3 Conclusie

Zowel fysische als niet-fysische modellen worden gebruikt bij animaties. Welk model nu 'beter' is valt moeilijk objectief te bepalen, het is afhankelijk van de smaak van de uiteindelijke gebruikers. Voor grote projecten worden de modellen vaak specifiek aangepast aan de specifieke noden van de animatie en zijn ontwikkelingsomgeving, zoals in [2] voor de tekenfilm "Prince of Egypt". We vatten nog even de belangrijkste punten van beide soorten mo-

dellen samen:

fysische modellen

- Realistisch gedrag door gebruik fysica - Navier-Stokes vgl.
- Fluid solvers
- Moeilijk in te schatten gedrag
- Vergen veel geheugen, zeker 3 dimensionaal

niet-fysische modellen

- Meer controle over de simulatie mogelijk
- Realistisch gedrag moeilijk na te bootsen
- Flexibeler, makkelijker aan te passen aan concrete animatie

Hoofdstuk 4

'Stylized' renderen van fluïda

In het non-photorealistic rendering domein is er al heel wat research gedaan naar de visualisatie van fluïda. Die heeft zich vooral toegespitst op wetenschappelijke renderingen. Die hebben meestal als doel het begrip vergroten over het fenomeen, dus de visualisatie moet zoveel mogelijk details overzichtelijk weergeven. Dit soort visualisaties bestudeert bijvoorbeeld hoe verschillende 2d velden, bv. een snelheids -en dichthedenveld, samen tot één enkele 2d visualisatie kunnen gerendered worden, zodat de data van de velden duidelijk overkomt. In [8], [14] en [10] kijkt men naar schilderkunst en probeert men daaruit te leren over de manier waarop mensen waarnemen. Die concepten worden dan gebruikt bij renderen van weermodellen e.d.

Waar wetenschappelijke renderingen net details willen verduidelijken door ze te versterken, is dit bij animatie net het omgekeerde. Uit artistieke overwegingen worden fluïda veel simpeler voorgesteld dan ze in werkelijkheid zijn. Enkel op gerichte plaatsen laat men detail toe, zonder dat realisme een belangrijk criterium is. Men mikt meer op een 'realistisch uitzicht', d.w.z. de kijker moet het fenomeen herkennen.

Cartoon renderen van 3d scènes kent een groeiende populariteit, omdat de hardware krachtiger wordt en toelaat dit real-time te doen. Voor spelletjes is dit belangrijk en het cartoonachtige uitzicht zorgt voor een aparte look. (zie figuur 4.1). Enkel 3d scènes niet foto-realistisch renderen ziet er meestal 'te echt' uit voor traditionele 2d animatie doeleinden. Een combinatie met meer 2d-georiënteerde methodes werkt soms beter.

Vaak worden particle systems gebruikt. 3d particles kunnen voor de nodige diepte-informatie zorgen. In [6] en [4] worden primitieven aan particles toegekend om een stylized rendering te verkrijgen. In [23] combineert men

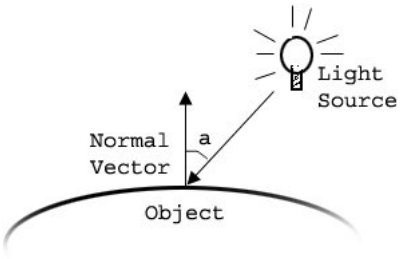
Figuur 4.1: Het spel XIII, gebaseerd op de gelijknamige stripreeks. Men heeft geprobeerd een 3d shooter op een strip-achtige manier te renderen, met veel 2d elementen, cartoon shading, e.d.

particles met toon shading rendering methodes en schaduwtechnieken.

Een andere aanpak voor stylized renderen is de advectie van een door de animator gemaakt texture door het fluïdum. Het texture wordt door een fluid solver vervormd en kan zo voor mooie effecten zorgen. Onder meer Witting [2] en Neyret [5] beschrijven hoe je dit kan doen. In het deel 'overige technieken' tenslotte vatten we nog 2 aparte manieren samen die Witting voorstelt om een fluid simulatie te renderen.

4.1 Toon shading

Strips en tekenfilms worden getekend in een aparte, niet foto-realistische stijl. Silhouetten worden expliciet getekend en outlines benadrukken de vorm van de objecten. Ze worden ingekleurd met slechts enkele kleuren. De overgangen tussen de kleuren zijn ruw; er zijn duidelijke grenzen zichtbaar. *Cartoon rendering* (ook wel *cell rendering* genoemd) beoogt het bekomen van deze stijl bij het renderen van 3d meshes. Het inkleuren noem men wel eens *painting*, het zoeken van outlines en silhouetten *inking*, naar analogie met handgetekende animatie.



Figuur 4.2: De vectoren in kwestie. Merk op dat we voor L de vector van de vertex naar de lichtbron nemen, en niet omgekeerd, zoals hier afgebeeld.

Bij cell shading maakt men gebruik van minstens 2 kleuren: een donkere kleur die schaduw simuleert, en een lichtere om belichting te suggereren. Om het probleem van cell shading op te lossen moeten de grenzen tussen die verschillende kleuren gevonden worden.

Een makkelijke manier om dit impliciet te doen is door een 1-dimensionaal texture te gebruiken ([27]). Een 1D texture gedraagt zich zoals een 2D texture met hoogte 1: er is dus maar 1 enkel texture coördinaat nodig. Als coördinaat bij een vertex wordt $\text{Max}[(L \cdot n), 0]$ gebruikt, waarbij n de normaal in de vertex is en L de genormaliseerde vector van de vertex naar de lichtbron. Merk op dat het scalair product $(L \cdot n)$ de cosinus geeft van de hoek tussen n en L (indien ze genormaliseerd zijn), de hoek a in figuur 4.2. Bij een hoek van 0 graden geeft dat coördinaat 1, een grotere hoek betekent een kleinere coördinaat. Negatieve cosinus waarden betekenen dat een vertex geen belichting ontvangt en worden in een coördinaat 0 omgezet. Vermits de graad van belichting stijgt met de stijgende cosinuswaarde van de hoek, moeten de texels van donker naar licht gerangschikt zijn in het texture.

Een gemakkelijke manier om dit te doen is gewoon de gewenste kleuren in het texture steken. Maar stel dat er al bepaalde materiaal- en lichteigenschappen zijn gedefinieerd in een scène die we willen toonshaden. In dit geval kan het texture gegenereerd worden door gebruik te maken van de volgende formule:

$$C_i = a_g \times a_m + a_l \times a_m + \text{Max}[(L \cdot n), 0] \times d_i \times d_m \tag{4.1}$$

Dit is de formule voor diffuse belichting in een vertex C_i met 1 lichtbron. a_m is de ambient factor van het materiaal, a_g de globale ambient belichting,

a_l de ambient component van de lichtbron, d_l en d_m de diffuse componenten van respectievelijk lichtbron en materiaal.

Door de kleur te berekenen zonder diffuse belichting (dus $\text{Max}[(L \cdot n), 0] = 0$) krijgt men de donkere kleur met enkel ambient licht. Door de diffuse component maximaal te nemen ($\text{Max}[(L \cdot n), 0] = 1$) berekent men kleur van het oppervlak bij maximale belichting.

De texture kan aangepast worden om extra effecten te bekomen. Door meer texels te gebruiken heeft men controle over de verhouding licht/donker. Als een derde kleur wordt toegevoegd kunnen zogenaamde *diffuse highlights* worden verkregen; in tegenstelling tot specular highlights zijn ze onafhankelijk van de camerapositie.

Eén van de problemen is dat er soms te veel aliasing optreedt. Dit komt voornamelijk omdat in de standaard grafische pipeline de kleur per vertex wordt berekend en dan geïnterpoleerd. In [26] wordt dit aangepakt door gerichte subdivision. Een driehoek wordt opgedeeld wanneer zijn 3 vertices niet dezelfde kleur hebben. Ook anti-aliasing en bepaalde texture filters kunnen tot mooiere resultaten leiden. Een andere oplossing kan per pixel cell shading zijn via een fragment program.

4.2 Particles renderen

4.2.1 Billboards

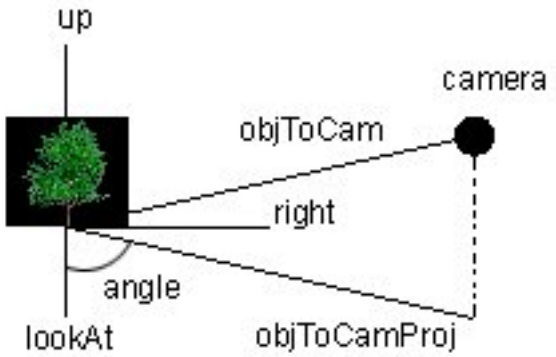
Technieken

Op de site van lighthouse3d staat een goede tutorial over billboarding ([30]). Hierop is deze korte uitleg gebaseerd.

Bij billboarding is het de bedoeling de oriëntatie van een 2d object te veranderen zodat het naar de camera gericht is. Hiervoor moet het geroteerd worden. Wat we moeten doen is dus de rotatie-assen zoeken en de hoeken voor de rotaties.

Men onderscheidt 2 soorten billboarding:

- Bij *cylindrical billboarding* mag een object slechts rond één welbepaalde vector geroteerd worden, meestal de up vector.
- Bij *spherical billboarding* zijn er geen restricties voor de rotatie.



Figuur 4.3: billboarding

Stel dat de oriëntatie van het object gekend is in de vorm van zijn up, right en lookat vector. Stel ook dat we zowel de positie van het object als die van de camera kennen. We kunnen dan gemakkelijk de vector van het object naar de camera vinden. Projecteer die naar hetzelfde vlak als het vlak waarin de lookat vector ligt (zie figuur 4.3), en zoek de hoek tussen deze vector en de lookat vector van het object. Als we roteren rond de up vector over de hoek die we net hebben gevonden hebben we cylindrical billboarding gerealiseerd.

Voor spherical billboarding volstaat het hier nog een stap aan toe te voegen, namelijk een rotatie rond de right vector. In figuur 4.3 zien we dat de rotatiehoek hiervoor gelijk is aan de hoek tussen de vector van het object naar de camera en zijn projectie, die al in de vorige stap was berekend.

Billboards gebruiken om rook te renderen

In [6] wil men fysisch realistische rook op een cartoon-achtige manier renderen. Als model voor de rook wordt een fluïd solver gebruikt, om realisme te garanderen. Hierin worden gewichtloze particles losgelaten die in de simulatie rondbewegen. Elk particle wordt als gerendered als een ge-billboarded *quad* (vierkant) met daarop een texture gemapt. Sommige texels kunnen een *alfa waarde* van 0 hebben. Hierdoor worden ze doorschijnend, en lijkt het resultaat op de figuur op de tekening. Deze techniek heet *alpha blending*. Afhankelijk van de alfa waarde wordt de kleur van de pixel berekend als een combinatie van de texture kleur en de achtergrond. Het texture stelt in dit geval een of andere primitieve voor, bv. een cirkel. Het renderen van de

particles als billboards zorgt voor de uiteindelijke animatie.

Men wil de beweging van de rook enkel op plaatsen met veel actie weergeven, door contouren uit te tekenen. Waar er niets gebeurt moeten zich grote, uniform ingekleurde gebieden bevinden. Van elk billboard de randen renderen is dus onmogelijk, dan zou er te veel te zien zijn. Daarom wordt in 2 fasen gewerkt. Eerst worden alle particles gerendered, zij het zonder hun randen. Er wordt ook naar de *depth buffer* gerendered, dat is nodig voor fase 2. Tijdens fase 2 gaat een globaal algoritme op zoek naar interessante plaatsen om een silhouet te creëren.

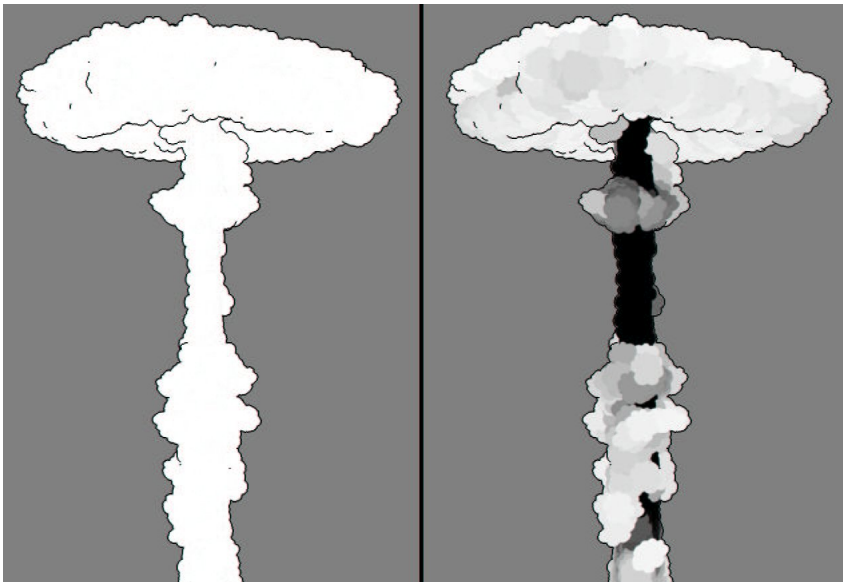
Dit globaal algoritme is een aangepaste vorm van *depth differencing* [9]. Depth differencing kijkt voor elke pixel naar de depth buffer, en als de afgeleide groter is dan een bepaalde drempel γ, wordt er een silhouet pixel getekend. Het idee is dat een pixel interessant genoeg is om tot het silhouet te behoren als zijn diepte erg verschilt van zijn naburige pixels. Depth differencing heeft wel enkele nadelen: Het vertraagt het renderen aanzienlijk, en kan enkel randen van 1 pixel breed produceren. Ook kan aliasing optreden. De afgeleide van de depth buffer kan benaderd worden met finite differencing:

$$\forall i, j color(i,j) = \begin{cases} black & \text{als } \|\nabla z(i,j)\| > \gamma; \\ color(i,j) & \text{anders.} \end{cases} \tag{4.2}$$

Om tot een mooi resultaat te komen moeten de particles op een goede manier beheerd worden. In de eerste plaats dienen er particles gedeleted te worden als ze in een gebied met onvoldoende dichtheid komen. Om gaten te vermijden kan het ook nodig zijn om particles te creëren in gebieden met genoeg densiteit maar onvoldoende particles.

De kwaliteit kan nog verbeterd worden wanneer bij het renderen van de particle primitieven rekening gehouden wordt met bepaalde eigenschappen van de particles. Men doet in [6] enkele voorstellen:

- Maak de grootte van een particle afhankelijk van de dichtheid rondom het deeltje. Dit kan gaten mee helpen vermijden.
- Implementeer squash and stretch [17][11] (zie ook inleiding) door een particle uit te rekken afhankelijk van zijn snelheid.
- Roteer een particle in de richting waarin het zich voortbeweegt.



Figuur 4.4: Resultaten van cartoon billboard rendering uit [6]. Links: een rookpluim. Rechts: dezelfde pluim maar de kleur is afhankelijk van de snelheid.

- Render particles in verschillende kleuren afhankelijk van hun snelheid. Dit kan temperatuur suggereren, en geeft meer variatie en dynamiek (zie 4.4, rechts).

In [23] bouwt men verder op de resultaten van de rendering methode uit [6], met als doel een realtime methode die zoveel mogelijk gebruik maakt van de kracht van de gpu, via shaders. Er wordt nu binnen dezelfde animatie met meerdere rendering primitieven gewerkt. De textures worden uit 3d modellen gegenereerd. In een pre-processing stap worden geometrische modellen gebruikt, bijvoorbeeld door een cluster van random bollen aan te maken. Deze rendert men naar 2 textures. Het eerste texture bevat een normal map in de RGB-kanalen, dus in de kleurkanalen zitten de x, y, en z componenten van de normaal van het geometrisch model dat met die pixel overeenstemt. In het α- kanaal van texture 1 zit per-pixel diepte informatie. Het tweede texture bevat de primitieve zelf, met per pixel een RGB tegen een zwarte achtergrond en een α-waarde. Voor het genereren van de α maakt men het model iets groter, voor de creatie van outlines (zie verder).

Een pixel shader is een programma op de GPU dat per pixel wordt aangeroepen. Zo realiseert men per-pixel toon shading volgens de volgende formule:

$$pixel_{rgb} = K_a(x, y) + C * K_L * q(max(0, N \cdot L))$$

Waarbij K_a per-pixel ambient licht is, C de diffuse kleur uit de texture, K_L de kleur van het licht, N de normaal uit de normal map, L de vector naar de lichtbron. Door per-pixel ambient belichting kunnen zogenaamde *gradient fills* bekomen worden. Dit is populair in strips en zorgt voor variatie in homogeen belichte gebieden. De quantisatiefunctie q kan geïmplementeerd worden via een 1D texture map, zoals uitgelegd in 4.1.

De per pixel diepte informatie wordt ook in de pixel shader bij de depth buffer opgeteld. Echter, door de perspective-transformatie is de verdeling van detail in de z-buffer van nature niet lineair, meer bepaald bij de omzetting naar niet-homogene coördinaten; het meeste detail ligt dicht bij het near plane. Men houdt dus zelf een lineaire depth buffer bij om de (per pixel) diepte-waarden gewoon bij de depth buffer te kunnen optellen. Omdat de α-waarde voor een iets grotere oppervlakte is genomen bij het maken van het texture worden automatisch outlines gerendered: rond het texture is er een grens waar α 1 is, en d 0. Die wordt dus meer in de diepte gerendered, en in het zwart (achtergrondkleur). Zo verkrijgt men geen outlines waar andere particles in de buurt zijn.

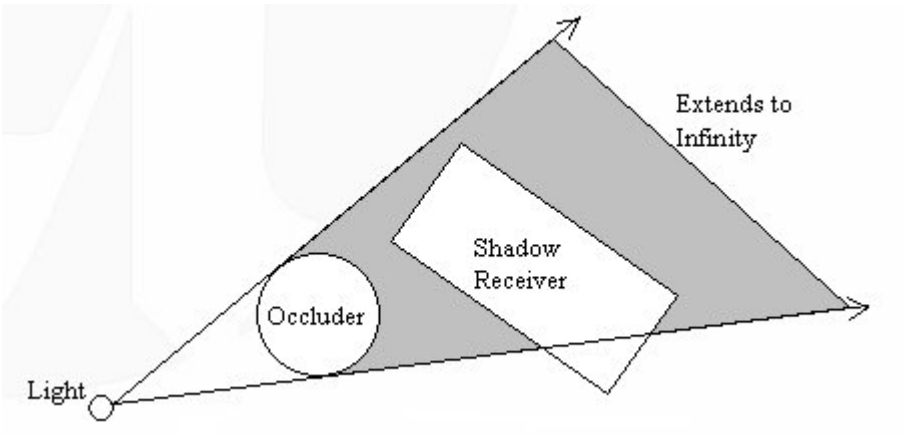
Een andere techniek die men voorstelt voor outlines, zonder pixel shaders te gebruiken, is het renderen van een 2de billboard in outline-kleur, dat achter de primitieve geplaatst wordt. Door de grootte van dit 2de billboard en de afstand te variëren is er controle over dikte en hoeveelheid van de outlines. Er werd ook *self-shadowing* toegevoegd, door de shadow volume methode te gebruiken. Een kleine toelichting:

Shadow volume methode

De *shadow volume* methode bedacht door Frank Crowe is een manier om schaduwen te genereren in 3d scenes. Voor elk object dat een schaduw veroorzaakt (*occluders*) creëert men een volume waarbinnen schaduw te zien is. Om dit volume te vinden, zoekt men eerst naar het silhouet op de occluder; de grens tussen belichte en onbelichte polygonen. Deze grens wordt dan in de richting van het licht tot een volume uitgerokken. Zie figuur (4.5). Dit volume wordt dan naar de stencil buffer gerendered om te bepalen welke pixels in de schaduw liggen en welke niet. Eén manier om dit te doen is:

1. Render de gehele scene (zonder de shadow volumes) zodat de depth buffer correct is gevuld.

2. Render de front faces van de shadow volumes naar de stencil buffer. Incrementeer de stencil buffer als de depth test passed.

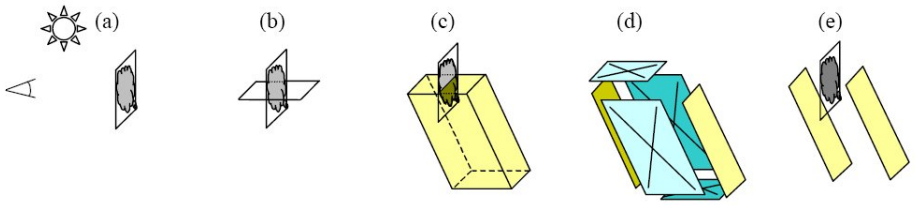


Figuur 4.5: Het shadow volume van een occluder

3. Render de back faces van de shadow volumes naar de stencil buffer, decrementeer indien de depth test passed.

Voor stencil buffer waarden $\neq 0$ zijn de overeenkomstige pixels nu pixels die met beschaduwde zones overeenstemmen. Deze methode heet *depth-pass* maar werkt niet altijd correct. Merk op dat dit een zeer beknopte beschrijving is en dat zowel het vinden van de silhouetten, het uitrekken tot een volume en het renderen ervan verre van triviaal zijn. Hiervoor bestaan telkens verschillende methodes, voor meer info zie [28] en [29].

Om dit te gebruiken voor cartoon rook gaat men nu voor elk billboard een shadow volume afleiden. Men neemt als occluder een vierkant ter grootte van het billboard, dat loodrecht op de view vector staat, zie 4.6 b. Nu heeft men een volume met 6 vlakken, maar 4 daarvan zijn parallel met de view vector (4.6 c end d); zij dragen dus niet bij tot de stencil operaties in het shadow volume algoritme. Enkel het voor - en achtervlak moeten dus gerendered worden, wat resulteert in een zeer minimaal volume (4.6 e). Stel nu dat een billboard half in en half buiten zo'n volume ligt, dan snijdt het een polygon van het volume in een rechte snijlijn. Het gevolg zou een rechte schaduwgrens zijn. Dit zou tot lelijke en ongewenste visuele resultaten leiden. Om dit te verhelpen worden de pixels van het shadow-volume en de billboards met die depth-map aangepast. Zo lijkt de intersectie niet langer een lijn maar wordt ze golvend. Zulke met diepte informatie uitgebreide billboards noemt men *nailboards*, eerder uitvoerig besproken in [24]. Figuur 4.7 geeft een voorbeeld



Figuur 4.6: Een shadow volume genereren voor een quad billboard

van cartoon rook met schaduweffecten uit [23].

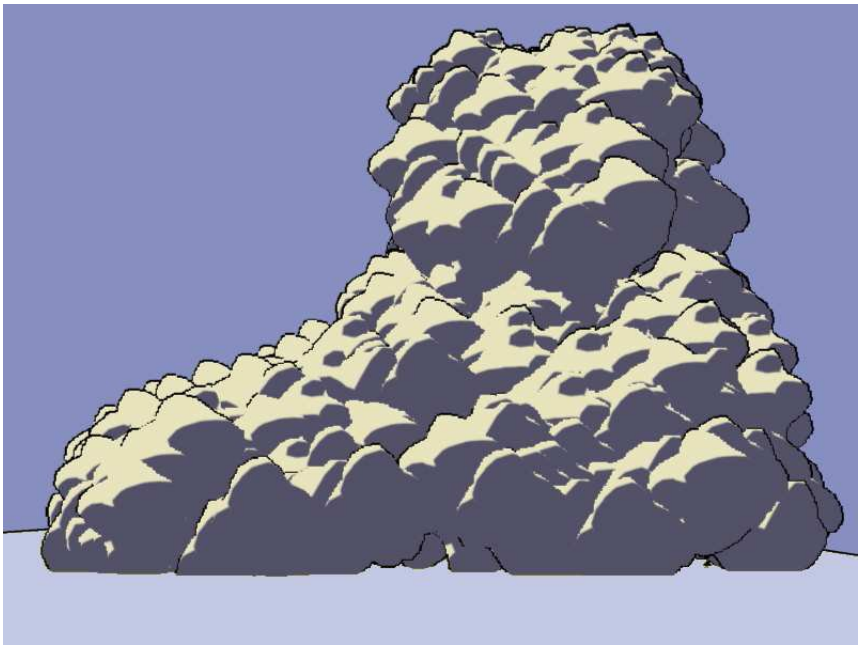
De vervanging van depth differencing en hardware implementatie van de schaduwalgoritmes en toon-shading maken dat de cartoon rook simulatie nu in real time kan gerendered worden. Dit maakt hem nu ook geschikt voor spelletjes. Bovendien is er door het vervangen van depth differencing meer controle voor de animator toegevoegd, die nu ook de dikte van de silhouetten kan regelen.

4.2.2 2.5d gemodelleerde rook

In [4] modelleert men de rook deeltjes als 2.5d objecten (zie ook [1]). Onafhankelijk van de concrete simulatie kan de animator verschillende primitieven modelleren. Voor elke primitieve worden enkele extreme frames gemaakt. Op die manier kunnen primitieven, eens gemaakt, daarna hergebruikt worden bij andere simulaties. Voor een specifieke sequentie kiest de animator enkele primitieven die hij wil gebruiken. Bij hun ontstaan krijgen particles een random primitieve toegekend vanuit de selectie. Ze behouden die tot ze verdwijnen, Dit helpt bij het waarborgen van frame-to-frame coherentie.

Het tekenen van een frame gebeurt dan als volgt: (merk op dat men in [4] de animator de rook zelf laat modelleren volgens een eigen systeem. Dit is niet noodzakelijk voor het renderen, zo lang er particles zijn kunnen die op deze manier gerendered worden. Het maken van het in-between traject wordt dan overgeslagen)

- Genereer een in-between pad - en acceleratie curve en construeer daaruit een 3d pad.
- Laat de particles evolueren doorheen het pad.



Figuur 4.7: Rook gerendered met 2-tone shading, en self shadowing (zie [23]).

- Bepaal een tekenvolgorde op de particles afhankelijk van hun diepte.
- Render elk particle als volgt: bekijk zijn oriëntatie en positie, maak een in-between particle aan de hand van de extreme frames van zijn primitieve (zie ook hoofdstuk 2 en [1]).
- Teken het particle. Dit kan door de particle positie naar scherm coördinaten te transformeren, en op die plek de in-between primitieve te tekenen. De particle als billboard renderen met de in-between als texture geeft een meer 3d effect.

Deze manier van particles renderen heeft een aantal voordelen voor de animator. Hij kan de rook in zijn eigen stijl renderen, wat het makkelijker maakt om in de uiteindelijke scène in passen. Hij kan de primitieven later opnieuw gebruiken, en kan makkelijk met verschillende stijlen experimenteren. Dit is ook een gevolg van het feit dat de primitieven onafhankelijk van de simulatie zijn gemodelleerd.

4.3 Texture advectie

Fluid solvers hebben veel geheugen nodig en worden daarom meestal op relatief 'ruwe' grids uitgevoerd. Om de illusie van meer gedetailleerde flow te bekomen kan men een texture gebruiken dat door de simulatie wordt vervormd. Deze methode kan ook gebruikt worden voor stylized rendering simpelweg door een geschikt texture te kiezen/maken.

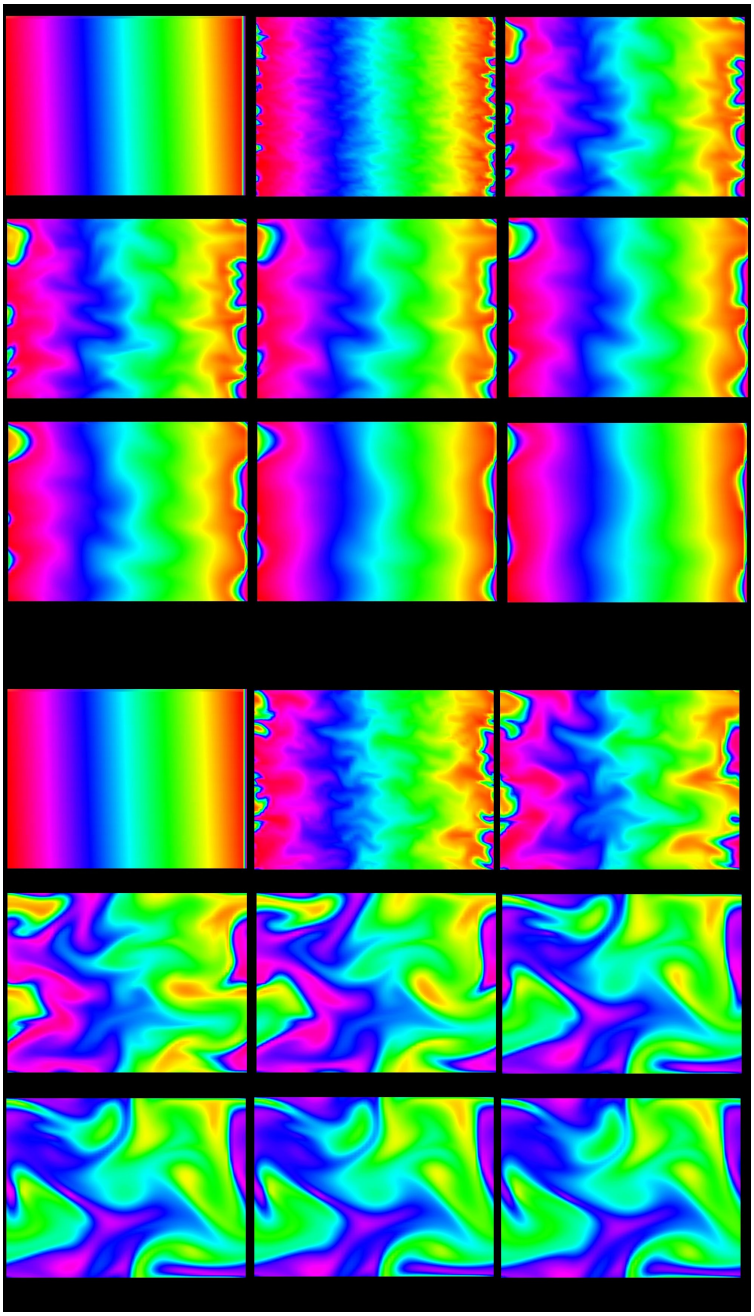
Texture advectie is zeker niet triviaal. Simpele texture advectie kan bekomen worden door advectie van texture coördinaten (u,v) met het fluïdum. Witting [2] stelt differentiaalvergelijkingen voor texture mapping op die samen met de simulatie worden berekend en opgeslagen, zodat ze tijdens het renderen reeds beschikbaar zijn. Maar het texture onbelemmerd laten meebewegen levert problemen op. Na verloop van tijd vervormt het texture te fel en is zijn originele structuur nog nauwelijks zichtbaar. Neyret [5] identificeert 2 vereisten waaraan een goed advectie algoritme moet voldoen:

- Continuïteit in ruimte en tijd. De simulatie moet een continu verloop hebben, zonder dat er breuken ontstaan of de flow wordt belemmerd.
- De lokale statistische eigenschappen van een texture moeten binnen een bepaalde marge behouden blijven. Dit betekent concreet dat een texture nergens te fel mag uitgerokken worden, omdat dan het oorspronkelijke uitzicht van het texture verloren gaat.

Deze vereisten zijn tegengesteld. Als een texture nergens te fel mag worden vervormd, hoe er dan voor zorgen dat de flow nergens in zijn gedrag wordt belemmerd?

4.3.1 Regenerated textures

Stam [18] stelt voor 3 textures te blenden. Voor elk van de drie textures worden na verloop van tijd de texture coördinaten teruggezet naar hun startpositie. Men spreekt van *regenerated textures*. De tijdsduur τ die een texture heeft alvorens zijn coördinaten worden teruggezet is de *latency time*. Elk van de drie textures heeft een gewicht α dat bepaalt in welke mate het meetelt in de blending operatie. Dit gewicht start bij 0 en gaat weer terug naar 0 bij het einde van een levenscyclus. Dit zorgt ervoor dat de animatie continu is in de tijd. Door de cycli van de textures een faseverschuiving van $\frac{2\pi}{3}$ te bezorgen blijft het totale gewicht in de blending constant. De methode werkt vrij goed als het texture min of meer periodiek is, maar niet zomaar voor



Figuur 4.8: Texture advectie met 2 regenerated textures. Boven: advectie met een lage latency. Onder: advectie voor een hoge latency.

elk texture. Bovendien is het moeilijk een geschikte latency τ te kiezen. Als τ groot is dan worden de vervormingen te groot (zie 4.8 bovenaan), terwijl lage waarden voor τ te weinig beweging hebben (4.8 onderaan). Het probleem is dat er één globale waarde is voor τ terwijl die eigenlijk afhankelijk zou moeten zijn van de snelheid en hoeveelheid vervorming, en die varieert heftig afhankelijk van de plaats.

In figuur 4.8 zien we 2 keer hetzelfde texture dat door een fluid simulatie wordt vervormd. De sequenties zijn gegenereerd met een programma dat particles gebruikt voor de advectie van 2 regenerated textures. Als solver gebruikt men de fft solver van Stam [25] (de periodiciteit van deze solver is goed te zien aan de screenshots die bij elkaar 'aansluiten'). Er wordt telkens een zelfde hoeveelheid kracht gebruikt. In de sequentie bovenaan is de latency zeer klein. We zien dat het texture reageert en licht vervormd maar bijna direct stilvalt. Bij lage latency is er trouwens een bijkomend effect, *ghosting*. Dit is moeilijk te zien op de screenshots, maar is goed waar te nemen als men de animatie ziet. Het komt erop neer dat de blending bijna is waar te nemen en er een soort flikkering is te zien. In de sequentie onderaan is een hoge latency gebruikt. Het texture reageert veel heftiger. Het texture blijft ook duidelijk feller en feller vervormen, in contrast met de advectie bovenaan.

4.3.2 Adapted advection

Neyret [5] probeert dit probleem op te lossen en formuleert een eigen methode voor texture advectie. Hij maakt gebruik van N lagen $\{T^i, i = 1..N\}$ van telkens 3 regenerated textures parametrisaties $\{(u^i_j, v^i_j), j = 1..3\}$(zie vorige paragraaf). Zij $\alpha^i_j = \frac{1}{3}\left(1 - \cos\left(2\pi\frac{t-t0^i_j}{\tau^i}\right)\right)$ de blending gewichten en $T()$ de texture map dan is $T^i = \sum_{j=1}^{3} \alpha^i_j T(u^i_j, v^i_j)$. Elke laag heeft zijn eigen latency τ^i en kan dus aan een verschillende snelheid worden aangepast. Het is nu de bedoeling om lokaal, afhankelijk van de snelheid, telkens de meest aangepaste laag T^* te kiezen.

Eerst is een manier nodig om te weten waar welke vervorming van toepassing is. Daarom wordt per vertex, en voor elk texture j^i, de *accumulated deformation* d^i_j bijgehouden. Per laag T^i definieert men de geaccumuleerde vervorming d^i als de som van de vervormingen van zijn 3 texture parametrizaties j^i. Door de faseverschuiving van de textures gedragen de d^i's zich vrijwel constant. De vervorming wordt berekend gebruik makend van de *strain tensor* $\varepsilon = \frac{1}{2}(G + G^t)$ die vervorming beschrijft in de mechanica. G is hier de matrix van de snelheidsgradiënt. De norm hiervan is $\|\varepsilon\| = \sqrt{\sum_{i,j} \varepsilon^2_{i,j}}$.

Door die over de tijd te integreren voor een gegeven stuk van het fluïdum bekomt men de accumulatie $\|\varepsilon\|(x, y)$. Dus elke tijdstap wordt voor elke vertex (x, y) de accumulatie als volgt bijgeteld: $d^i_j(x, y) += \|\varepsilon\|(x, y)dt$.

De gebruiker kan zelf de gewenste hoeveelheid vervorming d^* specifiëren. Dit zal tussen de geaccumuleerde vervorming van 2 lagen liggen: $d^i < d^*$ en $d^* < d^{i+1}$. Dan kan de ideale laag l^* als volgt worden achterhaald:

$$l^* = \frac{d^* - d^i}{d^{i+1} - d^i}$$

Zij f het deel achter de komma van l^*. Nu blendt men lagen i en i+1 met als gewichten f en $1 - f$. Het uiteindelijke texture wordt dus gegeven door:

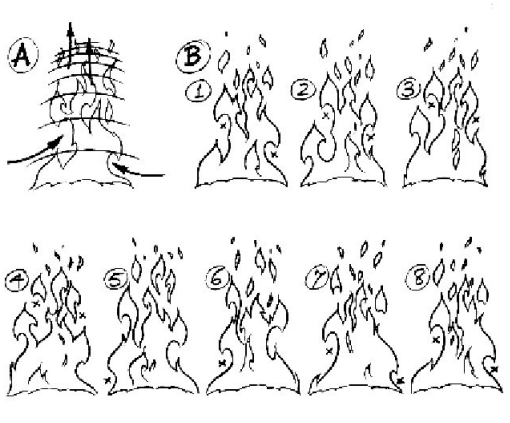
$$T^* = (1 - f)T^i + fT^{i+1}$$

4.4 Simulatie van traditionele, handgetekende 2d animatie

In plaats van een fenomeen te simuleren en daarna uit te tekenen kan je ook het tekenen ervan zelf simuleren. Dit is wat Patterson en Yu gedaan hebben ([31]). Ze bestuderen de manier waarop een animatie van vuur tot stand komt wanneer het volledig met de hand is getekend.

Vuur is een complex en chaotisch fenomeen. Steeds veranderende luchtstromen rond de reactie sturen het dansen van de vlammen. In het warme centrum stijgen de gassen terwijl koelere lucht langs de zijkanten wordt binnengezogen. Dit is zichtbaar als uitsnijdingen die zich van beneden naar boven verplaatsen. In de sequentie van handgetekende vuur frames in figuur 4.9 is dit goed zichtbaar. Het tekenen van zulke frames is tijdrovend, vaak wordt dan ook een kortere sequentie van frames herhaald. Yu en Patterson simuleren tekentechnieken zoals in figuur 4.9 in een procedureel model. Door bepaalde parameters stochastisch te variëren krijg je evengoed een vloeiende vuur-animatie, maar dan zonder frames te herhalen.

Een vlam wordt gegenereerd vanuit een basismodel, aan de hand van parameters. Die parameters bepalen dan verder de vorm en positie van het model. Men gebruikt 5 basismodellen (figuur4.10 onderste rij). Uit de modellen maakt men vlammen met behulp van splines, zoals in de bovenste rij van figuur 4.10. De nodige tussenpunten kunnen uit de parameters gegenereerd worden. Die bepalen de lengte, breedte, hoek, ... van de vlam. Merk



Figuur 4.9: Handgetekende vuur-frames.

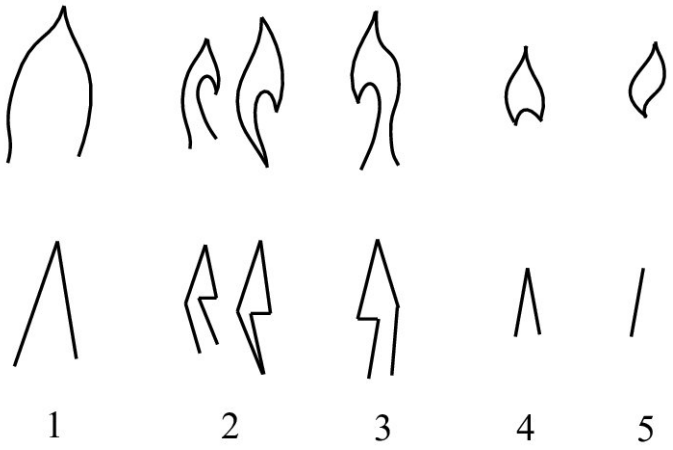
op dat het type vlam ook een parameter is. Door de modellen uit figuur 4.10 te spiegelen kunnen ze bovendien aan de linker- of rechterkant van het vuur worden gebruikt.

In het model onderscheidt men 2 delen. De individuele vlammen boven het vuur, zichtbaar als gesloten curves in figuur 4.9, en de vuurkern. Voor de *top flames* kan men de gesloten modellen gebruiken. De kern bestaat uit vlammen, met open curves, die onderling verbonden zijn door *connection curves*.

Zoals voor vlammen zijn er ook voor de connection curves een aantal modellen met parameters. Die bepalen op welke manier tussenpunten aangemaakt worden tussen 2 vlammen. Die punten zijn dan de controlepunten voor de verbindingskromme.

De top flames worden uit de kern afgeleid. Men neemt posities van de bovenste vlammen in de kern en berekent dan een positie afhankelijk van het tijdstip t. De grootte en breedte variëren ook met de tijd en met een kleine random component.

Om nu tot een animatie van vuur te komen moeten de parameters gekozen worden voor alle vlammen en de connection curves, en dit voor alle tijdstippen t. Yu en Patterson gebruiken voor de vuurkern 7 vlammen, zoals in figuur 4.9. De buitenste vlammen, die zich ook het laagst bevinden, veranderen noch van type noch van positie gedurende de simulatie. Bij de andere vlammen worden bijna alle parameters stochastisch gecontroleerd, om variatie in de vormen te verzekeren.



Figuur 4.10: Verschillende types vlammen.

4.5 Overige mogelijkheden

Buiten texture advection beschrijft Witting [2] nog 2 andere manieren om de output van zijn een fluid solver te renderen:

4.5.1 Temperature contours

De fluid solver die in [2] gebruikt wordt houdt een temperatuurveld bij, dat als volgt gerendered kan worden: (dit geldt natuurlijk ook voor temperatuurvelden gegenereerd door andere solvers)

Er wordt een tekening aangemaakt waarbij temperatuur lineair aan *transparency* wordt gekoppeld. Zo verkrijgt men een tekening die op verschillende plaatsen andere gradaties van doorzichtigheid heeft. Het resultaat kan op een cirkel of rechthoek gemapped worden.

4.5.2 Image smearing

Bij *image smearing* filtert men een foto aan de hand van een 2d snelheidsveld. Gegeven 2 tijdstippen en een simulatie, wordt elke pixel gemixed met de pixels die langs zijn plaats zijn gekomen tijdens de simulatie tussen de 2 tijdstippen. Zo verkrijgt men een 'uitgesmeerde' foto.

4.6 Conclusie

De manier van renderen is natuurlijk in de eerste plaats afhankelijk van de smaak van de animator en het beoogde resultaat. Texture advectie geeft een andere stijl dan een aanpak met particles. Een tweede criterium is de gewenste controle. Simulatie van traditionele technieken is afhankelijk van de gekozen parameters, waarbij het resultaat moeilijk te voorspellen is en wat ervaring vereist. Het gebruik van particles in combinatie met een aantal rendering primitieven geeft meer mogelijkheden. De keuze van de primitieven bepaalt in grote mate de stijl van de animatie. Ook texture advectie kan in de stijl van de animator ingevuld worden, vermits die zelf een texture kan kiezen.

Hoofdstuk 5

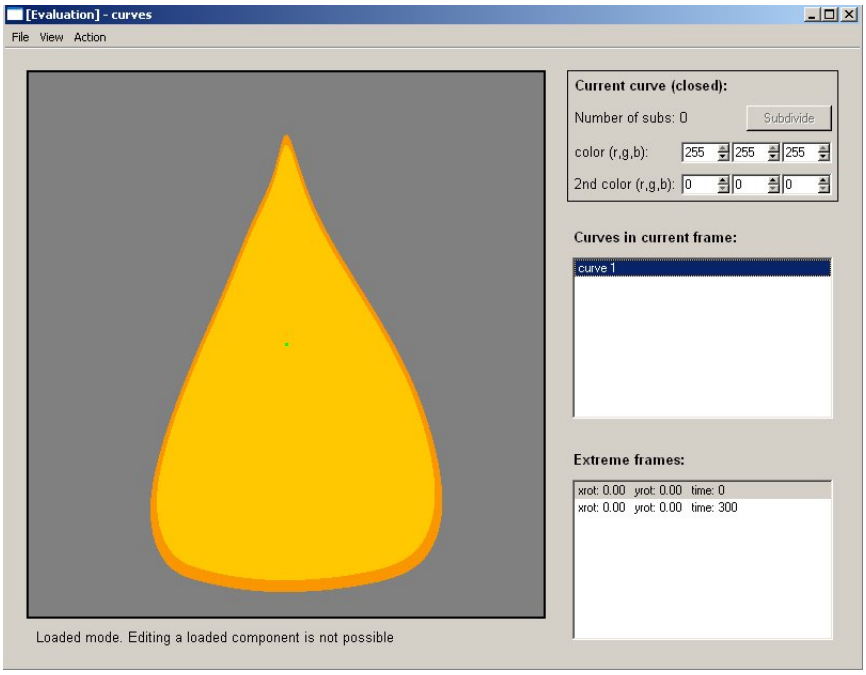
Implementatie

In het kader van dit eindwerk is een implementatie gemaakt. De doelstelling was een aantal gaseous phenomena op een stylized manier te renderen, door een aantal technieken uit deze thesis te implementeren. Snelheid en gebruiksgemak waren niet echt belangrijke criteria, al was het wenselijk dat beide in een minimale hoeveelheid aanwezig waren. In een professionele productieomgeving zijn deze parameters natuurlijk des te belangrijker. De user interfaces, waar aanwezig, zijn gemaakt met behulp van QT. Voor het renderen is OpenGL gebruikt [22].

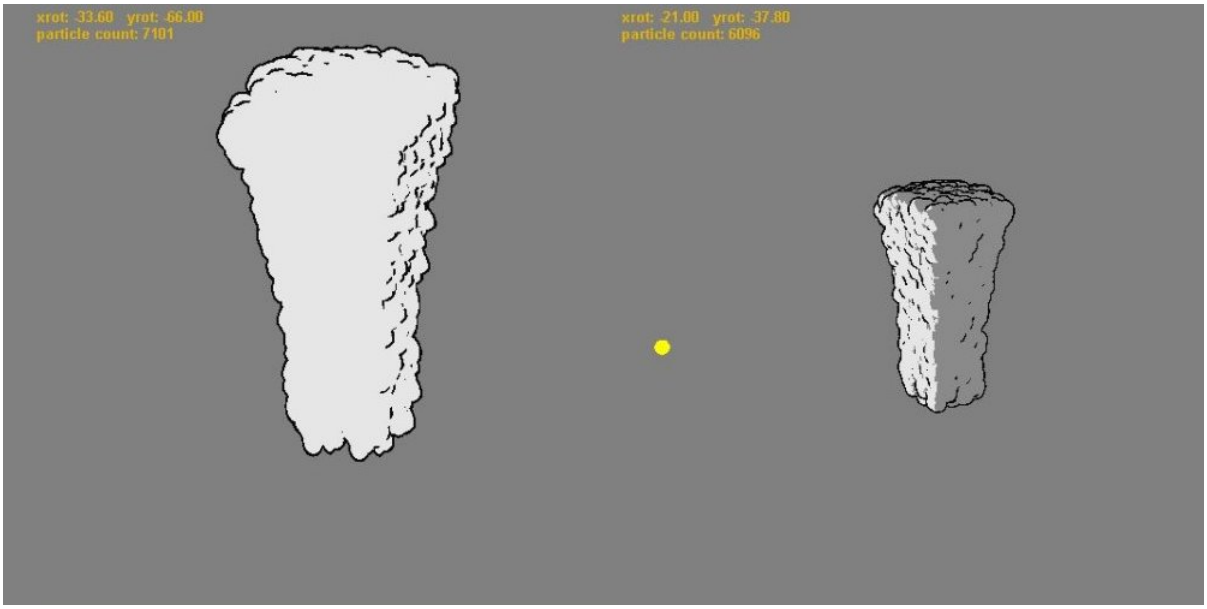
Om 2.5d rendering primitieven te maken is een eenvoudig programmaatje gemaakt (zie figuur 5.1). De extreme frames van een 2.5d primitieve staan in de lijst rechts, aan de hand van hun rotatiehoek t.o.v. de x -en y-as, en hun frametijd. Subdivision curves worden gebruikt als 'tekenprimitieve'; de gebruiker voegt controlepunten toe, en kan nadien een aantal keer subdividen. Een primitieve kan uit meerdere open en/of gesloten curves bestaan. Natuurlijk moeten voor elk extreme frame evenveel curves en controlepunten gespecificeerd zijn, zodat er correct geïnterpoleerd kan worden. Voorts kan men ook de kleur van de primitieve en zijn silhouet kiezen. In figuur 5.5 zijn enkele voorbeeldprimitieven te zien.

5.1 Billboarding

Om rook op een fysische manier te simuleren is de 2d solver van Stam ([19]) als vertrekpunt gebruikt. Deze solver is met 1 dimensie uitgebreid. Dit is niet helemaal gelukt, in 3d wordt hij soms onstabiel. Voor onze doeleinden was het echter voldoende enkele situaties te kunnen simuleren. De solver wordt



Figuur 5.1: De user interface om 2.5d primitieven te maken.



Figuur 5.2: Voorbeeld van een rook rendering met billboards. Rechts het effect van self shadowing ten opzichte van de (gele) puntlichtbron.

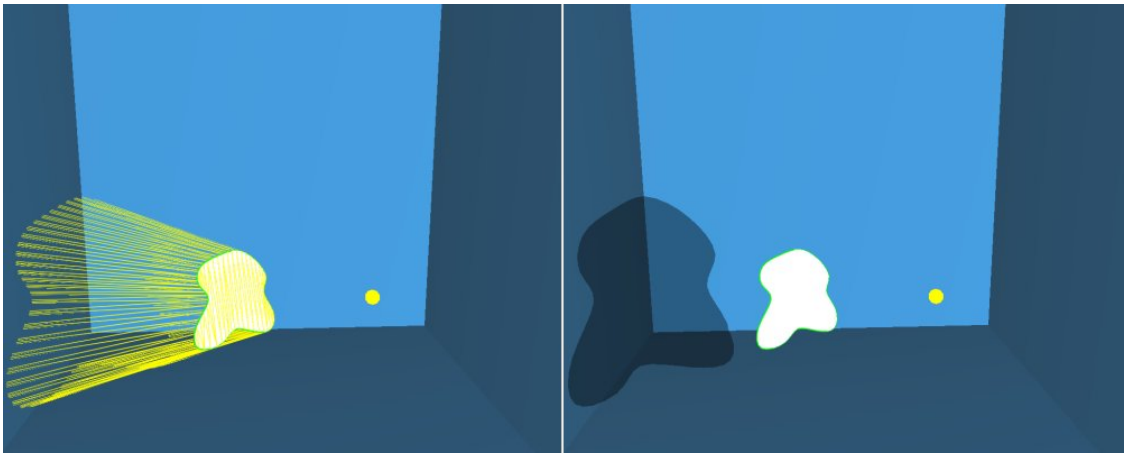
gebruikt om particles te advecteren. Die particles worden met 2.5d primitieven gerendered, zoals voorgesteld in [4] (zie sectie 4.2.2). Een screenshot van het rendering programma zie je in figuur 5.2.

Elk particle krijgt bij zijn ontstaan een vaste primitieve toegekend, wat voor coherentie tussen frames moet zorgen. Bij het renderen wordt voor elk particle een in-between gegenereerd uit de extreme frames van zijn 2.5d component. Dit doen we door de naburige extreme frames te zoeken, gebruik makend van de huidige rotatiehoeken t.o.v. x- en y-as. De in-between wordt dan als een billboard gerendered zodat hij naar de camera gericht is en zijn 2d vorm niet verraadt.

Voor het genereren van outlines zijn 2 technieken geïmplementeerd. De eerste is het gebruik van depth differencing (zie [6], zie 4.2.1). Dit vereist een 2de rendering pass omdat eerst de depth buffer moet gevuld zijn, en de outlines zijn slechts 1 pixel breed. Een tweede manier is het renderen van een extra silhouet billboard per particle (zie [23]). Dit extra billboard wordt achter het gewone billboard geplaatst, t.o.v. de camera positie, en is iets groter. De resultaten zijn beter, omdat de grootte van de outlines nu kan aangepast worden. Het renderen verloopt op deze manier ook veel sneller dan bij depth differencing.

Geïnspireerd door de mooie resultaten van self-shadowing in [23] is gepoogd de shadow volumes techniek te implementeren voor de gebruikte 2.5d primitieven. Het silhouet is al gegeven in de punten van de curve. Voor het genereren van een volume 'billboarden' we een primitieve in de richting van de lichtbron. De volumes renderen we daarna via de stencil buffer (zie 4.2.1). In figuur 5.3 staat een primitieve en zijn shadow volume, en de schaduw die dit oplevert. Zoals men ziet bestaat een volume uit een groot aantal driehoeken. Voor een primitieve met n punten in z'n silhouet zijn dat er 2n voor de zijvlakken alleen. En voor onze primitieven is n al vlug groter dan 10. Als we dit toepassen op een particle systeem met enkele duizenden particles vertraagt dat het systeem enorm. Voordeel is dat door het detail van de shadow volumes zelf we redelijk mooie intersecties hebben met de andere primitieven. Dit vermijdt veel van de moeilijkheden die ontstaan als men shadow volumes wil genereren voor getexturede billboards zoals in [23]. In figuur 5.2 rechts zien we het effect van self-shadowing op de rendering van de billboards.

De gebruiker kan de rendering opties via een menu instellen, zodat het effect meteen zichtbaar is. Buiten de outlines zijn er nog enkele andere opties. Zo kan de grootte van de particles afhankelijk worden gemaakt van de dichtheid in zijn omgeving, of ze worden gestretched in de richting waarin



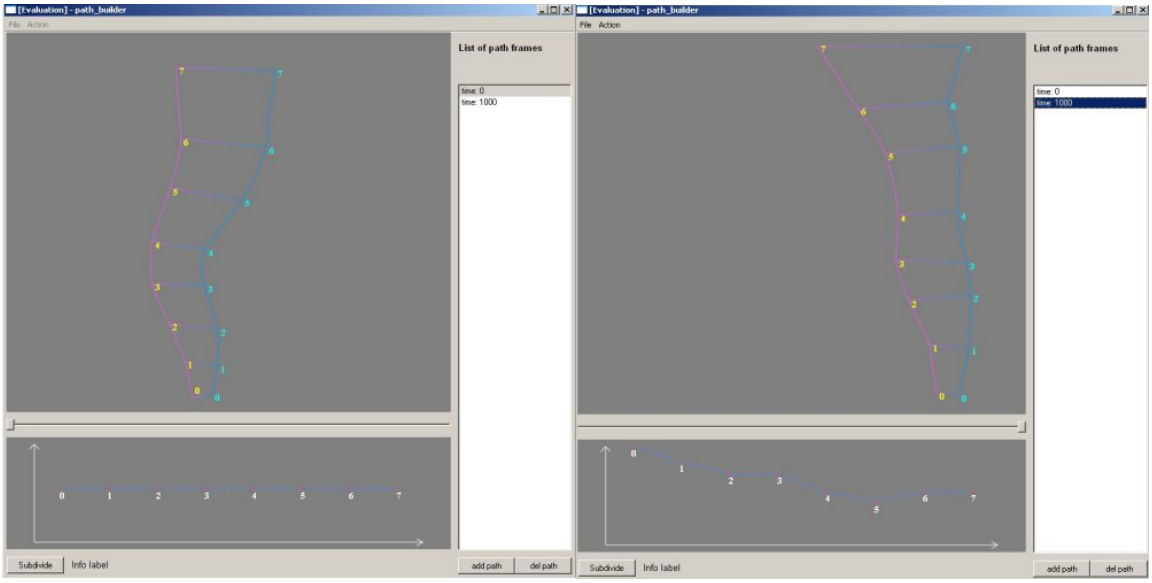
Figuur 5.3: Links: het shadow volume voor een primitieve ten opzichte van de gele puntlichtbron. Rechts: de schaduw op de muren van de kamer

ze bewegen (zie [6]). Dit laatste effect is een toepassing van het (squash en) stretch principe voor cartoons zoals ook beschreven in [17] (zie 2.3).

5.2 Extreme paths

Om een beter gecontroleerde simulatie te verkrijgen is een implementatie van *extreme paths* gemaakt, zoals voorgesteld in [4] (zie 3.2). De gebruiker kan in een apart programma paden tekenen 5.4. Bij elk pad hoort een tijdstip, waarop het pad er zo hoort uit te zien, en een snelheidscurve. Met die snelheidscurve kan men op verschillende plaatsen in het pad verschillende snelheden instellen. Een pad bestaat uit 2 uiterste curves. Om het design van de paden te vergemakkelijken worden subdivision curves gebuikt die door de controlepunten gaan.

Tijdens de simulatie wordt voor een gegeven tijdstip t het in-between pad voor t berekend, alsook de bijbehorende snelheden. Dit gebeurt via gewone lineaire interpolatie. De particles die worden losgelaten volgen een random weg tussen de 2 uiterste curves van het in-between pad. Ze krijgen ook een random z-coördinaat toegekend. Op die manier is een diepte-ordening mogelijk. Verder krijgt ook elk particle een vaste primitieve, en een random grootte die tussen bepaalde grenzen varieert. Het renderen van een frame komt dan neer op het geordend tekenen van de particles. Eerst wordt weer de juiste in-between primitieve berekend en getekend voor elk particle.



Figuur 5.4: De user interface om extreme paths te ontwerpen.

Eventueel worden outlines gegenereerd door een 2de, grotere primitieve te gebruiken, zoals reeds eerder beschreven. Het ordenen op basis van de z-coördinaat laten we over aan de z-buffer. We maken geen gebruik meer van billboarding, gezien we een 2d-toepassing beogen. De view vector is gericht in de richting van de negatieve z-as en een orthografische projectie zorgt voor een 2d-rendering effect. Dit systeem vormt een basis waarmee rook, water en vuur zijn gerendered.

Test case: rook

Om rook te renderen werden vrij eenvoudige primitieven gebruikt, simpele gegolfde vormen. Zwarte outlines geven een mooi cartoon-achtig resultaat. In fig. 5.7 een voorbeeld van donkere rook. Het gebruikte pad is dat in fig. 5.4. De bruine primitieven in 5.5 werden voor deze simulatie gebruikt. Door de kleur van de primitieven tijdsafhankelijk te maken krijgt het geheel wat meer dynamiek. De particles worden groter naarmate ze verder het pad volgen. Aan het eind van het pad faden de particles weg door blending te gebruiken. Een andere optie zou zijn ze te laten krimpen. De particles gewoon verwijderen geeft een veel bruter effect.

Test case: water

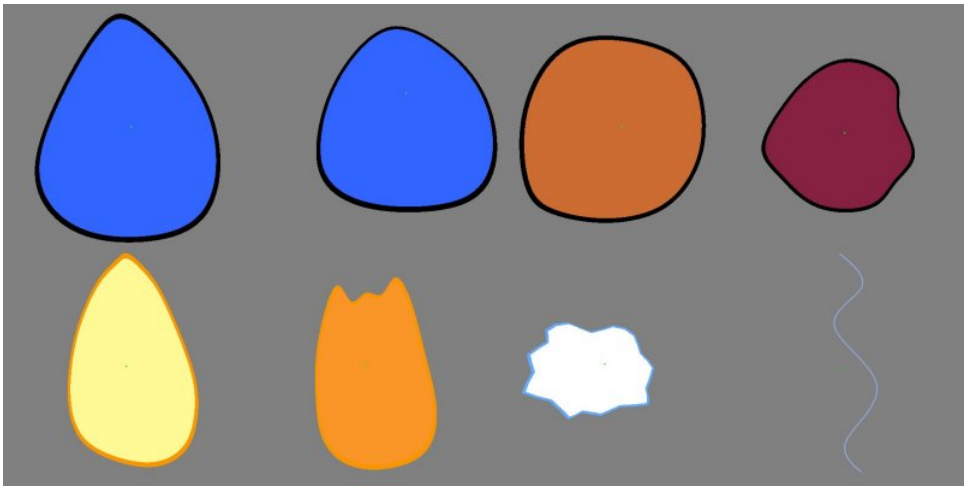
Door fel blauwe druppelvormige primitieven met een lichtblauwe outline te gebruiken krijgt een simulatie een wateruitzicht (fig. 5.8). Af en toe krijgt een particle een *speedline* primitieve toegewezen. Zulke speedlines geven een extra visuele hint voor de richting en de snelheid van de stroom. De speedlines worden bovenop de andere primitieven getekend, ze moeten immers altijd zichtbaar zijn. Afhankelijk van de snelheid worden de speedlines gestretched. Op het einde van het pad krijgen de particles een nieuwe *schuim/foam* primitieve toegewezen. Hun richtingsvector wordt berekend alsof ze tegen het uiteinde van het pad aanbotsen en weerkaatsen. De schuimparticles zijn onderhevig aan een beetje zwaartekracht en faden weg na een aantal tijdstappen.

Test case: vuur

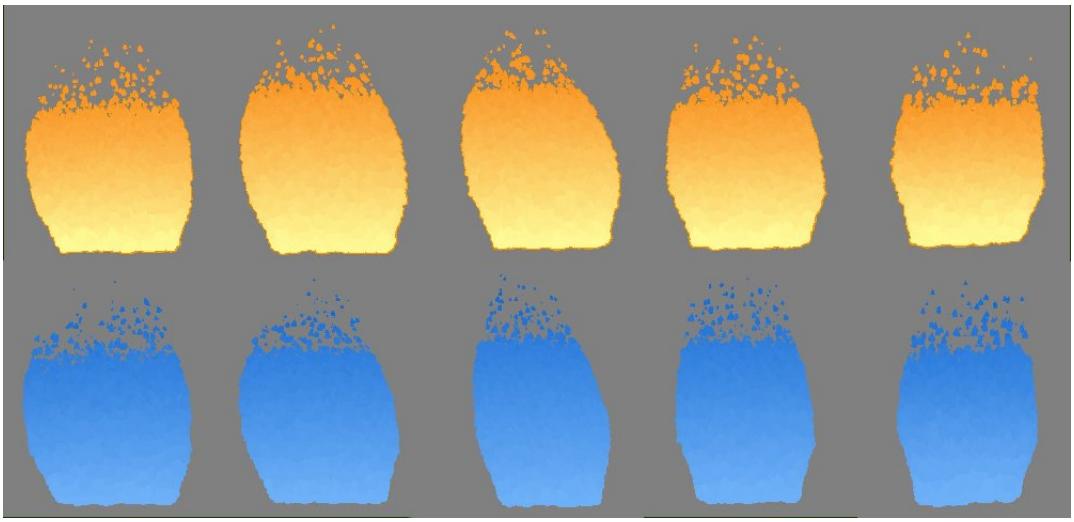
Vuur is misschien wel het meest chaotische fenomeen vergeleken met water en rook. Een minimale variatie in de kleur van de vlammen is nodig. Om losse vlammen boven de basis te bekomen krijgen de particles een extra attribuut, namelijk levensduur. De leeftijd van de particles wordt heftig gevarieerd, en slechts een klein percentage mag ouder worden dan een bepaalde grens. Zij vormen de ontsnappende vlammen bovenaan. Particles die hun maximum ouderdom hebben bereikt krimpen in grootte tot ze helemaal verdwijnen. Variatie in maximum leeftijd, grootte en krimptijd worden als paramaters ingesteld en geven andere effecten. Figuur 5.9 toont een vuuranimatie op verschillende tijdstippen. In figuren 5.1 en 5.5 links onderaan staan 2 vlamfiguren die zijn gebruikt. In fig. 5.6 staat een vergelijking van een vlamanimatie. De reeks bovenaan is gerendered met gele vlammen en outlines, onderaan met blauwe vlammen zonder outlines.

5.3 Conclusie

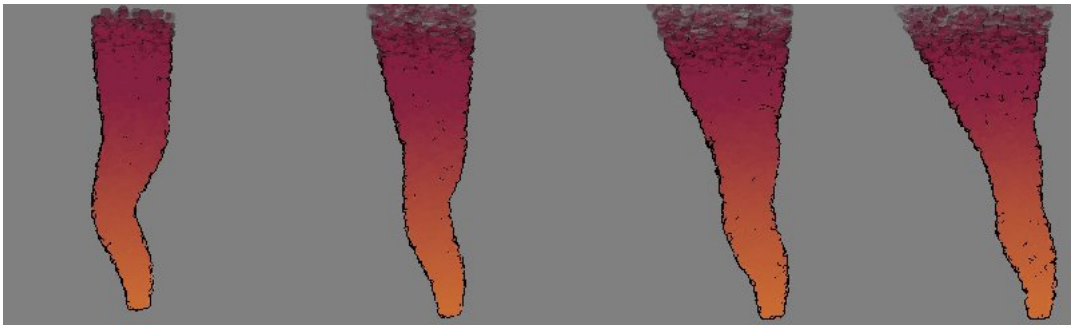
Het gebruik van 2.5d rendering primitieven is een zeer flexibele manier om particles te renderen. Met slechts enkele extreme frames kan een animator voor veel variatie zorgen in de uiteindelijke rendering, bovendien in zijn eigen stijl, en met mogelijkheden voor hergebruik. Bovendien kunnen we ze gebruiken om eender welk soort particles te renderen, of die nu een fluïd solver volgen of een zelfgedefinieerd pad. We hebben ze gebruikt voor een volledige 2d toepassing en in een 3d omgeving via billboarding. Toch zijn er ook een aantal minpunten; Het feit dat er elk frame in-betweening nodig is veroorzaakt niet alleen extra berekening, maar maakt dat er elke rendering



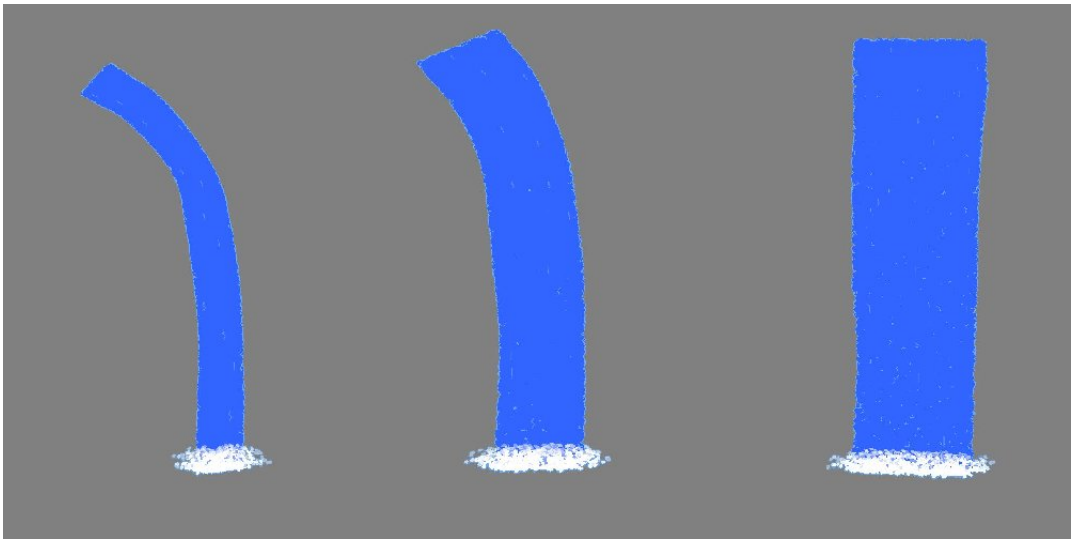
Figuur 5.5: Enkele van de gebruikte 2.5d primitieven. Linksboven een waterdruppel vanuit verschillende standpunten. Linksonder 2 extreme frames van een vlam. Rechtsboven 2 donkergekleurde rook-puffs. Rechtsonder een foamprimitieve en speedline.



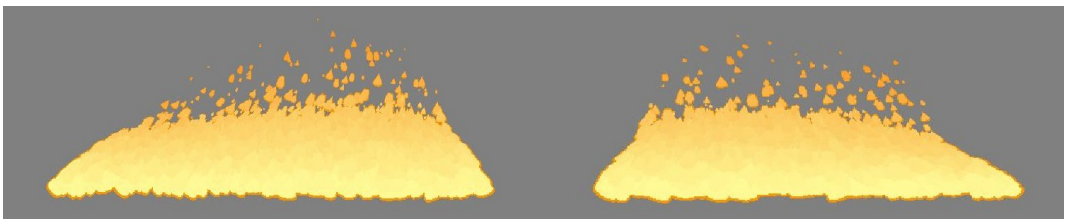
Figuur 5.6: Een dikke, brandende vlam. Bovenaan zijn geel-oranje vlammen met oranje outlines gebruikt als primitieve. Onderaan blauwe vlammen zonder outlines.



Figuur 5.7: 2d donkere rook.



Figuur 5.8: Een waterval simulatie op verschillende tijdstippen.



Figuur 5.9: Een breed vuur op verschillende tijdstippen.

stap nieuwe vertices naar de grafische kaart moeten worden gestuurd. Dit is een hele grote bottleneck, zeker omdat die subdivision curves nogal wat punten bevatten.

In systemen die met 'vaste' primitieven werken kunnen vooraf textures gegenereerd worden. Die laadt men vooraf in het grafische geheugen zodat elke tijdstap per particle slechts een quad(4 vertices + 4 texture coördinaten) moeten worden verzonden naar de grafische kaart. Met de point sprite extensie wordt dit nog versneld.

Qua gebruiksgemak zijn in-between paden handig om snel een gewenste 2d animatie mee te ontwerpen. Ze zijn geschikt voor verschillende soorten fenomen, en in die zin dus ook zeer flexibel. We hebben aangetoond dat er verschillende soorten gaseous phenomena vrij gemakkelijk mee te simuleren zijn, op een door de animator gekozen manier.

Het afstellen van een particle simulatie of fluïd solver met krachten, vergt veel meer geduld. Het levert wel een soort realistisch gedrag dat moeilijk te kopiëren is door paden te tekenen.

5.4 Mogelijke uitbreidingen

Voor 3d applicaties zou het interessant zijn onze billboard toepassing in real-time te laten draaien. We zouden de billboardstap door een vertex shader kunnen laten doen. Dit is makkelijk te realiseren via offsetting in oogcoördinaten. Zoals eerder vermeld vormt de in-betweening van 2.5d primitieven een grote flessenhals. Om realtime snelheden te bereiken met deze primitieven moet alvast een snellere interpolatiemethode geïmplementeerd worden.

Een andere mogelijkheid is, uitgaande van bestaande 2.5d primitieven, in een pre-processing stap een aantal in-betweens te genereren en naar een texture te renderen. Elk particle kennen we dan een vast texture toe. Zo kunnen we ze renderen als getexturede quads, wat veel snelheidswinst zou geven, maar we missen dan wel 'at-runtime' interpolatie. Door een aantal in-betweens te genereren kunnen we toch een stuk variatie behouden. Echter, self-shadowing en cell shading op de gpu toevoegen zoals in [23] is niet mogelijk voor 2.5d primitieven, daarvoor missen we de informatie in de normal -en depth map, die ze verkregen door hun primitieven expliciet uit geometrische modellen te genereren.

Een simulator met collision detection kan een andere interessante uitbreiding zijn. Zowel een fluid solver als het extreme path systeem zijn hier in theorie voor geschikt. De geparametriseerde particle simulator uit [3] beschikt over interactie met vaste objecten. De particles botsen er tegen aan alsof ze geen massa hebben, de objecten zelf voelen geen kracht van de botsingen. Een van de mogelijke problemen zou kunnen zijn dat particles die te dicht bij objecten in de scene komen met hun primitieven gedeeltelijk in die objecten komen te hangen.

Bibliografie

[1] Fabian Di Fiore, Philip Schaeken, Koen Elens, Frank Van Reeth. **Automatic In-betweening in Computer Assisted Animation by Exploiting 2.5D Modelling Techniques**. In Proceedings of Computer Animation (CA), blz. 192–200, 2001.

[2] Patrick Witting. **Computational fluid dynamics in a traditional animation environment**. In Proceedings of SIGGRAPH, blz. 129–136, 1999.

[3] Morgan McGuire. **A real-time, controllable simulator for plausible smoke**. Tech Report, Brown University, 2006.

[4] Fabian Di Fiore, Johan Claes, Frank Van Reeth. **A framework for user control on stylized animation of gaseous phenomena**, Proceedings of Computer Animation and Social Agents (CASA), blz. 171–178, 2004.

[5] Fabrice Neyret. **Advected textures**. In Proceedings of the Eurographics Symposium on Computer Animation, blz. 147-153, 2003.

[6] Stephen Chenney, Alex Mohr, Andrew Selle. **Cartoon rendering of smoke animations**. In Proceedings of Non-Photorealistic Animation and Rendering (NPAR), blz. 57-60, 2004.

[7] Antoine McNamara, Zoran Popovic, Jos Stam, Adrien Treuille. **Keyframe control of smoke simulations**. ACM Transactions on Graphics (TOG) 22, 3, blz. 716-723, 2003.

[8] David H. Laidlaw. **Loose, artistic 'textures' for visualzation**. IEEE Computer Graphics and Applications 21, 2, blz. 6-9, 2001.

[9] Oliver Deussen, Thomas Strothotte. **computer-generated pen-and-ink illustration of trees**. In Proceedings of SIGGRAPH, blz. 13-18, 2000.

[10] James T. Enns, Cristopher G. Healy. **Perception and painting: A search for effective, engaging visualizations**. IEEE Computer Graphics and Applications 22, 2, blz. 10-15, 2002.

[11] Stephen Chenney, Rob Iverson, Mark Pingel, Marcin Szymanski. **Simulating cartoon style animation**. In Proceedings of Non-Photorealistic Animation and Rendering (NPAR), blz. 133-138, 2002.

[12] Ronald Fedkiw, Willi Geiger, Duc Quang Nguyen, Nick Rasmussen. **Smoke simulation for large scale phenomena**. ACM Transactions on Graphics (TOG) 22, 3, blz. 703-707, 2003.

[13] Ronald Fedkew, Henrik Wann Jensen, Jos Stam. **Visual simulation of smoke**. In Proceedings of SIGGRAPH, blz. 251-260, 2001.

[14] R.M. Kirby, David H. Laidlaw, H. Marmanis. **Visualizing multivalued data from 2D incompressible flows using concepts from painting**. In Proceedings of the conference on Visualization, blz. 333-340, 1999.

[15] W.T. Reeves. **Particle systems - A technique for modeling a class of fuzzy objects**. In Proceedings of SIGGRAPH, blz. 359-376, 1983.

[16] Tommi Ilmonen, Janni Kontkanen. **The second order particle system**. In Journal of Winter School on Computer Graphics (WSCG2003), blz. 240-246, 2003.

[17] John Lasseter. **Principles of traditional animation applied to 3D computer animation**. In Proceedings of SIGGRAPH, blz. 35-44, 1987.

[18] Jos Stam. **Stable fluids**. In Proceedings of SIGGRAPH, blz. 121-128, 1999.

[19] Jos Stam. **Real-time fluid dynamics for games**, In Proceedings of the Game Developer Conference, 2003.

[20] Nick Foster, Dimitris Metaxas. **Realistic animation of liquids**. In Graphical Models and Image Processing 58, blz. 471–483, 1996.

[21] Nick Foster, Dimitris Metaxas. **Modeling the motion a hot, turbulent gas**. In Proceedings of SIGGRAPH, blz. 181–188, 1997.

[22] Tom Davis, Jackie Neider, Dave Schreiner, Mason Woo. **The OpenGL programming guide version 1.2**

[23] Andi Fein, Morgan McGuire. **Real-time rendering of cartoon smoke and clouds**, In Proceedings of Non-Photorealistic Animation and Rendering (NPAR), blz. 21–26. 2006.

[24] Gernot Schaufler. **Nailboards, A rendering primitive for image caching in dynamic scenes**. In Proceedings of Eurographics, blz. 151-162, 1997.

[25] Jos Stam. **A simple fluid solver based on the fft**, In Journal of Graphics Tools, 6(2), blz. 43–52, 2001.

[26] Johan Claes, Fabian Di Fiore, Gert Van Sichem, Frank Van Reeth. **Fast 3D cartoon rendering with improved quality by exploiting graphics hardware**, In Proceedings of Image and Vision Computing New Zealand (IVCNZ), blz. 13–18, November, 2001.

[27] Adam Lake, Carl Marshall, Mark Harris, Marc Blackstein. **Stylized rendering techniques for scalable real-time 3d animation**, In Proceedings of Non-Photorealistic Animation and Rendering (NPAR), 13–20, 2000.

[28] Eric Lengyel. **The mechanics of robust stencil shadows**, http://www.gamasutra.com/features/20021011/lengyel_01.htm, 2002.

[29] Hun Yen Kwoon. **The Theory of Stencil Shadow Volumes**, http://www.gamedev.net/reference/articles/article1873.asp, 2002.

[30] Antonio Fernandes. **Billboarding tutorial** http://www.lighthouse3d.com/opengl/billboarding/

[31] John W. Patterson, Jinhui Yu. **A fire model for 2d computer animation**. Computer Animation and Simulation '96, Springer-Verlag, blz. 49–60, 1996.

[32] Jonathan Richard Shewchuck. **An introduction to the conjugate gradient method without the agonizing pain**. 1994.

[33] **Wikipedia, the free ecyclopedia** http://wikipedia.org

Auteursrechterlijke overeenkomst

Opdat de Universiteit Hasselt uw eindverhandeling wereldwijd kan reproduceren, vertalen en distribueren is uw akkoord voor deze overeenkomst noodzakelijk. Gelieve de tijd te nemen om deze overeenkomst door te nemen en uw akkoord te verlenen.

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Non-Photorealistic Rendering and Animation of Gaseous Phenomena
Richting: **Licentiaat in de informatica** Jaar: **2006**
in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Deze toekenning van het auteursrecht aan de Universiteit Hasselt houdt in dat ik/wij als auteur de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij kan reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

U bevestigt dat de eindverhandeling uw origineel werk is, en dat u het recht heeft om de rechten te verlenen die in deze overeenkomst worden beschreven. U verklaart tevens dat de eindverhandeling, naar uw weten, het auteursrecht van anderen niet overtreedt.

U verklaart tevens dat u voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen hebt verkregen zodat u deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal u als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze licentie

Ik ga akkoord,

Dries HERBOTS

Datum: