# *Extending an Haptic API*

**Pascal Porta**

promotor :
Prof. dr. Chris RAYMAEKERS

Universiteit Maastricht

Eindverhandeling voorgedragen tot het bekomen van de graad
Master in de informatica human-computer interaction

universiteit hasselt

Universiteit Hasselt

# Extending an Haptic API

Thesis voorgedragen tot het behalen van de graad van master in de informatica

**Pascal Porta**

Promotor: Prof. Dr. Chris Raymaekers

Begeleider: Lode Vanacken

Academiejaar 2006-2007

**Abstract**

Deze thesis heeft tot doel een overzicht te geven van API's voor haptisch renderen. We zullen starten met een overzicht van bestaande technieken en algoritmen. Er zal een opsomming worden gegeven van bestaande haptische API's waarna we een selectie zullen maken van welke API's we zullen testen. Nadien zullen we een conclusie proberen te maken om te zien welke API het beste is. Tenslotte zal er ook een uitbreiding worden gemaakt voor HAL, de haptische library van het expertisecentrum voor digitale media.

## Woord vooraf

Deze thesis markeert het einde van mijn universitaire studies. Ik zou dit werk nooit vervolledigd kunnen hebben zonder de hulp van een aantal personen, die ik dan ook graag hier zou willen bedanken.

Eerst en vooral zij die me begeleid hebben bij het maken van deze thesis, mijn promotor Prof. Dr. Chris Raymaekers en mijn begeleider Lode Vanacken. Dank u voor de wijze raad en de vele tips die me goed hielpen mijn eindwerk te vervolledigen.

Eveneens zou ik graag enkele mensen willen vermelden die me tijdens mijn hogere studies hebben bijgestaan. Zo wil ik mijn moeder en zus Annick bedanken voor hun steun en het vele nalezen; in het bijzonder mijn moeder zonder wie ik deze studies nooit had kunnen aangaan.

Tenslotte zou ik ook graag enkele medestudenten willen aanhalen, die in de jaren dat ik studeerde voor een goede motivatie en sfeer hebben gezorgd. Bedankt Kathleen, Bart en Jochem voor de goede tijden die we samen beleefd hebben.

Pascal Porta

# Inhoudsopgave

Hoofdstuk 1 Inleiding ..... 14
Deel I Literatuurstudie ..... 15
Hoofdstuk 2 Toepassingsdomeinen ..... 17
Medische sector ..... 17
Chirurgie ..... 17
Tandheelkunde ..... 19
Simulatoren ..... 20
Teleoperaties ..... 21
Entertainment ..... 22
Kunst ..... 23
Conclusie ..... 24
Hoofdstuk 3 Haptics bij het menselijk lichaam ..... 25
Haptics bij de mens ..... 25
Een kunstmatig haptisch systeem ..... 26
Conclusie ..... 28
Hoofdstuk 4 Algoritmen & technieken ..... 29
Botsingsdetectie ..... 29
Bounding volume hiërarchie ..... 30
Spatial subdivision ..... 35
Binary space partitioning ..... 36
Optimized Spatial Hashing ..... 37
LDI Collision Detection ..... 38
Voronoi diagrammen ..... 40
Scenegraphs ..... 41
Haptic rigid body simulation ..... 43

Surface haptics ..... 44
Haptic texture rendering ..... 46
Conclusie ..... 46
Hoofdstuk 5 Bestaande API's ..... 48
Ghost SDK ..... 49
OpenHaptics Toolkit ..... 51
H3D ..... 53
Chai 3D ..... 55
Haptik Library ..... 57
OpenSceneGraph Haptic Library (osgHaptics) ..... 59
HAL ..... 60
Reachin API ..... 62
Andere toolkits ..... 63
Conclusie ..... 63
Hoofdstuk 6 Bestaande hardware ..... 64
SensAble PHANToM ..... 64
Force Dimension ..... 64
Novint ..... 65
Conclusie ..... 65
Deel II Implementatie ..... 66
Hoofdstuk 7 Demo's haptische API's ..... 67
SensAble OpenHaptics Academic Edition v2.0 ..... 67
Conclusie ..... 71
CHAI 3D v1.6.1 ..... 72
Conclusie ..... 76
SenseGraphics H3D API v1.5 ..... 76
Conclusie ..... 80

HAL v0.1.1 ..... 80
Conclusie ..... 83
Featurevergelijking en benchmarks ..... 84
SensAble OpenHaptics Academic Edition v2.0 ..... 84
CHAI 3D v1.6.1 ..... 85
SenseGraphics H3D API v1.5 ..... 85
HAL v0.1.1 ..... 85
Conclusie ..... 86
Gebruiksgemak ..... 86
SensAble OpenHaptics Academic Edition v2.0 ..... 86
CHAI 3D v1.6.1 ..... 87
SenseGraphics H3D API v1.5 ..... 87
HAL v0.1.1 ..... 87
Conclusie ..... 88
Features ..... 88
Conclusie ..... 89
Performantie ..... 91
Scenario 1: 1 kubus zonder textuur ..... 92
Scenario 2: 1 kubus met textuur ..... 94
Scenario 3: 100 kubussen zonder textuur ..... 97
Conclusie ..... 101
Conclusie ..... 101
Hoofdstuk 8 Uitbreiding HAL ..... 103
HAL extentie 1: Het elimineren van de gebondenheid aan enkel de haptische apparaten van SensAble ..... 103
Conclusie ..... 106
HAL extentie 2: Het uitbreiden van het pseudodevice met een spacemouse . 106

Conclusie ..... 108
HAL extentie 3: Het uitbreiden van het pseudodevice met een grafisch front-end ..... 108
Conclusie ..... 110
Hoofdstuk 9 Conclusie ..... 111
Algemene conclusie ..... 111
Future work ..... 113
Bibliografie ..... 115
Bijlage 1 ..... 120
Bijlage 2 ..... 121
Bijlage 3 ..... 122
Bijlage 4 ..... 123
Bijlage 5 ..... 124
Bijlage 6 ..... 125
Bijlage 7 ..... 126

# Lijst met figuren

Figuur 1 – Een Da Vinci chirurgische robot [38] ..... 18
Figuur 2 – Het SOFA framework [2] ..... 18
Figuur 3 – Een laproscopische simulator [37] ..... 19
Figuur 4 – Een tandheelkundige simulatie [31] ..... 19
Figuur 5a – Een basis vliegsimulator met MSFS [39] ..... 20
Figuur 5b – Een pneumatische vliegsimulator [39] ..... 20
Figuur 6 – Een militaire vrachtwagensimulator [40] ..... 21
Figuur 7 – Het bedienen van een robotarm bij NASA [5] ..... 21
Figuur 8a – Een moleculaire manipulator [6] ..... 22
Figuur 8b – Een nanomanipulator systeem [6] ..... 22
Figuur 9 – Een spelcontroller met haptische feedback ..... 23
Figuur 10 – Een arcadespel met gespecialiseerde invoerapparaten ..... 23
Figuur 11 – Een virtuele omgeving die gebruik maakt van haptics ..... 28
Figuur 12 – Een sphere bouding volume [41] ..... 30
Figuur 13 – Een cylinder bounding volume [41] ..... 30
Figuur 14 – Een AABB [41] ..... 31
Figuur 15 – Een OBB [41] ..... 31
Figuur 16 – Een MBR [41] ..... 31
Figuur 17 – Een 3-DOP [41] ..... 32
Figuur 18 – De top-down methode [30] ..... 33
Figuur 19 – Een recursieve partitionering met k-DOP’s [32] ..... 33
Figuur 20 – 2 voorbeelden van filtered edge collapse [25] ..... 34
Figuur 21 – Het testen op botsingen met behulp van OBB’s ..... 34
Figuur 22 – 2 intersecterende spheres [26] ..... 35

Figuur 23 – Het genereren van een BSP-tree [45] ..... 36
Figuur 24 – Het toepassen van de hashfunctie op de AABB's van een tetrahedron [47] ..... 37
Figuur 25 – Het genereren van een LDI [48] ..... 39
Figuur 26 – De 3 stappen van het LDI-algoritme [48] ..... 40
Figuur 27 – De node/core-structuur van OpenSG [42] ..... 42
Figuur 28 – De penalty-based methode ..... 45
Figuur 29 – Het godobject ..... 45
Figuur 30 – De proxy methode ..... 46
Figuur 31 – Een laagmodel voor de OpenHaptics Toolkit [33] ..... 52
Figuur 32 – Het OpenHaptics laagmodel met H3D [33] ..... 54
Figuur 33a – De PHANToM Omni [34] ..... 64
Figuur 33b – De PHANToM Desktop [34] ..... 64
Figuur 33c – De PHANToM PREMIUM [34] ..... 64
Figuur 34a – De Omega 3 [35] ..... 65
Figuur 34b – De Delta 3 [35] ..... 65
Figuur 34c – De Delta 6 [35] ..... 65
Figuur 35 – De Novint Falcon [36] ..... 65
Figuur 36 – Een kleine scene gemaakt met OpenGL en OpenHaptics ..... 68
Figuur 37a – Een kubus ..... 69
Figuur 37b – Een kegel ..... 69
Figuur 37c – Een cilinder ..... 69
Figuur 37d – Een sphere ..... 69
Figuur 38 – Een trianglemesh, zowel grafisch als haptisch gerenderd. De virtuele manipulator wordt met een cirkel geaccentueerd ..... 69
Figuur 39 – Enkele spheres voorzien van bepaalde haptische materiaaleigenschappen ..... 70

Figuur 40 – Een voorbeeldschema van de integratie van SensAble OpenHaptics in een bestaande OpenGL applicatie ..... 71

Figuur 41 – Een testscene gemaakt in CHAI 3D ..... 73

Figuur 42 – De haptische cursor in CHAI 3D ..... 74

Figuur 43a – Een object voor interactie ..... 75

Figuur 43b – Een object na interactie ..... 75

Figuur 44 – Een voorbeeldschema van een scene gerenderd met CHAI 3D ..... 75

Figuur 45 – Een scene opgesteld in X3D voor H3D ..... 77

Figuur 46 – De scene waarvan het vliegtuigje ingelezen werd als een trianglemesh ..... 78

Figuur 47 – Het resultaat van een reeks acties in X3D gekoppeld aan een pythonscript ..... 79

Figuur 48 – De werking van H3D met X3D en pythonscript ..... 79

Figuur 49 – Een scene voorzien van haptics met HAL ..... 81

Figuur 50 – Een scene voorzien van een polymesh ..... 82

Figuur 51 – Een applicatie gebruikmakend van HAL ..... 83

Figuur 52a – SensAble OpenHaptics ..... 92

Figuur 52b – CHAI 3D ..... 92

Figuur 52c – SenseGraphics H3D ..... 92

Figuur 52d – HAL ..... 92

Figuur 53 – De cpu load voor de verschillende API’s in scenario 1 ..... 92

Figuur 54 – De memory footprint van de verschillende API’s in scenario 1 ..... 93

Figuur 55 – Het timen van de haptische loop in scenario 1 ..... 93

Figuur 56a – SensAble OpenHaptics ..... 94

Figuur 56b – CHAI 3D ..... 94

Figuur 56c – SenseGraphics H3D ..... 94

Figuur 56d – HAL ..... 94

Figuur 57 – De cpu load voor de verschillende API’s in scenario 2 ..... 95

Figuur 58 – De memory footprint van de verschillende API's in scenario 2..... 95

Figuur 59 – Het timen van de haptische loop in scenario 2..... 96

Figuur 60a – SensAble OpenHaptics ..... 97

Figuur 60b – CHAI 3D ..... 97

Figuur 60c – SenseGraphics H3D..... 97

Figuur 60d – HAL ..... 97

Figuur 61 – De cpu load voor de verschillende API's in scenario 3..... 98

Figuur 62 – De memory footprint van de verschillende API's in scenario 3..... 99

Figuur 63 – Het timen van de haptische loop in scenario 3..... 100

Figuur 64 – De HAL Device classe en zijn kinderen..... 103

Figuur 65 – Het laagmodel na de Haptik Library integratie ..... 104

Figuur 66 – De nieuwe versie van het klassediagram ..... 105

Figuur 67 – De calibratie van de Haptik Library..... 106

Figuur 68 – Een schematisch voorbeeld van de werking van het pseudodevice door middel van een keyboard ..... 107

Figuur 69 – Een schematisch voorbeeld van de werking van het pseudodevice door middel van een spacemouse..... 107

Figuur 70 – Een schematisch voorbeeld van de werking van het pseudodevice met het grafisch front-end ..... 108

Figuur 71 – Het grafisch front-end voor het pseudodevice..... 109

Figuur 72 – De visualisatie van de krachtvector..... 109

# Lijst met tabellen

Tabel 1 – De featurelisting per haptische toolkit ..... 89

Tabel 2 – De verschillen in memory footprint voor scenario 1 en 2 ..... 95

Tabel 3 – De verschillen in duur van de haptische loop voor scenario 1 en 2 ..... 96

Tabel 4 – De verschillen in cpu load voor scenario 1 en 3 ..... 98

Tabel 5 – De verschillen in memory footprint voor scenario 1 en 3 ..... 99

Tabel 6 – De verschillen in duur van de haptische loop voor scenario 1 en 3 ... 100

## Hoofdstuk 1

# Inleiding

Haptisch renderen is het realtime modelleren van krachten als reactie op de handelingen van een persoon in een virtuele omgeving, en dus bijgevolg een interactieve activiteit. Jarenlang was haptisch renderen een moeilijke kwestie, want net als bij de computer graphics worden er bepaalde eisen gesteld qua prestatie. Met de enorme vooruitgang in kloksnelheid worden er echter steeds meer toepassingen mogelijk. Haptics worden tegenwoordig in veel toepassingsdomeinen gebruikt en in een aantal hiervan zijn ze zelfs een standaard geworden. Door het openbloeien van al deze nieuwe markten zijn er ook steeds meer mensen mee bezig, zo zien we momenteel tal van haptische API's. Maar niet alleen op het vlak van software wordt er vooruitgang geboekt, ook op het vlak van hardware. Er zijn reeds tal van apparaten in omloop.

In deze thesis trachten we een beeld te scheppen van wat we met haptics kunnen doen, in welke domeinen ze van nut zijn en in welke domeinen er wordt getracht ze toe te passen. Tijdens deze literatuurstudie zullen we ook de verschillende algoritmes bespreken die van toepassing zijn bij haptics. We geven een overzicht van de belangrijkste hardware, maar we zullen bijzondere aandacht geven aan de beschikbare API's om haptics toe te passen. Eveneens zullen we in verscheidene van deze API's een implementatie maken om zo beter te kunnen concluderen wat de waarde van die bepaalde API is.

Tenslotte zullen we aan de hand van deze evaluatie een bestaande haptische API uitbreiden.

**Deel I**

# Literatuurstudie

# Introductie

De literatuurstudie van deze thesis begint met een bespreking van de toepassingsdomeinen van haptics. Er zijn domeinen waarin haptics reeds lang worden gebruikt, maar er zijn er ook waarin er net met hun gebruik gestart wordt. In deze thesis wordt getracht een beeld te scheppen van de verschillende gebruiken van haptics en de voordelen die ze in die bepaalde sector te bieden hebben. Zo zullen we ontdekken dat in sommige sectoren haptics als standaard worden aanzien, maar dat er ook tal van andere domeinen bestaan waarin het gebruik van haptische apparaten tot aanzienlijke voordelen kan leiden.

Vervolgens zullen we bespreken hoe haptics bij de mens worden aanzien. Om een goed beeld te kunnen scheppen van de vele haptische technologieën dienen we eerst een goed begrip te hebben van hoe het menselijk lichaam deze gevoelens kan waarnemen. Niet alleen wordt er besproken welke soorten van haptics we kunnen waarnemen, maar ook de capaciteiten en de grenzen van het menselijke gevoelssysteem.

Na bespreking van hoe de mens haptics zal interpreteren, zal de thesis een overzicht geven van enkele belangrijke technieken en algoritmen die momenteel gebruikt worden. Zo zal er duidelijkheid ontstaan omtrent de eisen waaraan haptics moeten voldoen om zodanig door de mens als echt te worden ervaren. Door dit verband zal ook duidelijk te zien zijn welke prestaties er worden gevraagd van de gebruikte hard – en software.

De lezer zal vanaf dit punt een beeld hebben van wat de onderliggende technieken op vlak van haptisch renderen zijn, de thesis zal vanaf dan de bestaande softwarepakketten aanhalen. Meer bepaald zal de thesis een overzicht geven van de beschikbare API’s voor haptisch renderen. Bij dit overzicht zullen verscheidene eigenschappen van de besproken API’s worden aangehaald.

De laatste sectie van de literatuurstudie zal een overzicht geven van de beschikbare hardware die samen met deze API’s voor een kunstmatig haptisch systeem zorgen. Tenslotte zal er een conclusie worden getrokken van de huidige stand van zaken op het vlak van haptische technologie.

## Hoofdstuk 2

# Toepassingsdomeinen

Om het gebruik van haptics te kunnen verantwoorden zijn er in de eerste plaats duidelijke doelgroepen voor de technologie nodig. In de volgende secties zullen de toepassingsdomeinen van haptics besproken worden, evenals het nut van de haptische interface in dat bepaalde domein.

## *Medische sector*

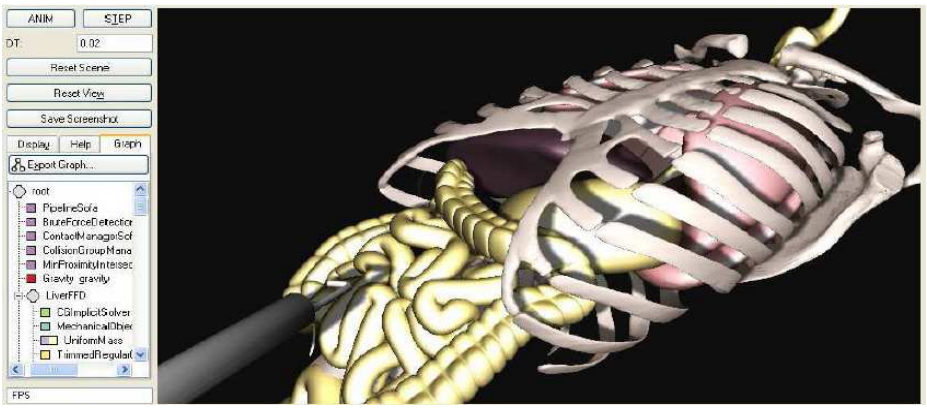
### Chirurgie

In [1] zien we dat een apparaat genaamd het “Da Vinci surgical robotic system” [Figuur 1] succesvol gebruikt wordt om operaties uit te voeren. Een dergelijk apparaat bestaat uit drie delen. Allereerst is er het invoerapparaat: dit deel bestaat uit een reeks gekoppelde manipulatoren die zeer precies de acties van de operator zullen digitaliseren. De operator aan deze manipulatoren beschikt boven zijn handen over een projectiescherm dat zorgt voor een optimale mapping van wat hij doet met zijn handelingen en wat er op de uitvoersite gebeurt. Voorts heeft dit apparaat nog een tweede console waarop een assistent de eindmanipulatoren kan aanpassen. Zo kan er bijvoorbeeld een speciaal apparaat gebruikt worden - zoals een scalpel – voor incisies, terwijl er een ander apparaat gebruikt zal worden voor het afbinden van bloedvaten. Eveneens zal deze operator de door het “Da Vinci apparaat” gebruikte materialen digitaal invoeren. Zo zal de chirurg een ander gevoel krijgen afhankelijk van het weefsel waar hij mee werkt. Er kan zo ook bijvoorbeeld een tastbaar onderscheid worden gemaakt tussen verschillende soorten garen voor het hechten van bepaalde wonden. Het gebruik van een dergelijk apparaat heeft tal van voordelen, zo zullen er onder andere vanwege het gebruik van kleine mechanische eindmanipulatoren veel minder grote incisies gemaakt moeten worden. Enkele voorbeelden hiervan zijn endoscopische, endovasculaire en laproscopische operaties. Bij deze operaties wordt er een sonde bij de patiënt ingebracht die voorzien is van een camera. De chirurg kan dan op afstand deze sonde manipuleren en zo dus veel grotere ingrepen vermijden. Dankzij deze technieken is het onder andere mogelijk om een volledige cardiologische operatie endoscopisch [1] uit te voeren. Een andere mogelijkheid

van deze technologie is dat er technisch gezien geen grenzen meer bestaan tussen de patiënt en de chirurg. De eindmanipulator kan bij een patiënt op een geheel andere locatie staan dan de console voor de operatoren.


**Figuur 1: Een Da Vinci chirurgische robot [38]**

Vorig voorbeeld betrof een echte operatie, maar er kan ook een operatie gesimuleerd worden door geen fysieke eindoperator te voorzien [Figuur 3]. Op deze manier kan een virtuele patiënt worden gegenereerd [Figuur 2] waarop de chirurgische ingreep zal plaatsvinden. Ook naar dit soort virtualisaties wordt er veel onderzoek verricht. Er bestaan ondertussen al tal van open frameworks, zoals bijvoorbeeld SOFA [2]. Zulke frameworks trachten een open platform te creëren waarop objecten onderling uitgewisseld kunnen worden. De betreffende objecten kunnen variëren van organen met hun bijhorende parameters tot nieuwe algoritmes voor bepaalde procedures.

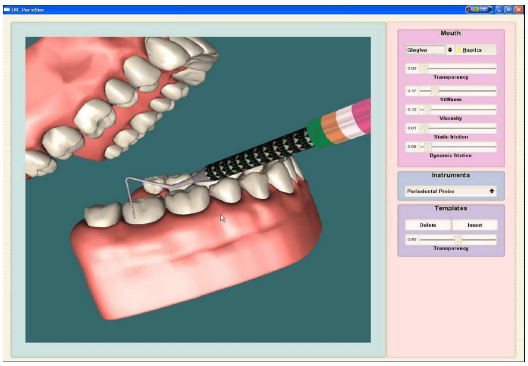



**Figuur 2: Het SOFA Framework [2]**

**Figuur 3: Een laproscopische simulator [37]**

## Tandheelkunde

Ook in de tandheelkunde wordt er veel gebruik gemaakt van robotica en haptische feedback. Volgens [3] is een der grote voordelen de mogelijkheid tot hergebruik van de haptische simulatoren. In tegenstelling tot het oefenen op sculpturen en zelfs patiënten, kan op een simulator exact hetzelfde testscenario telkens opnieuw gemanipuleerd worden, tot de techniek volledig beheerst wordt. In de tandheelkunde wordt de simulator niet enkel gebruikt voor het oefenen van een bepaalde procedure [Figuur 4], maar ook om een virtuele omgeving te creëren, zodat een bepaald probleem beter geanalyseerd kan worden. Zo kan er zelfs worden voorbereid op een specifieke operatie door deze virtueel in te oefenen, alvorens de echte operatie zal plaatsvinden. Voorts laten de technieken ook toe om experimentele procedures of chirurgische werktuigen eerst virtueel te testen alvorens deze op een echte persoon toe te passen. Al deze nieuwe procedures en hulpmiddelen kunnen de patient uiteindelijk alleen maar ten goede komen.




**Figuur 4: Een tandheelkundige simulatie [31]**

## *Simulatoren*

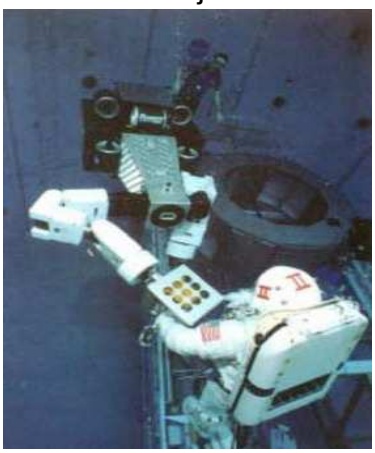
Voor vele toepassingen is het gebruikelijk dat ze eerst gesimuleerd worden vooraleer ze daadwerkelijk worden uitgevoerd. Een goed voorbeeld hiervan is de vliegtuigindustrie. Zo is het tegenwoordig gebruikelijk voor een piloot in opleiding om eerst zijn training in een simulator te doorlopen. Deze simulatoren kunnen tal van vormen hebben. Zo kan er gebruik worden gemaakt van een normale desktopcomputer om te trainen op een bepaald type van vliegtuig, zonder enige vorm van speciale invoerapparaten of haptische terugkoppeling. Een zeer bekend softwarepakket hiervoor is Microsoft Flight Simulator [4], dat zowel amateuristisch als professioneel in vliegtuigsimulaties gebruikt wordt. Deze software kan samen met enorm geavanceerde vliegtuigsimulatoren gebruikt worden om het vliegverloop van een echte commerciële lijnvlucht na te bootsen. Deze vliegtuigsimulatoren [Figuur 5] bestaan uit een zo realistisch mogelijke cockpit met visuele beeldprojectie. Dit geheel wordt dan gemonteerd op een stel pneumatische armen, die zorgen voor een zo realistisch mogelijke nabootsing van een echte vlucht. Deze simulatoren zijn weliswaar zeer duur, maar als men er rekening mee houdt dat er anders met een echt toestel gevlogen zou moeten worden, zal dit uiteindelijk op langere termijn goedkoper uitkomen. Ook kunnen er door middel van deze simulatoren bepaalde noodsituaties nagebootst worden die anders uiterst gevaarlijk zouden kunnen zijn. Voorbeelden hiervan zijn kritieke technische mankementen tijdens de vlucht. Naast dit voorbeeld bestaan er natuurlijk ook andere simulatoren dan enkel deze voor commerciële lijnvluchten. Er zijn ook simulatoren voor de scheepvaart en voor baanvoertuigen [Figuur 6].


**Figuur 5: a: (Links) Een basis vliegsimulator met MSFS b: (Rechts) Een pneumatische vliegsimulator [39]**

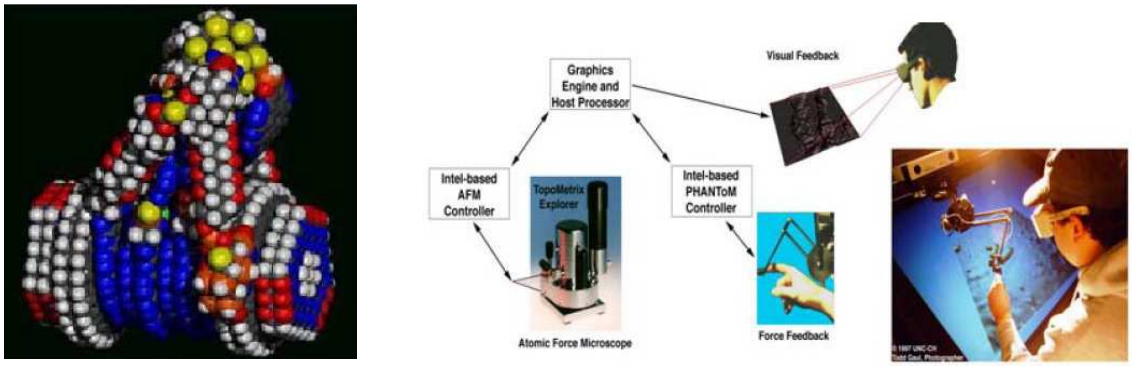
**Figuur 6: Een militaire vrachtwagensimulator [40]**

## *Teleoperaties*

We zagen net dat bij medische ingrepen reeds de mogelijkheid bestaat om deze op lange afstand uit te voeren. Maar dit soort technologieën wordt reeds in andere sectoren succesvol toegepast. Zo maakt NASA [5] gebruik van haptische manipulatoren wanneer op afstand onderhoud moet uitgevoerd worden aan een satelliet. Dit om ervoor te zorgen dat de satelliet niet beschadigd kan worden met de robotarm [Figuur 7]. In [6] wordt er zelfs gesproken over de mogelijkheid om haptics en teleoperaties te combineren om het gebruik van nanorobots succesvol toe te kunnen passen. Haptics zouden in deze wereld goed van pas komen omdat de fysische eigenschappen op moleculair niveau anders zijn dan deze van de normale wereld. De technologie zou, indien men ze zou realiseren, nieuwe types van exploratie mogelijk kunnen maken. Zo zouden bijvoorbeeld biologische weefsels via deze telepresence verkend kunnen worden en er zouden ook op deze manier zelfs wijzigingen op aangebracht kunnen worden. Dit is weliswaar nog toekomstmuziek, maar de ontwerpen voor de moleculaire manipulator en de designfases voor de haptische interface zijn reeds in ontwikkeling [Figuur 8].



**Figuur 7: Het bedienen van een robotarm bij NASA [5]**




**Figuur 8: Links: Een moleculaire manipulator Rechts: Een nanomanipulator systeem [6]**

## *Entertainment*

In de sector van het computerentertainment zien we ook dat haptics en krachtterugkoppeling meer en meer de regel worden. In [7] wordt besproken hoe er op dit moment haptische apparaten gebruikt worden in deze branche. Er wordt hier een onderscheid gemaakt tussen het materiaal waarover de gewone gebruiker thuis beschikt en de apparaten die we enkel in speciaalzaken terugvinden. Zo zien we dat momenteel vrijwel iedere controller [Figuur 9] van een spelconsole beschikt over een trilfunctie [8]. Alle software die voor die bepaalde console wordt gemaakt, kan gebruik maken van deze extra functionaliteit. Niet alleen blijkt dit de spelervaring te versterken bij de speler, maar ook blijkt in [9] dat het gebruik van haptische apparaten meer mensen toegang kan geven tot een spel. Zo is het hier gerealiseerde spel ook te spelen door visueel gehandicapten. Voor mensen zonder handicap leidde het spelen met haptische feedback zelfs tot het behalen van betere scores. Deze apparaten stellen de ontwikkelaars ook in staat om nieuwe, creatieve controleschema's toe te passen. Zo wordt in [10] haptische feedback gebruikt om de kracht te bepalen waarmee de bal geslagen zal worden. Naarmate een bepaalde knop langer wordt ingedrukt, zullen de vibraties langzaam toenemen om de speler een beter idee te geven van de kracht die hij zal uitoefenen op de bal. Uit [8] zien we ook dat het grootste deel van de gebruikers van een spelconsole de haptische functies ervaart als: fijn om een spel dat ermee voorzien is te spelen, een beter gevoel krijgt van zulk spel, het meer realistisch lijkt om te spelen en een betere controle heeft van de acties in het spel.
Anderzijds wordt in [7] ook nog het onderscheid gemaakt tussen de standaard spelconsoles en de apparaten in een speelhal [Figuur 10]. Deze laatste groep is – door haar commerciële aard - in staat om meer gebruik te maken van duurdere haptische manipulatoren. Deze types van apparaten proberen hun invoerapparaat zodanig vorm te geven dat het overeenkomt met het object uit de echte wereld. Door hier dan nog een veel sterkere krachtterugkoppeling op toe te passen wordt de ervaring voor de speler des te rijker. We zien deze toepassingen in verschillende pretparken opduiken.


**Figuur 9: Een spelcontroller met haptische feedback**


**Figuur 10: Een arcadespel met gespecialiseerde invoerapparaten**

## *Kunst*

Ook in de kunstsector is het gebruik van haptics steeds meer aanwezig. De discipline van sculpteren kan tegenwoordig virtueel worden uitgevoerd met digitale apparaten. Een groot voordeel hiervan is dat deze digitale 3D-objecten kunnen aanwend worden in bepaalde toepassingen. Zo kunnen de gesculpteerde modellen worden aangewend als digitale content in een videospel.

## *Conclusie*

Voorbeelden uit de voorgaande toepassingsdomeinen leren ons dat haptische apparaten reeds alom aanwezig zijn in ons dagelijks leven. Op professioneel vlak uit zich dat vooral in veiligheid en specifieke training, terwijl de gemiddelde persoon eerder met de ontspannende en de lucratieve apparaten in contact zal komen. Ook zien we dat haptische apparaten ingezet kunnen worden op nieuwe technologische domeinen waartoe anders geen toegang mogelijk zou zijn, denken we bijvoorbeeld maar aan de sector van de nano-operaties.

## Hoofdstuk 3

# Haptics bij het menselijk lichaam

In hoofdstuk 4 worden tal van algoritmes besproken, en om een idee te kunnen scheppen van de prestatie-eisen bij deze algoritmes dienen we eerst te bepalen welk soort gevoelens de mens kan waarnemen en wat de grenzen van het menselijk lichaam hieromtrent zijn. Daarom wordt in deze sectie besproken welke soorten gevoelens ons lichaam kan onderscheiden en aan welke eigenschappen de kunstmatige vibraties dienen te voldoen.

### *Haptics bij de mens*

Gevoel kan worden ingedeeld in drie afzonderlijke gebieden:

- Krachtterugkoppeling
- Tastbare feedback
- Haptics

Krachtterugkoppeling is de discipline die ons gewaar maakt van krachten, ook wel het kinestatische systeem genoemd. Het systeem wordt bepaald door gevoelsensoren in de spieren, in de pezen en in de gewrichten. Om realistische krachten te genereren dienen we de gemiddelde krachten van de verschillende gewrichten te bepalen. Daar de mens meestal interageert met zijn handen, zijn de vingers zeer belangrijk voor deze krachtgevoelens.
Krachten op de vingers moeten minder zijn dan 30-50 Newton, de middenvinger, de wijsvinger en de ringvinger kunnen respectievelijk 7 Newton, 6 Newton en 4,5 Newton uitoefenen.
Tastbare feedback maakt de mens gewaar van wat hij precies voelt. Dit niet zozeer qua krachten, maar op vlak van texturen, druk en temperatuur. Dit wordt ook het cutaan systeem genoemd en het wordt bepaald door gevoelsensoren in de huid.
De vingers zijn het meest gevoelig, zij hebben aan de toppen gemiddeld 135 gevoelsensoren per vierkante centimeter. Deze sensoren kunnen voor het aanvoelen van texturen tot 10.000hz aan vibraties waarnemen maar zijn het meest gevoelig rond 230hz. Dit geldt weliswaar alleen indien dit een enkel signaal

betreft. Twee simultane signalen kunnen door onze vingers worden onderscheiden tot aan 320hz per signaal. Bij hogere frequenties merken onze vingers de onderlinge verschillen tussen de signalen niet meer.
Het haptische systeem gebruikt zowel krachtterugkoppeling als tastbare feedback, maar met het verschil dat het geassocieerd wordt met een actieve procedure.
Het haptische systeem zelf is onderverdeeld in twee aparte systemen, Deze zijn:

- Het motorisch gedeelte
- Het sensorgedeelte

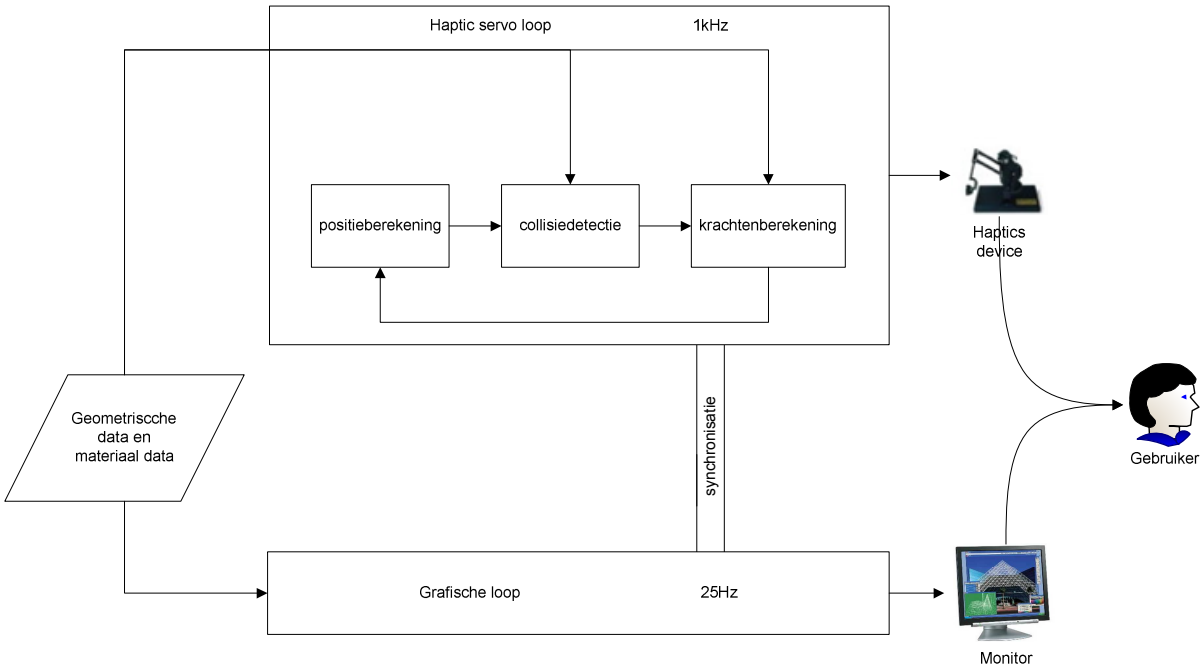
Deze twee systemen werken ook altijd samen.
Haptics kunnen op twee verschillende manieren worden waargenomen. Dit kan op een actieve en op een passieve manier. Wat actieve haptics betreft zal de persoon in kwestie zelf instaan voor de acties en het gevoel dat daardoor ontstaat, terwijl bij passieve haptics een derde zal instaan voor de waargenomen gevoelens. Met andere woorden, de persoon kan zich laten leiden door een derde maar toch de haptics waarnemen.
We dienen eveneens op te merken dat alle haptische systemen meestal sterk samenwerken met het visuele systeem. Voor een optimale gewaarwording dienen beelden en krachten gecombineerd te worden. Het visuele systeem is hierbij dominant. Dit wil zeggen dat als we visueel denken aan een grens te zitten, ons haptische systeem dit niet verder zal proberen.

De eigenschappen van ons lichaam zorgen ervoor dat er bepaalde eisen zijn aan een kunstmatig haptisch systeem. In de volgende sectie zullen we bespreken hoe een kunstmatig systeem moet werken om een realistische gewaarwording bij de mens te genereren.

## *Een kunstmatig haptisch systeem*

Om een virtuele scene zo realistisch mogelijk voor te stellen, moet er aan een aantal eisen worden voldaan. De scène moet niet alleen visueel gerenderd worden, maar moet ook zodanig in elkaar zitten dat er krachten op berekend kunnen worden. Zo moet er van alle objecten volumetrische en positionele data beschikbaar zijn. Deze data moet beschikbaar zijn voor zowel de haptische berekeningen als voor de grafische berekeningen. Deze wordt gebruikt om op botsingen te testen en zodoende de effecten van deze botsingen te modelleren. Zo moet bijvoorbeeld bij een botsing tussen elastische objecten bepaald kunnen worden hoeveel vervorming en veerkracht er moet worden gemodelleerd, afhankelijk van de positie en de eigenschappen die op dat moment van toepassing zijn op die bepaalde objecten. De botsingsalgoritmen zijn de belangrijkste

algoritmen bij het haptisch renderen, omdat alle verdere berekeningen afhangen van deze botsingsdata.
Anderzijds moet er ook de mogelijkheid zijn om wrijving te simuleren. Waar we in de grafische wereld een type structuur kunnen herkennen door 1 polygoon te visualiseren, waarvan we de oppervlakte voorzien van een beeld van bijvoorbeeld hout, moeten we bij het haptisch renderen voor deze polygoon bepaalde algoritmes uitvoeren. Een voorbeeld van een dergelijk algoritme is dat van het “Sandpaper system” [20]. Er bestaan ook verscheidene technieken om van een grafisch beeld een voor haptics geschikt textuurveld te berekenen.
Al deze algoritmen dienen eveneens veel performanter te zijn dan zijn hun grafische tegenhangers, omdat er bij haptisch renderen - in tegenstelling tot bij het grafisch renderen - ook krachten moeten worden berekend die terug naar de hardware gestuurd moeten worden. Door de eigenschappen van het menselijk lichaam moeten al deze algoritmen nog extra performant zijn. De algoritmen moeten niet enkel correct gebeuren, maar ze moeten ook vrijwel in directe tijd worden berekend. Een kleine vertraging is reeds merkbaar voor de mens.
Eveneens dient door de samenhang van de zintuigen de haptische rendering perfect synchroon te verlopen met de grafische rendering en eventueel zelfs met audio-feedback. Dit vergt niet alleen zeer veel kracht van het systeem, maar ook van het haptische uitvoerapparaat zelf. Dit apparaat moet in staat zijn om in directe tijd krachten te kunnen genereren voor de gebruiker. Intern op de computer worden de grafische berekeningen gedaan aan 25 Hz, terwijl een apparaat zoals de PHANToM zijn krachten aan 1kHz zal moeten renderen. Deze eisen gelden dan nog enkel voor een omgeving met één enkele gebruiker. Met de opkomst van collaboratieve werkomgevingen moet deze data nog over een netwerk gestuurd worden, waaraan opnieuw zeer zware eisen gesteld zijn qua vertragingen en bandbreedte.
Een voorbeeldproces van haptisch renderen kunnen we eenvoudig voorstellen aan de hand van volgende figuur [Figuur 11].




**Figuur 11: Een virtuele omgeving die gebruik maakt van haptics**

## *Conclusie*

Na het bespreken van het menselijk systeem voor gevoelswaarneming zien we dat dit systeem zeer complex en zeer veeleisend is. Door deze eisen vergt het haptisch renderen zeer veel performantie van het computersysteem, en daarom dienen de algoritmes voor haptisch renderen zo optimaal mogelijk geïmplementeerd te worden. In het volgende hoofdstuk zullen we deze algoritmen apart bespreken.

## Hoofdstuk 4

# Algoritmen & technieken

In de voorgaande hoofdstukken hebben we zeer oppervlakkig enkele benodigde algoritmen en technieken aangekaart. Om een goed beeld te krijgen van welke algoritmen en technieken er nodig zijn voor haptisch renderen en wat hun functie is, zullen we er in dit hoofdstuk enkele bespreken.

### *Botsingsdetectie*

Er bestaan tal van algoritmen om botsingen tussen objecten waar te nemen. Deze algoritmen zijn veelal afhankelijk van het type van object waarmee gewerkt wordt in de virtuele scene.
De verschillende objecttypes kunnen bijvoorbeeld bestaan uit convexe polyhedra, nonconvexe polyhedra, polygoonsoepen, gekromde oppervlakten en natuurlijk de combinatie van verschillende van deze objecten met mekaar. De meeste efficiënte botsingsalgoritmen werken in twee stappen.
De eerste stap is het bepalen van de afstand tussen twee van deze objecten. Afhankelijk van deze afstand wordt er gekeken of de objecten met mekaar zouden kunnen botsen of niet. Door middel van de afstandbepaling kunnen dan mogelijke kandidaten voor botsingen geselecteerd worden. Na deze selectie kan de tweede stap uitgevoerd worden, de uitvoering van de meer accuratere botsingsalgoritmes.
Voor botsingsalgoritmes zijn er volgens [13] twee fundamentele soorten, de discrete methodes en de continue methodes.
De discrete methodes samplen objectbewegingen en gaan zo per frame de botsingen trachten te detecteren. Een nadeel aan deze methodes is dat door het samplen de kans kan bestaan dat een botsing gemist wordt. Weliswaar bestaan er algoritmen die de kans hierop proberen te minimaliseren, maar deze wegen zwaar door op de prestaties. Hierdoor zijn de algoritmen veel minder geschikt voor realtime haptische rendering. Erger is het feit dat dit type algoritmen gebruik dient te maken van backtracking om het eerste contacttijdstip te kunnen berekenen. Dit tijdstip is nodig voor constraintgebaseerde analytische simulaties.
De continue methodes hebben het voordeel dat ze het eerste contacttijdstip berekenen tijdens de botsingsdetectie zelf. De berekening hiervan is een onderdeel van het algoritme. Een nadeel is weliswaar dat de continue methodes in het algemeen trager werken dan de discrete methodes.

Gewoonlijk wordt er bij haptics veelal gebruik gemaakt van de discrete methodes. Om de botsingen tussen polygonen zelf te detecteren zijn er tal van algoritmen en optimalisaties mogelijk. We zullen er hier enkele bespreken; het eerste voorbeeld is gebaseerd op bounding volumes.

## Bounding volume hiërarchie

Om het principe van bounding volume hiërarchie (BVH) te kunnen bespreken moeten we eerst weten wat bounding volumes zijn. Dit is een techniek die in vrijwel alle algoritmen aangewend wordt om zo de collisiedetectie te versnellen. Bounding volumes zijn simpele lichamen waarin het echte object zich zal bevinden. Er zijn tal van verschillende bounding volumes, zoals:

- **Spheres:** [Figuur 12] De simpelste vorm, snel om botsingen tussen de spheres te berekenen, maar vrij onnauwkeurig en er is veel verspilling van ruimte. Wel moet hierbij vermeld worden dat spheres handig te gebruiken zijn omdat ze rotatie-invariant en dus snel op te bouwen zijn.

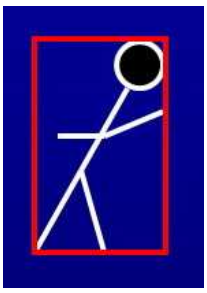
**Figuur 12: Een sphere bounding volume[41]**

- **Bounding cylinders:** [Figuur 13] Hier zal het object zich bevinden in een cylinder. Deze cylinder is geschikt voor objecten die enkel kunnen wentelen om hun verticale as; de cylinder zal zich dan ook aligneren met de verticale richting van de scene. Deze methode wordt onder andere gebruikt in videospellen voor het weergeven van mensen die rechtop staan.

**Figuur 13: Een cylinder bounding volume[41]**

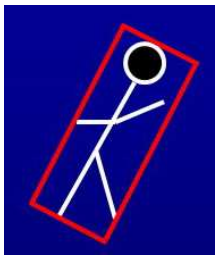
- **Axis Aligned Bounding Box (AABB):** [Figuur 14] Een simpele kubusvorm die rondom het object gemaakt wordt, de kubus ligt gealigneerd met het

assenstelsel. Dit type bounding box is het snelst gemaakt, maar is – hoewel het nauwkeuriger is dan spheres – nog steeds niet erg nauwkeurig.



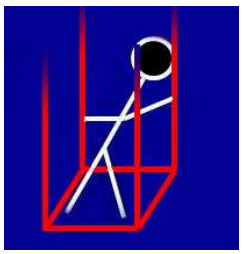
**Figuur 14: Een AABB[41]**

- **Oriented Bounding Box (OBB):** [Figuur 15] Deze bounding box is net als de AABB een kubus rondom het object, maar deze zal er zich zo dicht mogelijk rondom positioneren. OBB's geven vanzelfsprekend preciezere resultaten, maar ze zijn moeilijker te testen. AABB's hebben dan weer als nadeel dat ze bij het roteren van het model opnieuw berekend dienen te worden.



**Figuur 15: Een OBB[41]**

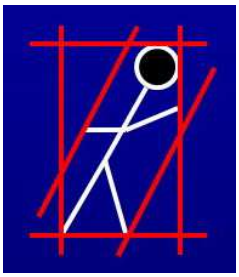
- **Minimum Bounding Rectangle (MBR):** [Figuur 16] Bij deze vorm van bounding box komt er eigenlijk op neer dat er een rechthoek zal gedefiniëerd worden in 3D. Dit kan bijvoorbeeld gebruikt worden om in een geografische sector een landoppervlakte te definiëren. Alles wat boven deze sector (oneindig) tegen dit object botst zal zich dan in dit gebied bevinden. Dit kan bijvoorbeeld gebruikt worden om te kijken in welk luchtruim een vliegtuig zich zou bevinden.



**Figuur 16: Een MBR[41]**

- **Discrete Oriented Polytope (DOP):** [Figuur 17] Dit is een veralgemening van de AABB. Een DOP is een convexe polytoop waarin het object zich zal

bevinden. Er wordt ook over k-DOP's gepraat, waarin de k dan staat voor het aantal vlakken waaruit de polytoop bestaat. Een DOP wordt gemaakt door een aantal vlakken op een afstand te nemen van het object; de oriëntatie van de vlakken is afhankelijk wat op dat moment het best geschikt is. Door deze vlakken dan te verschuiven tot ze met het object botsen, verkrijgen we aan de hand van de intersecterende vlakken een convexe polytoop. Door bijvoorbeeld te oriënteren met het XYZ-assenstelsel, en voor iedere as voor ieder uiteinde hiervan een vlak te nemen en deze naar mekaar toe te schuiven verkrijgen we op deze manier dus een 6-DOP.



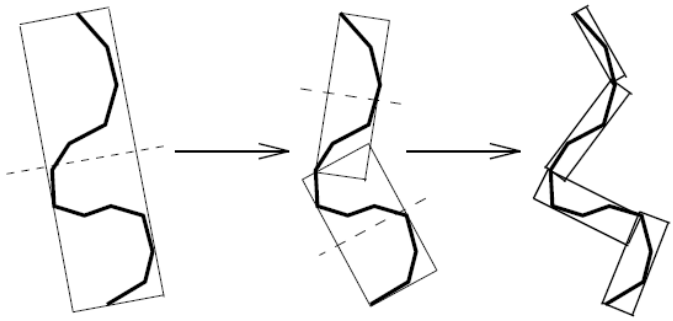
**Figuur 17: Een 3-DOP[41]**

Er bestaan verschillende methodes die gebruik maken van Bounding Volumes om zo te trachten zo weinig mogelijk rauwe polygonen met mekaar te moeten gaan testen.
Allereerst zullen we hier even kort beschrijven hoe hiërarchische culling gebeurt, door middel van het opstellen van een Bounding Volume Tree (BVT).
Deze BVT zal bounding volumes hiërarchisch representeren in een boomstructuur. Het opstellen van zo een BVT kan volgens [23] op 3 manieren;

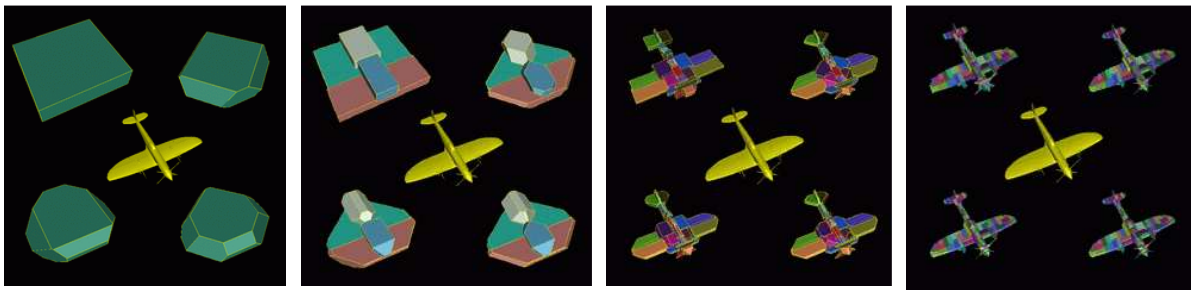
- **Repeated insertion:** Hier wordt er gebruik gemaakt van een algoritme dat de boom lokaal zal optimaliseren. De boom wordt dan opgebouwd door de objecten één voor één bij te voegen.
- **Recursieve partitionering:** Hier wordt er een algoritme gebruikt dat volumes opdeelt in subtrees. De boom wordt opgebouwd door het partitioneringsalgoritme recursief top-down uit te voeren. De uiteindelijke boom kan nadien nog door middel van dezelfde algoritmes worden aangepast.
- **Lineaire ordening:** Er wordt gebruik gemaakt van een functie die elk volume zal mappen naar een één-dimensionaal nummer. De boom kan dan worden opgebouwd als een standaard B-tree.

In [16] zien we een algoritme dat gebruik maakt van recursieve partitionering.
Volgens [16] zijn er twee manieren om een hiërarchie op te bouwen: de bottom-up methodes en de top-down methodes. De bottom-up methodes beginnen met een bounding volume voor iedere polygoon en mergen deze volumes tot de boom

opgebouwd is. De top-down methodes beginnen met een bounding volume voor alle polygonen en gaan deze recursief onderverdelen, tot uiteindelijk ieder blad onverdeelbaar is. De methode in [16] is een recursieve partitionering, het algoritme werkt top-down. Op [Figuur 18] zien we hoe de methode de langste as van de bounding box in twee zal delen. Indien deze niet te verdelen is, zal het algoritme de tweede langste as proberen te delen en indien nodig de volgende assen ook testen. Indien er geen deling mogelijk is, wordt het object gezien als onverdeelbaar. Op [Figuur 19] zien we een recursieve partitionering die gebruik maakt van DOP's.

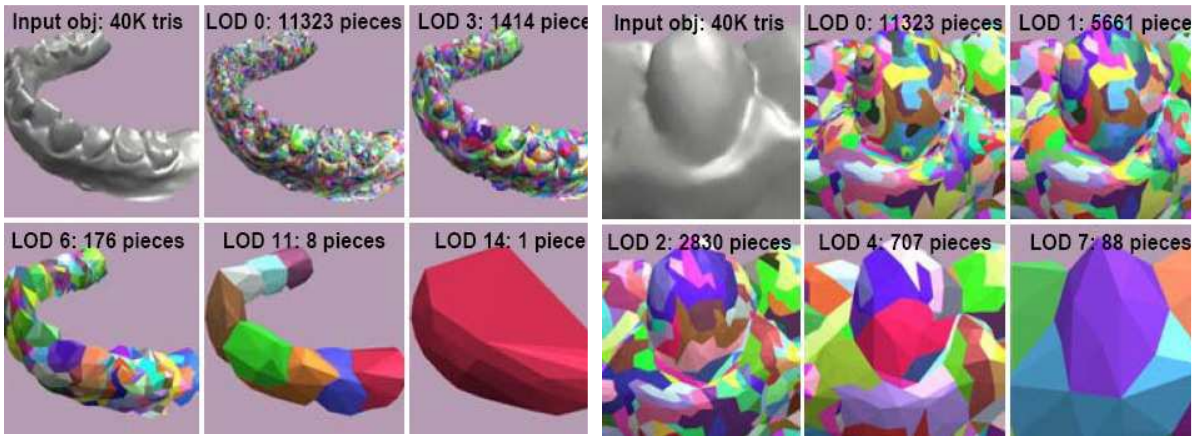


**Figuur 18: De top-down methode [30]**



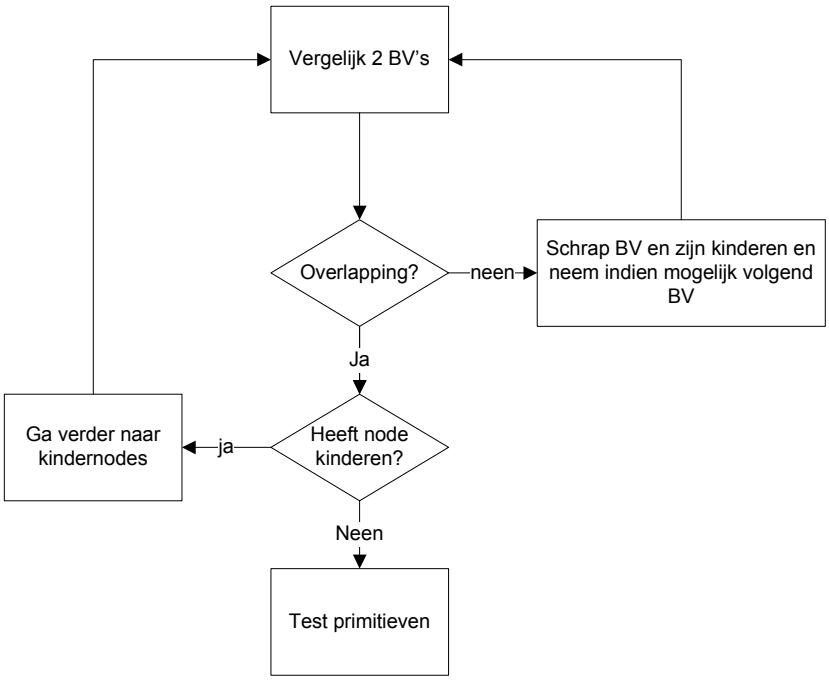
**Figuur 19: Een recursieve partitionering met k-DOP's [32]**

Er bestaan ook nog andere algoritmen om dit te doen: zo kan zelfs simpele binary space partitioning gebruikt worden [24]. In [25] wordt gebruik gemaakt van een meer geavanceerde manier die ook tot doel heeft de botsingsalgoritmes te versnellen. De methode maakte gebruik van Contact Level Of Detail (CLODs) om de objecten op te slaan in een BVH. De manier voor het hiërarchisch opbouwen maakt gebruik van een techniek genaamd Filtered Edge Collapse. Het algoritme slaagt erin het renderen van complexe modellen met een factor 100 te versnellen, maar heeft als groot nadeel dat er details verloren kunnen gaan.




**Figuur 20: 2 voorbeelden van filtered edge collapse [25]**

Om te testen op botsingen worden bijvoorbeeld de bomen van 2 objecten recursief met mekaar vergeleken. Indien er geen overlapping tussen de vergeleken bounding volumes is, kan er met de vergelijking gestopt worden en kan dus die bepaalde node - inclusief kinderen - geschrapt worden uit de test. Op deze manier wordt er dus aan optimalisatie gedaan door onnodige vergelijkingen te vermijden. Wordt er wel een overlapping gevonden, dan wordt er verder gezocht tot de precieze polygoon of polygoongroep gevonden wordt die overlappingen vertoont. Op [Figuur 21] zien we een simpele voorstelling hoe de boom zal worden afgelopen.

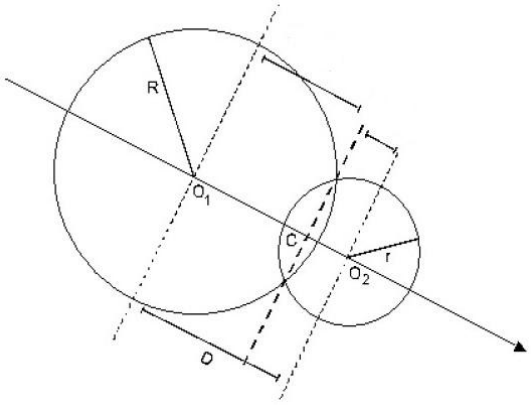



**Figuur 21: Het testen op botsingen met behulp van OBB's**

Alvorens te testen op overlappingen tussen BV's kan er nog een andere optimalisatie gedaan worden. Deze optimalisatie bestaat eruit een bepaalde waarde als threshold te nemen, bijvoorbeeld de diagonale waarde van het grootste object in de scene. Indien de afstand tussen het te testen object kleiner is

dan die van die threshold, zullen we het object verder testen. Indien de afstand groter is, moeten we dit object niet meer verder testen, want er is toch geen kans op overlapping. Deze techniek is een vorm van spatiale partitie.

Het testen van de overlappingen kan op verschillende manieren gebeuren.
Indien er gebruik gemaakt wordt van spheres kunnen botsingen en penetratiediepte zeer gemakkelijk berekend worden. In [26] zien we dat we bij sphere-objecten kunnen testen op botsingen door de afstand tussen twee spheres met mekaar te vergelijken [Figuur 22]. Indien deze afstand kleiner is dan de som van de straal van ieder van deze spheres, weten we dat deze twee objecten intersecteren.




**Figuur 22: 2 intersecterende spheres [26]**

Een andere simpele manier bestaat erin het aantal dimensies te verkleinen. Hier zal er per as (x,y,z) een lijst gegenereerd worden met de begin- en eindpunten van de overlappingen. Door deze lijst te sorteren, kunnen de punten gevonden worden waar er op alle assen overlappingen zijn; een object is immers enkel geïntersecteerd indien dit op alle assen zo is.
Zoals we net zagen wordt er bij BVH's een hiërarchie rondom de objecten opgebouwd. We zullen nu een techniek bespreken die objecten in de ruimte verdeelt.

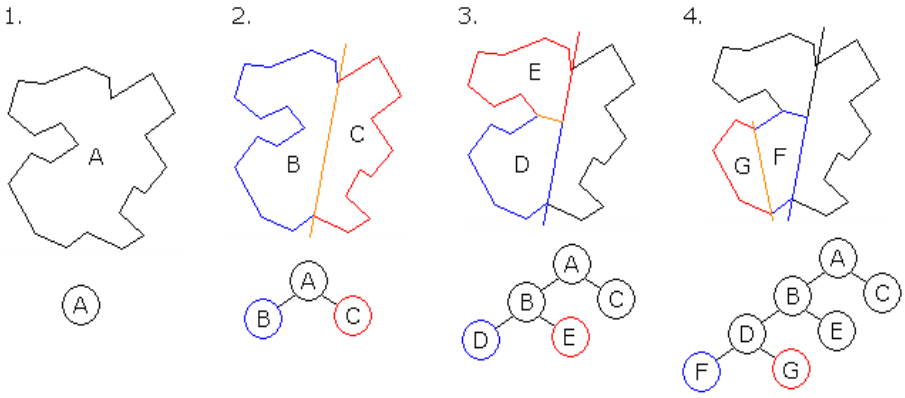
## Spatial subdivision

Bij spatial subdivision wordt de volledige ruimte waarin een geometrische scene zich bevindt, opgedeeld in niet-overlappende regionen. Net zoals bij BVH's worden ook spatial partition systemen hiërarchisch opgebouwd. Deze regionen worden namelijk geordend in een boomstructuur die beter bekend staat als een space-partitioning tree.
De meeste spatial partition systemen gebruiken vlakken om de ruimte op te delen. Afhankelijk van de positie van een object, valt het object dan in een bepaalde

zone. Een van de meest gebruikte vormen van spatial partitioning bestaat erin een ruimte recursief te gaan indelen door middel van deze vlaktes. Deze methode staat beter bekend als binary space partitioning.

### *Binary space partitioning*

Zoals we reeds zagen in de vorige sectie, verdeelt de BSP-methode een ruimte recursief in convexe sets door middel van vlakken. De hiërarchie waarin we de regionen opslaan, wordt een BSP-tree genoemd. Volgens [44] is door gebruik te maken van een BSP-tree het aantal operaties te reduceren van O( n log2 (n)), voor intersecterende objecten, tot constante tijd O(n), voor objecten die van elkaar gescheiden zijn. De enige operatie die volgens [44] nodig is voor het opstellen van een BSP-tree, is de partitionering van een convexe zone door een enkele hyperplane tot twee subregionen, die als resultaat eveneens convex zullen zijn. Op [Figuur 23] zien we hoe een polygoon wordt opgedeeld volgens de BSP-methode.

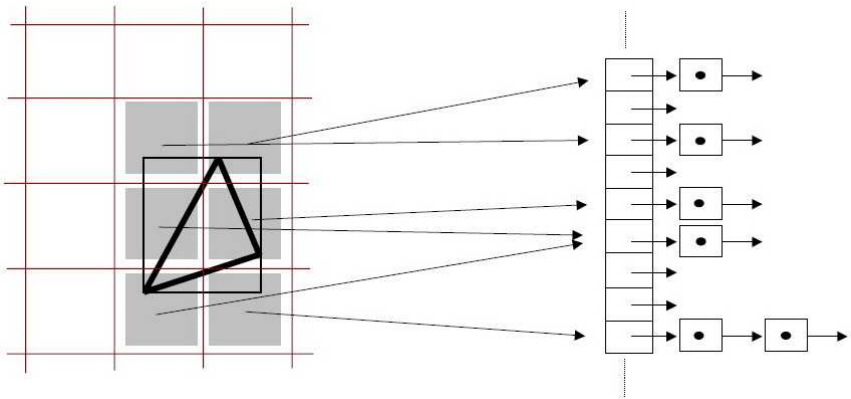



**Figuur 23: Het genereren van een BSP-tree [45]**

Volgens [46] zijn BSP-trees oorspronkelijk vooral bedoeld voor het correct sorteren van polygonen voor grafisch renderen. Maar tegenwoordig worden ze meer gebruikt voor snelle collisietesten, het verwijderen van verborgen oppervlakten en netwerkoptimalisaties. Het optimaliseren van de collisietesten is vooral te danken aan het feit dat het goedkoop is een object te positioneren in een bladnode. Enkel de bladnodes dienen namelijk getest te worden voor collisies. Er dient weliswaar opgemerkt te worden dat BSP-trees vooral geschikt zijn voor statische scenes. De reden hiervoor is dat er – voor elke toevoeging van een object aan de scene - een deel van de BSP-tree of zelfs de volledige tree herberekend dient te worden. Er bestaan verschillende algoritmen voor het genereren van een BSP-tree. Veel van deze algoritmen zullen verscheidene mogelijkheden testen om zo tot een zo goed mogelijk compromis te komen, maar het produceren van de BSP-tree is een veeleisend proces.

### *Optimized Spatial Hashing*

In [47] wordt een algoritme voorgesteld dat gebruik maakt van een hashfunctie voor het comprimeren van een potentieel oneindig groot spatiaal grid. De methode is volgens [47] heel efficiënt en maakt geen gebruik van complexe datastructuren zoals octrees of BSP-trees. Het algoritme dat wordt beschreven dient voor tetrahedrische meshes, maar kan ook worden aangepast voor andere primitieven.
Allereerst wordt $R^3$ impliciet verdeeld in kleine AABB's. Vervolgens werkt het algoritme in twee stappen. Een eerste stap bestaat erin alle vertices van alle objecten te classificeren aan de hand van deze AABB's. De tweede stap bestaat erin alle tetrahedrons te classificeren aan de hand van diezelfde AABB's. Indien een tetrahedron met zo een dergelijke 3D-cel intersecteert, dan zullen alle met deze cel geassocieerde vertices gecheckt worden op interferentie met deze tetrahedron. Het consistent verwerken van alle objectprimitieven laat het testen op botsingen toe. Indien een vertex intersecteert met een tetrahedron, dan wordt er een botsing gedetecteerd. Op deze manier kunnen ook self-collisions (botsingen van een object met zichzelf, bijvoorbeeld bij een vervormbaar object) gedetecteerd worden door na te gaan of deze vertex en de tetrahedron niet bij hetzelfde object behoren.
Voor het classificeren van de vertices en de tetrahedrons wordt er een hashfunctie gebruikt. De hashfunctie voor de eerste stap zal de gediscrete 3D posities van de vertices omzetten naar een 1-dimensionale index h. Vervolgens zal de vertex en de objectinformatie opgeslagen worden in een hashtabel op positie h. De hashfunctie voor de tweede stap zal eerst de minimum- en maximumwaardes die de AABB van een tetrahedron beschrijven discreet maken. Vervolgens worden voor alle cellen die door deze tetrahedron beïnvloed zijn hashwaardes berekend. Alle vertices met dezelfde hashwaardes zullen getest worden op intersecties. Op [Figuur 24] is te zien hoe er voor elke door een tetrahedron bezette AABB een hashwaarde wordt berekend, die nadien kan worden vergeleken met de corresponderende vertices.

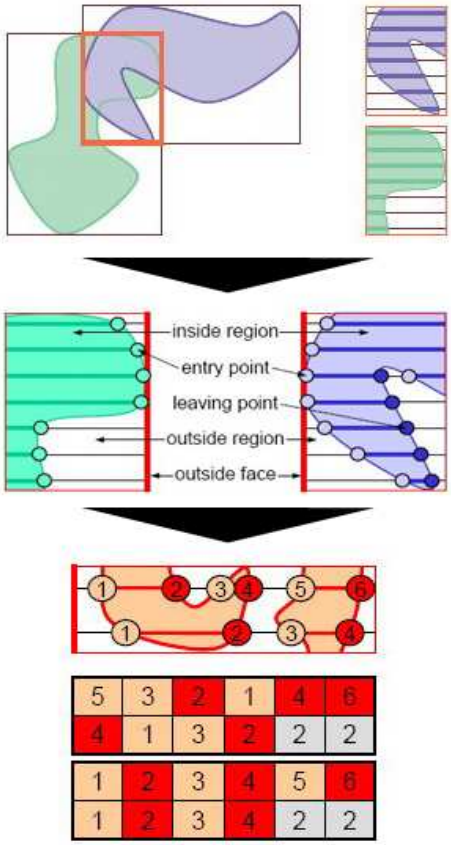


**Figuur 24: Het toepassen van de hashfunctie op de AABB's van een tetrahedron [47]**

De uiteindelijke intersectietest is gebaseerd op barycentrische coördinaten. Deze geven informatie die gebruikt kan worden voor fysisch gebaseerde collisieresponsie. Deze methode laat volgens [47] toe een scene met tot 20.000 tetrahedrons realtime te verwerken, onafhankelijk van het aantal objecten.

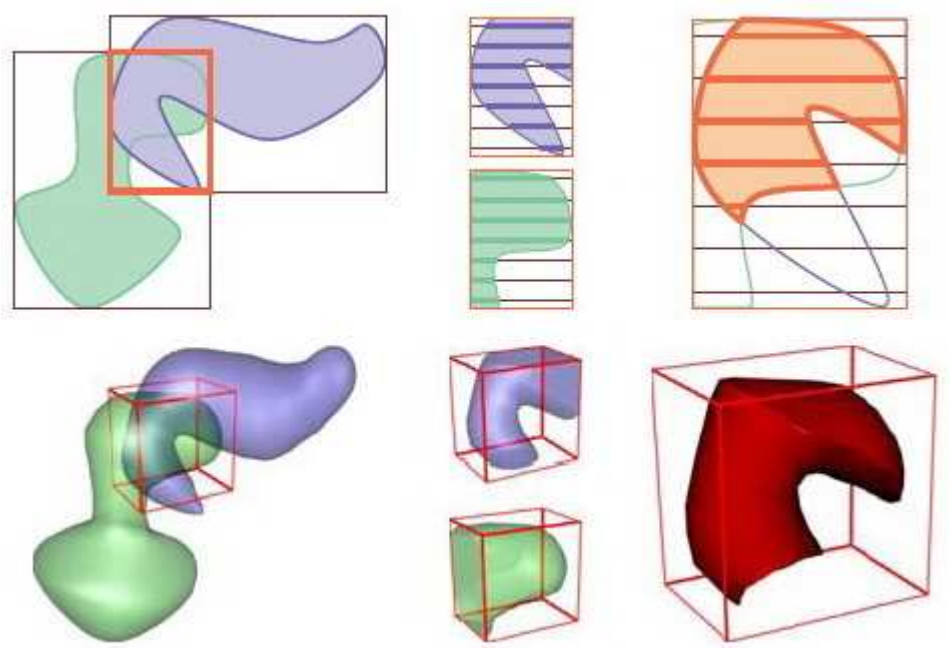
## LDI Collision Detection

In [48] wordt een botsingsalgoritme voorgesteld dat - net zoals het optimized spatial hashing algoritme - geen nood heeft aan een complexe datastructuur. Er wordt een algoritme voorgesteld dat gebruik maakt van de Layered Depth Image (LDI) compositie. Volgens [48] is het algoritme zeer makkelijk te implementeren en kan het hardwarematig versneld worden uitgevoerd door gebruik te maken van OpenGL.
In tegenstelling tot de meeste algoritmes wordt er hier geen gebruik gemaakt van de objectoppervlaktes, maar van volumetrische data. Het algoritme werkt in drie stappen.
De eerste stap bestaat uit het berekenen van AABB intersecties voor objectparen. Indien deze AABB's overlappen wordt er een Volume-of-Interest (VoI) berekend. Ondanks de eventuele onnauwkeurigheid van AABB's wordt er toch gebruik van gemaakt, aangezien ze heel snel kunnen worden gegenereerd.
De tweede stap bestaat uit het genereren van LDI's voor de VoI's van iedere AABB van de objecten. Dit zorgt voor een expliciete volumetrische representatie gediscreet volgens een voorgedefinieerde LDI resolutie. Een LDI classificeert een volume als binnen- en buitenregionen van een object. De corresponderende intersecties worden gerepresenteerd als entry point en leaving point. Deze manier van werken is te vergelijken met scan-conversie algoritmes voor het vullen van concave polygonen. Op [Figuur 25] is te zien hoe een object wordt omgezet naar een LDI. Door het opslaan van deze punten werken LDI's volgens [48] in vergelijking met andere datastructuren met veel minder geheugen.




**Figuur 25: Het genereren van een LDI [48]**

De derde stap zal de eigenlijke collisietest uitvoeren. In [48] worden er hiervoor twee soorten testen voorzien. Allereerst de intersection volume test voor het genereren van een intersecterend volume tussen twee objecten, en ten tweede de vertex-in-volume test voor het testen van individuele vertices met een object.
Op [Figuur 26] stellen we de drie stappen nog even schematisch voor. Het algoritme is zoals gezegd zeer snel te implementeren door gebruik te maken van basis OpenGL code, en biedt dankzij de LDI's een performante collisiedetectie, geschikt voor een dynamische simulatie. Het algoritme kan wel enkel werken met gesloten objecten omdat er altijd volumetrische data nodig is.



Stap 1 Stap 2 Stap 3

**Figuur 26: De 3 stappen van het LDI-algoritme [48]**

## Voronoi diagrammen

Een andere vrij recente techniek [18] voor collisiedetectie maakt gebruik van 2$^{de}$ order voronoi diagrammen. Dit algoritme doet eerst aan AABB culling en zal nadien op de overgebleven objecten een 2$^{de}$ orde voronoidiagram berekenen om interobject queries op te stellen. Nadien kunnen er op deze queries exacte primitieftesten uitgevoerd worden. Deze methode biedt een uniforme manier om aan collisiedetectie te doen in een complexe omgeving, bestaande uit verschillende vervormbare objecten.

Veel van de besproken algoritmes worden uitgevoerd door een CPU. Maar recente ontwikkelingen in de wereld van de grafische processoren (GPU's) stellen ons nu in staat ook hun kracht meer en meer aan te wenden. GPU performantie evolueert veel sneller dan die van de CPU. Vroeger konden GPU's enkel aangewend worden voor pure grafische berekeningen; hun functies waren statisch in de hardware geïmplementeerd. Maar tegenwoordig is de GPU voorzien van programmeerbare pipelines. Hierdoor kunnen algoritmen voor collisiedetectie omgezet worden naar programmeerbare shaderprocedures. Aangezien de GPU zeer veel parallelle berekeningen aankan, kan dit een zeer grote impact hebben op de performantie van collisiedetectiealgoritmen. Eveneens zijn er anderen die standaard OpenGL code gebruiken voor het berekenen van bijvoorbeeld LDI's [48] of voronoidiagrammen [18]. Deze technieken werken veel sneller dan wanneer we dit door een traditionele CPU laten berekenen. Andere technieken zoals [27] gebruiken de GPU om de vervorming van objecten te berekenen als we deze

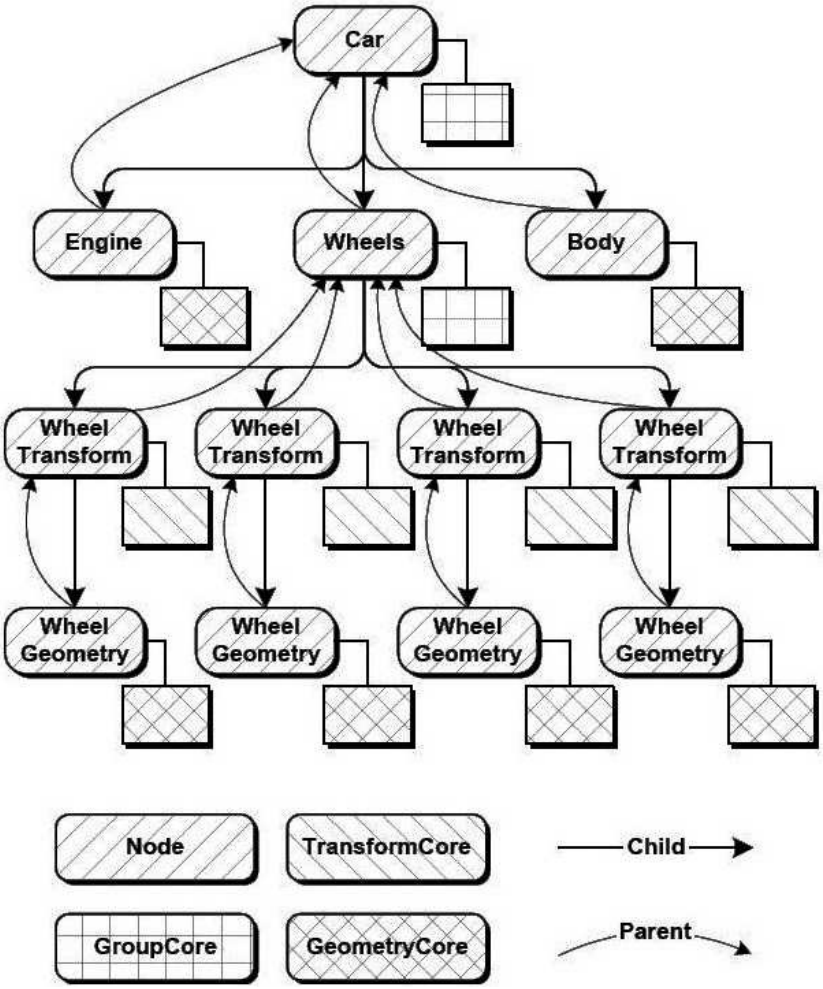
manipuleren met een haptische manipulator. Ook wordt er bij het programmeren van graphics steeds meer gebruik gemaakt van scenegraphs. Maar deze scenegraphs kunnen eveneens gebruikt worden voor haptics; meer hierover in de volgende sectie.

## *Scenegraphs*

Een scenegraph is een datastructuur voor een grafische scene. In [42] wordt een scenegraph beschreven als een datastructuur voor graphics die zijn staat behoudt. Low level grafische libraries zoals OpenGL zijn staatloos. Dit wil zeggen dat voor iedere frame alle data opnieuw moet worden doorgestuurd naar de grafische hardware. Scenegraphs laten toe de grafische scene voor te stellen als objecten. Bij scenegraphs moet de scene eenmalig gedefinieerd worden en enkel de wijzigingen van een bepaald object moeten worden doorgegeven. Scenegraphs behouden dus hun staat en zullen zo voor een zo optimaal mogelijk gebruik zorgen van de grafische hardware.
De data in een scenegraph wordt voorgesteld als een acyclische graaf. De objecten van de scene worden gerepresenteerd in de scenegraph door middel van nodes. Er zijn volgens [42] twee soorten nodes, de bladnodes en de interne nodes. De bladnodes stellen de te renderen geometrie voor, terwijl de interne nodes de graaf opdelen in logische groepen.
Het voorstellen van een scene als scenegraph heeft verschillende voordelen: zo kunnen er optimalisaties toegepast worden vanwege de hogere niveau kennis van de volledige scene. Er kunnen onder andere nodes of gehele subtrees getest worden op zichtbaarheid. De scenegraph kan eveneens geoptimaliseerd worden om per frame zo optimaal mogelijk te renderen voor een grafische pipeline. Ook op vlak van interactie heeft een scenegraph voordelen; zo kunnen picking- en collisietesten geoptimaliseerd worden.
Ook geheugenbeheer kan door middel van scenegraphs worden verbeterd. In een grafische scene komen namelijk vele elementen voor die in wezen kopieën zijn van elkaar. Neem bijvoorbeeld een auto; de rootnode kan hier bijvoorbeeld het chassis zijn, terwijl de 4 deuren of de 4 wielen eigenlijk niets meer zijn dan 4 maal hetzelfde object met een andere locatie. Scenegraphs zorgen ervoor dat deze objecten maar één maal moeten worden gedefinieerd. Op [Figuur 27] kunnen we zien hoe een scenegraph-library zoals OpenSG[1] dit aanpakt.
OpenSG werkt met nodes en cores. Nodes zijn uniek, maar dienen enkel als datacontainer. De cores kunnen meerdere malen gebruikt worden door ze te koppelen aan een node. Er bestaan verschillende cores; zo zijn er onder andere cores voor geometrie, voor transformaties en voor materialen. Op [Figuur 27] is te

[1] http://www.opensg.org

zien hoe een scenegraph van een auto wordt gemaakt. Alle onderdelen worden gedefinieerd als nodes met een core, maar de geometriecore voor een wiel kan voor ieder wiel opnieuw worden gebruikt.




**Figuur 27: De node/core-structuur van OpenSG [42]**

Scenegraphs zijn niet alleen toe te passen bij het grafisch renderen, maar ook bij het haptisch renderen. Zo kan er volgens [42][43] parallellisme toegepast worden waardoor zowel het grafisch als het haptisch renderen gebruik kan maken van dezelfde scenegraph. Het voordeel hiervan is dat de scene maar één maal - voor zowel het grafisch als haptisch renderen - gedefinieerd dient te worden.
We zullen in hoofdstuk 5 per API aangeven of deze al dan niet werkt met een scenegraph.

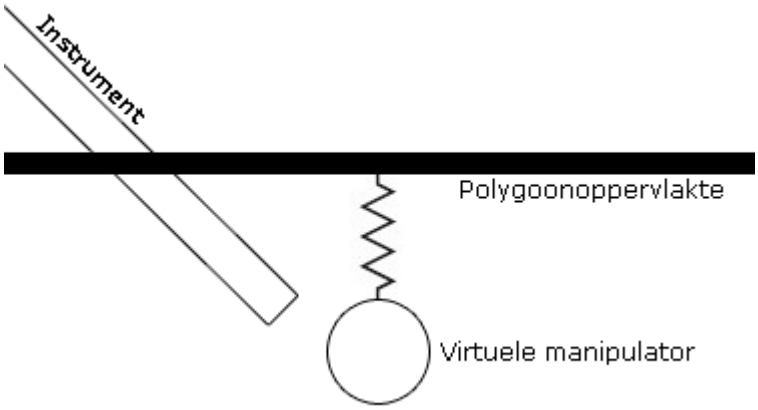
## *Haptic rigid body simulation*

Om een zo realistisch mogelijke wereld te creëren moeten we ook de fysische wereld zo reëel mogelijk simuleren. We moeten er voor zorgen dat bij het detecteren van een botsing de correcte krachten worden berekend, die dan via het apparaat zo reëel mogelijk worden gesimuleerd. Een techniek die hier zeer belangrijk voor is, bestaat uit het creëren van rigid bodies. Rigid bodies zijn objecten die we voorstellen in de virtuele wereld als onvervormbaar. De afstand tussen twee punten van een rigid body blijft constant, onafhankelijk van de krachten die erop inwerken. Om rigid bodies te simuleren bestaan er volgens [14] op dit moment drie elementaire technieken. Deze technieken zijn niet alleen van toepassing bij haptic rendering, maar ook bij grafisch renderen. De technieken zijn de volgende: De penalty-based methode, de constraint-based methode en de impuls-based methode. Ieder model maakt gebruik van verschillende contactstaten en berekent de krachten op een specifieke manier.
De penalty-based methode maakt gebruik van twee contactstaten. “Geen contact”: dit is als er een positieve afstand is tussen twee objecten, en “rustend contact”: dit is er wanneer de objecten intersecteren. Om de krachten te berekenen wordt er als het ware een veerkracht berekend voor ieder contactpunt. Als twee objecten dus intersecteren zullen ze voor alle intersecterende contactpunten krachten berekenen, zodat de objecten uit elkaar worden geduwd. Deze methode wordt op grote schaal toegepast omdat de krachten snel te berekenen zijn, en dus ook binnen de prestatie-eisen vallen die nodig zijn voor haptic rendering.
Constraint-based methodes gebruiken drie contactstaten. “Geen contact”: als er een positieve afstand bestaat tussen de objecten, “botsend contact”: als de afstand tussen de objecten nul of minder bedraagt en de lichamen op tenminste één plaats in mekaar geschoven zijn, en ten laatste “rustend contact”: wanneer de afstand tussen de objecten nul bedraagt, of wanneer de objecten van mekaar weg bewegen voor alle intersectiepunten. Voor de constraint-based methodes bestaan er twee schema’s: het event-driven schema en het time-stepping schema. Het event-driven schema is weliswaar niet geschikt voor haptics aangezien er geen gegarandeerde berekentijd bestaat.
Het time-stepping schema berekent op discrete tijdstappen de krachten afhankelijk van de botsingen die er op dat moment zijn. Omdat de berekentijd sterk afhankelijk is van het aantal botsingen, moeten er wel constraints gestabiliseerd worden om zo tot een aanvaardbare berekentijd te komen [21]. Constraint-based methodes maken gebruik van de Newton-Euler vergelijking voor beweging, onderworpen aan geometrische constraints, die ervoor moeten zorgen dat de objecten niet intersecteren. De numerische integratie van de Newton-Euler vergelijking moet gestopt worden voordat de objecten intersecteren. Bij een botsing moeten de snelheden en versnellingen van een object aangepast worden in die mate dat niet-intersecterende constraints niet worden geschonden, en de

numerische integratie opnieuw gestart kan worden. Eerst moeten er contactimpulsen berekend worden die voldoen aan de constraints, nadien zullen de contactkrachten berekend worden die resulteren in correcte versnellingen. De constraint-based methode is vooral geschikt voor puntinteractie in de virtuele wereld, maar voor rigide objecten is deze minder geschikt.
Impuls-based methodes kennen twee contactstaten. “Geen contact”: als er een positieve afstand bestaat tussen de objecten, en “botsend contact”: als de afstand gelijk is aan 0. Impuls-based technieken passen rigide constraints exact toe. Ze implementeren rustend contact als een serie microbotsingen en passen impulsen toe om objectintersectie te voorkomen. Impuls-based technieken geven in de grafische wereld aanvaardbare resultaten, maar voor haptics zijn ze niet overtuigend. Ook zijn ze cpu-intensief als er vele rustende contacten zijn. Ze maken gebruik van hetzelfde schema als time-driven constraint methodes. Met andere woorden, de numerische integratie moet worden onderbroken alvorens er intersecties plaatsvinden. Ook moeten er juiste snelheden worden berekend.
In [14] wordt nog een techniek voorgesteld die een combinatie gebruikt van de constraint-based methode en de penalty-based methode. Deze techniek berekent constraint-based impulsieve krachten bij contact en penalty-based krachten gedurende het contact. Uit de resultaten blijkt dat deze combinatie de stabiliteit en de waargenomen rigiditeit van de objecten ten goede komt, terwijl de performantie gelijk is aan bestaande technieken.

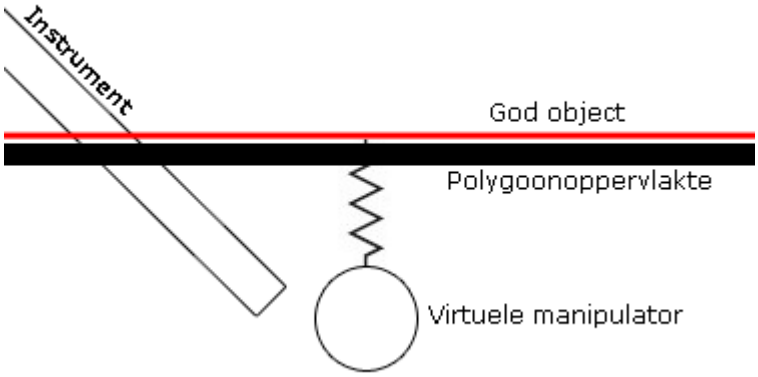
## *Surface haptics*

Een variatie op rigid body die gebruikt wordt bij haptics is die van surface haptics. Daar de gebruiker over een specifiek apparaat beschikt om zijn acties te doen, moet de virtuele manipulator ook gekoppeld worden aan dit apparaat. De gebruiker zal dan via deze hardware oppervlaktes kunnen aanvoelen zoals in de echte wereld. In [14] zien we dat er om surface haptics te renderen drie technieken zijn: de penalty-based methode, de godobject gebaseerde methode en de proxy gebaseerde methode.
Wanneer er gebruik gemaakt wordt van de penalty-based methode [Figuur 28] is de kracht volledig afhankelijk van de afstand tussen de virtuele manipulator en de dichtstbijzijnde polygoon van het gepenetreerde object. De kracht wordt hierbij berekend door het product van de afstandsvector en de stijfheidvariabele. Dit is te vergelijken met een veer die wordt opgespannen tussen de polygoon en de manipulator. Een nadeel aan deze methode is dat de kracht per frame volledig afhankelijk is van dichtstbijzijnde polygoon: dit kan tot fouten leiden indien het bijvoorbeeld een dun object is. Het zou dan zelfs mogelijk zijn door het object te prikken, waardoor de kracht totaal verkeerd zou zijn - vergeleken met wat de persoon verondersteld is te voelen.



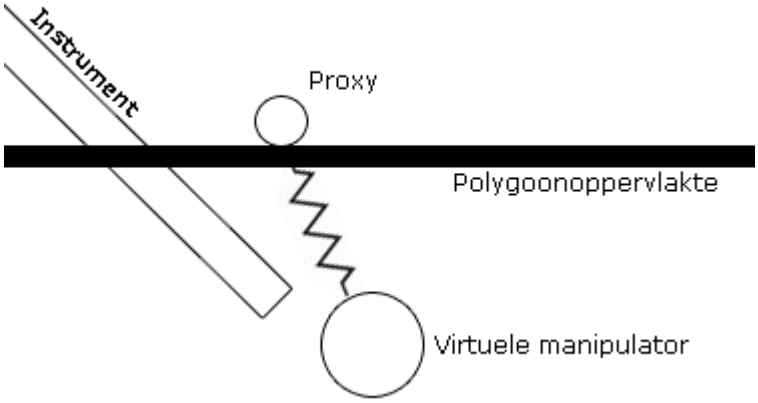

**Figuur 28: De penalty-based methode**

De godobject gebaseerde methode [Figuur 29] is ontworpen om de nadelen van de penalty-based methode te compenseren. Door gebruik te maken van een godobject is het immers niet meer mogelijk dat een manipulator doorheen een object vliegt. Het godobject werkt dus als het ware als een referentievlak tegen de polygoon waar de manipulator het dichtste bij is. Het godobject schuift mee over de oppervlakte zolang er penetratie gedetecteerd wordt, zodoende is het dus onmogelijk om door de polygoon heen te vliegen.




**Figuur 29: Het godobject**

Hierop voortbouwend is de proxy methode [Figuur 30]. Deze plaatst op het godobject nog een volume dat overeenkomt met waar de manipulator het object is binnengedrongen, deze polygoon schuift weliswaar ook mee naarmate de manipulator zich nog verder beweegt. De manipulator kan gebruikt worden om precies te berekenen hoe het oppervlakte van het object daar moet aanvoelen. De reden waarom er nog steeds gebruik gemaakt wordt van het godobject is dat er een kans bestaat dat er in het op dat moment gepenetreerde object onvolmaaktheden zitten, waardoor een enkele kleine polygoon toch nog door het object zou kunnen dringen. Vandaar dat het proxyvolume over het godobject zal glijden.




**Figuur 30: De proxy methode**

Sinds de voorgaande algoritmen werken we reeds met een Surface Contact Point (SCP), dit komt overeen met het proxyvolume. We kunnen nu deze proxy gaan gebruiken om krachten te genereren, zodat we de textuur van een bepaald oppervlakte kunnen waarnemen. Dit noemen we haptic texture rendering, en we zullen in de volgende sectie technieken hiervoor bespreken.

## *Haptic texture rendering*

Eén van de meest elementaire technieken in haptic rendering is die van texture rendering. De eerste techniek hieromtrent was die van Minsky [20]. De techniek staat bekend als het “Sandpaper systeem”. Dat geschikt was voor 2D haptische rendering van texturen. Het algoritme berekent de krachten aan de hand van een texturaal hoogteveld. De belangrijkste bijdrage van de techniek van Minsky was de hypothese dat textuurinformatie aangebracht kan worden door de krachten te laten divergeren met het contactoppervlak. Zoals in de vorige sectie vermeld, worden de krachten berekend door middel van de afstand tussen de virtuele manipulator en het proxyvolume. Het proxyvolume wordt in [22] voorgesteld als een kleine sphere die over de oppervlakte zal bewegen. Door de beweging van deze proxy te combineren met de penetratiediepte kunnen we texturen realistisch laten aanvoelen.

## *Conclusie*

Haptisch renderen is de volgende stap in de zoektocht naar het reëel nabootsen van een virtuele wereld. We zien dat er voor het renderen talloze technieken bestaan, waarvan vele technieken hun oorsprong vonden in de computer graphics. De reden hiervoor is dat er veel eerder gestart kon worden met het ontwikkelen van computergraphics dan met de ontwikkeling van haptics. De

performantie-eisen voor graphics lieten namelijk toe dat veel meer mensen toegang hadden tot de benodigde hardware. Ook zijn verschillende van de algoritmes uit de grafische wereld toe te passen bij het haptisch renderen: een voorbeeld hiervan zijn de hiërarchische bounding volume trees en andere botsingsalgoritmes. Door de hoge performantie-eisen is het dus pas sinds kort mogelijk om realistische krachten in complexe omgevingen weer te geven. Maar we zien aan de talloze algoritmen dat er ook zeer veel onderzoek gedaan wordt naar het verbeteren van deze technieken. De algoritmen worden zodanig geoptimaliseerd, dat haptic renderen goed mogelijk is, en door de continue evolutie van de CPU en vooral de GPU kunnen ook steeds complexere omgevingen gesimuleerd worden.

## Hoofdstuk 5

# Bestaande API's

Om een bespreking te kunnen maken van haptische API's dienen we eerst een beeld te scheppen van wat er op dit moment reeds beschikbaar is. We zullen hier een overzicht geven van de bestaande haptische toolkits, dit zowel voor de open source API's als de niet open source API's. De API's waarvan we een overzicht zullen geven zijn:

- Ghost SDK
- OpenHaptics Toolkit
- H3D
- Chai3D
- Haptik Library
- OpenSceneGraph haptic Library (osgHaptics)
- HAL
- Reachin API

We zullen nu per API een kort overzicht geven:

### *Ghost SDK*

Ontwikkelaar:

SensAble Technologies Incorporated

Website:

http://www.sensable.com

Huidige versie:

Ghost SDK v4

Scenegraph voor haptic rendering:

Ja

Scenegraph voor grafische rendering:

Neen, maar wel mogelijk via GhostGL

Geschikte hardware:

De GHOST SDK werkt met PHANToM apparaten die beschikken over:

- EPP IEEE – 1284 parallelle poort interface
- SensAble Technologies PCI interfacekaart voor de PHANToM premium 1.0, 1.5, 3.0

Compatibele platformen:

Win32

Silicon Graphics Incorporated workstations met IRIX 6.5

Beschrijving:

De “General Haptics Open Software Toolkit” (GHOST) werd ontwikkeld door SensAble Technologies Incorporated, het was de eerste commercieel verkrijgbare API voor haptics. De GHOST toolkit is een C++ object georiënteerde toolkit die dient om haptische scenes hiërarchisch op te bouwen in een scenegraph. In deze scenegraph worden de geometrische objecten en hun spatiale effecten hiërarchisch geordend. GHOST zorgt automatisch voor een stabiele 3D krachtterugkoppeling en zal de programmeur afschermen van de methodes om deze krachten aan de hand van de scene manueel te berekenen.

Zoals hierboven beschreven berekent GHOST automatisch de interactiekrachten tussen het Haptisch Interactie Punt (HIP) en de scene. Het HIP kan op twee manieren gebruikt worden: een eerste manier bestaat erin het te gebruiken als de fysieke locatie van het haptisch interactieapparaat binnen zijn fysieke werkruimte. De tweede manier bestaat erin het HIP te gebruiken als de computergegenereerde locatie van

het HIP gekoppeld aan de oppervlakte van geometrische objecten. Deze laatste locatie hebben we tijdens de algoritmes besproken als het Surface Contact Point (SCP). Het SCP wordt in GHOST gebruikt voor de berekening van alle oppervlakte interactie krachten.
Voorts moeten we opmerken dat GHOST weliswaar gebruik maakt van een scenegraph, maar dat GHOST deze enkel gebruikt voor het renderen van de haptische scene. Voor graphics voorziet GHOST wel het gebruik van callbacks zodat een grafische scene kan worden gemapt met de GHOST scenegraph. De gebruiker moet dus zelf zorgen voor de grafische rendering, deze staat vrijwel geheel los van GHOST. We dienen op te merken bij vorig statement dat er wel een GHOSTGL API bestaat. GHOSTGL wordt zelfs bij GHOST SDK v. 3 geleverd en stelt de gebruiker in staat de haptische scene door middel van OpenGL graphics te laten renderen. Voor zowel de haptische rendering als voor de grafische rendering dient er een lus te worden gestart.

## *OpenHaptics Toolkit*

Ontwikkelaar:

SensAble Technologies Incorporated

Website:

http://www.sensable.com

Huidige versie:

OpenHaptics Academic Edition for Windows - v2.0

Scenegraph voor haptic rendering:

Neen

Scenegraph voor grafische rendering:

Neen

Geschikte hardware:

De OpenHaptics Toolkit geeft ondersteuning voor alle SensAble PHANToM apparaten, dit van de goedkopere PHANToM Omni tot de grotere PHANToM premium apparaten.

Compatibele platformen:

Win32, Linux en Mac OS X

Beschrijving:

De OpenHaptics toolkit wordt gebruikt om haptics toe te voegen in tal van applicaties. De syntax is gemaakt om op deze van OpenGL te lijken. Het resultaat van deze feature is dat mensen die gewoon zijn van met de OpenGL API te programmeren ook sneller overweg zouden kunnen met OpenHaptics. De toolkit laat het de ontwikkelaar toe gebruik te maken van third-party libraries. De OpenHaptics toolkit bestaat eigenlijk uit drie grote onderdelen, deze zijn:

- De Haptic Device API ( HDAPI)
- De Haptic Library API (HLAPI)
- De PHANToM device drivers (PDD)

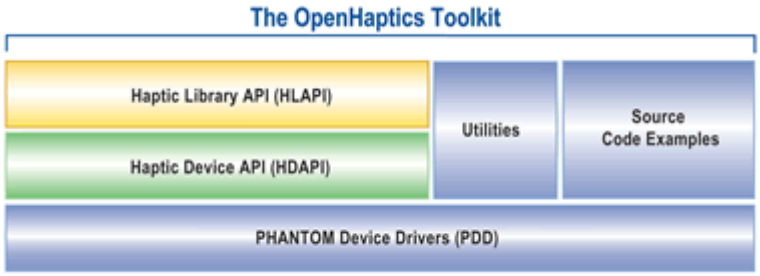
De HDAPI wordt gebruikt om ontwikkelaars toegang te geven tot de low-level functionaliteiten van het haptische apparaat. Hierdoor zijn ze in staat krachten zelf te programmeren en hebben ze toegang tot het gedrag van de

drivers. De HDAPI biedt een geoptimaliseerde interface tot alle SensAble PHANToM apparaten en is platformonafhankelijk.
De HLAPI zorgt voor high-level haptische rendering en is ontworpen om ontwikkelaars met ervaring met OpenGL vlot te laten werken met OpenHaptics. De HLAPI zorgt voor een goede link tussen het grafisch gebeuren en het haptisch gebeuren [Voorbeeld 1]. Het gebruik van de HLAPI zorgt er ook voor dat haptics op een eenvoudige manier kunnen worden toegevoegd in reeds bestaande applicaties.

```
hlBeginFrame();
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER BIT);
glColor3f(1.0, 0.0, 0.0);
hlBeginShape(HL_SHAPE_DEPTH_BUFFER, myShapeId);
glBegin(GL_POLYGON)
glVertex3f( 0.25, 0.25, 0.0);
        glVertex3f( 0.75, 0.25, 0.0);
        glVertex3f( 0.75, 0.75, 0.0);
        glVertex3f( 0.25, 0.75, 0.0);
glEnd();
hlEndShape();
glFlush();
hlEndFrame();
```

**Voorbeeld 1: Een stukje OpenGL code met in het rood de OpenHaptics code**

De PDD is een collectie van drivers waarmee alle PHANToM apparaten aangestuurd kunnen worden.
Indien we de OpenHaptics Toolkit schematisch voorstellen [Figuur 31], zullen we de PPD onderaan zetten. Deze zullen aangestuurd worden door de HDAPI en de HDAPI wordt aangestuurd door de HLAPI.
De OpenHaptics Toolkit is beschikbaar onder een commerciële licentie, maar SensAble stelt ook een versie ter beschikbaar voor academisch gebruik.




**Figuur 31: Een laagmodel voor de OpenHaptics Toolkit [33]**

### *H3D*

Ontwikkelaar:

SenseGraphics AB – Open source


Website:

http://www.sensegraphics.com/
http://www.h3dapi.org/

Huidige versie:

H3D API v1.5

Scenegraph voor haptic rendering:

Ja

Scenegraph voor grafische rendering:

Ja

Geschikte hardware:

H3D geeft ondersteuning voor alle SensAble PHANToM apparaten, dit van de goedkopere PHANToM Omni tot de grotere PHANToM premium apparaten. Ook de Force Dimension Omega.3 en de Force Dimension Delta 3DOF/6DOF worden standaard ondersteund.

Compatibele platformen:

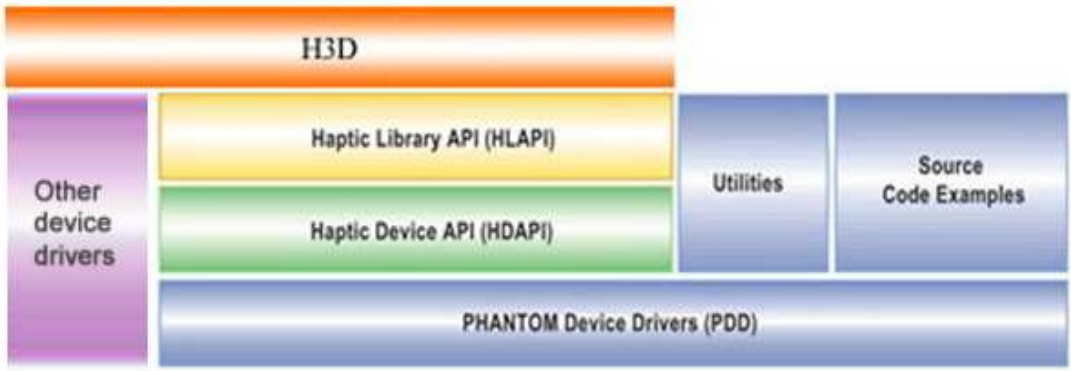
Windows XP, Linux, Mac OS X

Beschrijving:

De H3D API is een open source API die specifiek nieuwe applicaties, die nog van de grond af aan moeten worden opgebouwd, als doel heeft. De reden hiervoor is dat H3D een andere aanpak heeft dan de vorige toolkits. H3D voorziet namelijk in een unified scenegraph. Dit wil zeggen dat H3D gebruik maakt van dezelfde scenegraph voor zowel grafische rendering als voor haptische rendering. Er moet dus geen synchronisatie voorzien worden tussen het grafische gedeelte en de haptics, H3D zal dit voor de gebruiker automatiseren, de scene moet dus ook maar één maal worden opgebouwd voor beide systemen. Dit zorgt voor minder overhead en voor spaarzamer gebruik van geheugenbronnen dan bij het gebruik van aparte scenegraphs. H3D is voor het haptisch renderen wel afhankelijk van OpenHaptics, H3D is dus eigenlijk een vierde laag op de HLAPI van OpenHaptics [Figuur 32].

H3D kan ook overweg met Python en X3D; het voordeel hiervan is dat deze twee ervoor zorgen dat mensen veel gemakkelijker met H3D kunnen

werken. Mede hierdoor wordt het mogelijk snel prototypes te maken met H3D.
Voor geavanceerd gebruik kan er nog steeds C++ gebruikt worden en ook hier kunnen indien gewenst X3D-bestanden ingeladen worden voor de scene.
Het grootste voordeel van H3D is dat er snel applicaties gebouwd kunnen worden, maar een nadeel is dat het niet zomaar te integreren is in reeds bestaande applicaties.
H3D is vrij te distribueren onder de GNU PGL v2 licentie. Tenslotte willen we nog opmerken dat H3D pas sinds recent ook andere devices ondersteunt dan de SensAble apparaten. Maar deze ondersteuning maakt geen gebruik meer van HLAPI, waardoor er geen surface rendering meer mogelijk is. Ook ondersteuning voor de Novint Falcon is in ontwikkeling.




**Figuur 32: Het OpenHaptics laagmodel met H3D [33]**

## *Chai 3D*


Ontwikkelaar:

Open Source

Website:

http://www.chai3d.org/

Huidige versie:

CHAI 3D v1.61

Scenegraph voor haptic rendering:

Ja

Scenegraph voor grafische rendering:

Ja

Geschikte hardware:

SensAble PHANToM, Force dimension Omega en Delta, MPB Freedom 6.
Momenteel in ontwikkeling voor de FCS Haptic Master en de “thinking about haptics”
Ondersteuning voor I/O boards.

Compatibele platformen:

Win32, Linux, Mac OS X, QNX, RTAI

Beschrijving:

De CHAI 3D libraries werden ontwikkeld als reactie op het feit dat er teveel closed source haptic API’s bestonden. Er waren wel tal van API’s voor haptisch renderen beschikbaar, maar geen enkele ervan was volledig open. Het gevolg hiervan is de verplichting om met een bepaald haptisch apparaat te werken. Om deze problemen uit de weg te ruimen werd er een echte volledige open source API ontwikkeld: CHAI 3D. De API is te distribueren onder de GNU General Public v2 licentie.
De opzet van CHAI 3D is zo gemaakt dat er eigenlijk aan twee soorten van ontwikkeling gedaan kan worden.
Enerzijds biedt CHAI 3D de mogelijkheid te werken op een manier zoals we die kennen van SensAble’s HDAPI, namelijk dat er kan worden gewerkt op low-level niveau en dat er rechtstreeks gewerkt kan worden met de devicedrivers. Maar in tegenstelling tot HDAPI kan er dus bij CHAI 3D zelf gekozen worden voor welke haptische apparaten te gebruiken en indien gewenst dus een eigen custom apparaat.

Anderzijds biedt CHAI 3D ook de mogelijkheid om in C++ te programmeren door middel van scenegraphs. Net als bij H3D is ook hier scripting voorzien. Zo kan er bij CHAI 3D een XML file ingelezen worden die de scene bevat en de toolkit zal deze dan automatisch renderen door middel van OpenGL. CHAI 3D zal dus net als bij H3D de gebruiker in staat stellen snel haptische 3D-scenes te ontwikkelen. Momenteel ondersteunt CHAI 3D alle bekende 3DOF haptische apparaten zoals de PHANToM en de Delta apparaten, maar indien nodig kan de gebruiker zelf een nieuwe driver voor een ander apparaat toevoegen.
Eveneens kunnen er in CHAI - in tegenstelling tot andere API's - zelf controleschema's gemaakt worden. De meeste API's ondersteunen één enkel Haptisch Interactie Punt (HIP), maar met CHAI kan de gebruiker kiezen voor bijvoorbeeld meerdere HIP's, een lijncontactparadigma of een nieuw zelfgeschreven paradigma. CHAI zelf is geschreven in C++.

## *Haptik Library*

Ontwikkelaar:

Open Source

Website:

http://www.haptiklibrary.org/

Huidige versie:

Haptik Library 1.1RC1

Scenegraph voor haptic rendering:

Neen

Scenegraph voor grafische rendering:

Neen

Geschikte hardware:

Alle haptische apparaten van SensAble Technologies Incorporated, dit van de PHANToM Omni tot en met de premium reeks.
Omega – ForceDimension, Delta – ForceDimension, Freedom - MPB Technologies, Cubic - MPB Technologies, Virtuose6D35-45 – Haption, Virtuose3D15-25 – Haption
Er staan ook nog andere apparaten zoals de Falcon – Novint en de Haptic Master - FCS Robotics op de planning.
Mogelijkheid zelf een plugin toe te voegen voor zijn eigen apparaat of aan de Haptik community vragen of zij het willen doen.

Compatibele platformen:

Win32, GNU/Linux versie in beta-fase.

Beschrijving:

Haptik is een low-level API die als enige doel heeft een abstractielaag te vormen. De API tracht alle haptische drivers te overkoepelen om zo tot een uniforme interface te komen voor al de verschillende hardware. Met andere woorden, een applicatie die gebruik maakt van de Haptik Library, zal onafhankelijk van het type haptisch apparaat blijven werken, zolang er maar ooit een driver voor dat hardwareapparaat werd voorzien in de Haptik Library. De Haptik Library voorziet niets voor grafische rendering, noch enige vorm van scenegraphs. De hardware kan via callbacks worden aangesproken, maar alle algoritmes voor de krachten moeten door de gebruiker zelf worden gemaakt. Het gebruik van Haptik zorgt overigens niet voor een impact op de performantie vergeleken met het gebruik van de

proprietary driver. Haptik is geschreven in C++ en kan in iedere bestaande applicatie worden geïntegreerd. Haptik beschikt over een JAVA JNI interface, een Matlab Mex interface, een Simulink interface en Python bindings. De Haptik Library is vrij te distribueren onder de GNU GPL v2 licentie.

## *OpenSceneGraph Haptic Library (osgHaptics)*


Ontwikkelaar:

Open Source (gestart door VRlab, een afdeling van Umea University)

Website:

http://www.vrlab.umu.se/research/osgHaptics/

Huidige versie:

osgHaptics v0.3

Scenegraph voor haptic rendering:

Ja

Scenegraph voor grafische rendering:

Ja (via de OpenSceneGraph rendering library)

Geschikte hardware:

osgHaptics ondersteunt haptics door middel van de OpenHaptics Toolkit. Zodoende geeft deze ondersteuning voor alle SensAble PHANToM apparaten, dit van de goedkopere PHANToM Omni tot de grotere PHANToM premium apparaten.

Compatibele platformen:

Win32

Beschrijving:

osgHaptics is een initiatief van VRlab, een afdeling van de Umea Universiteit te Zweden. osgHaptics is geschreven in C++ en gebruikt de reeds bestaande OpenSceneGraph rendering library om grafisch te renderen. osgHaptics zorgt voor een link tussen deze grafische library en de OpenHaptics Toolkit. osgHaptics is te distribueren onder de GNU Lesser General Public License v2.1.

### *HAL*

Ontwikkelaar:

The Expertise centre for Digital Media (EDM),
Universiteit Hasselt
Open Source

Website:

http://www2.edm.uhasselt.be/software/hal/

Huidige versie:

HAL v0.1.1

Scenegraph voor haptic rendering:

Ja

Scenegraph voor grafische rendering:

Neen

Geschikte hardware:

SensAble PHANToM device

Compatibele platformen:

Win32

Beschrijving:

HAL is een academische library geschreven in C/C++ en is volledig open source. Net als de Ghost SDK maakt HAL gebruik van een interne scenegraph voor het renderen van zijn haptische scene. Voor het grafisch renderen moet er zelf voor een procedure worden gezorgd.
HAL voorziet momenteel het gebruik van de SensAble PHANToM (via de GHOST I/O wrapper), maar kan ook gebruik maken van een pseudoapparaat indien er geen PHANToM ter beschikking is.
HAL is gedistribueerd onder de LGPL licentie en kan dus - indien de gebruiker dit wenst - zelf uitgebreid worden.
De API bestaat uit 6 libraries:

- Core

  De kerncomponenten van HAL, in deze library zitten basisobjecten die door alle andere libraries gebruikt worden.

- Scenegraphs

  In deze libary zitten de verschillende scenegraphs. Momenteel is dit wel wat beperkt. Er is enkel één scenegraph aanwezig die gebruikt wordt voor collisiedetectie.

- Objects

  In deze library zitten de haptische vormen. Een voorbeeld hiervan zijn kubussen, spheres, en polymeshes.

- Forces

  Hier zijn er momenteel 2 soorten krachtvelden te vinden, een drag forcefield en een spherical forcefield.

- Phantom

  Deze library bestaat uit de PHANToM device driver, gebaseerd op de GHOST I/O wrapper.

- Pseudo

  Deze library kan gebruikt worden om een dummydevice voor te stellen. Dit kan gebruikt worden om de applicatie te testen.

### *Reachin API*


Ontwikkelaar:

Reachin Technologies

Website:

http://www.reachin.se/

Huidige versie:

Reachin API v4.2

Scenegraph voor haptic rendering:

Ja

Scenegraph voor grafische rendering:

Ja

Geschikte hardware:

Alle apparaten van SensAble, Novint, Force Dimension en 3D Connexion.

Compatibele platformen:

Win32

Beschrijving:

De Reachin API is een van de meest complete pakketten die momenteel te vinden zijn, maar ze zijn niet open source.

Reachin biedt twee versies van zijn product aan. Allereerst is er de standaardversie, die de gebruiker toelaat te ontwikkelen in Python en de Virtual Reality Modeling Language (VRML). VRML is de voorloper van X3D, dit laatste wordt in H3D gebruikt. Voorts is er nog de professionele editie die de gebruiker ook toelaat in C++ te werken en dus zodanig toelaat haptics te integreren in een reeds bestaande applicatie.

Er kan gekozen worden voor een VRML bestand om de scene zodanig te renderen of de haptics kunnen toegevoegd worden in OpenGL code. Reachin biedt ook nog speciale libraries aan voor krachtvelden en voor netwerkondersteuning. Ook 3D geluid wordt door de API ondersteund. We dienen wel op te merken dat het sinds versie 4.0 verplicht is met Microsoft Visual Studio te werken.

Voor zover we kunnen zien biedt Reachin een pakket aan met voor elk wat wils. Zo kan het pakket gebruikt worden zoals H3D gebruikt wordt , om snel prototypes te maken, of om meer low-level te gaan en haptics te integreren in een bestaande C++ applicatie.

## *Andere toolkits*

We merken op dat er nog tal van andere toolkits op de markt zijn, maar deze zijn niet te verkrijgen zonder ze eerst te kopen. Enkele voorbeelden hiervan zijn:

Immersion Studio van Immersion Corp. Immersion is vrijwel op alle gebieden van haptics actief, ze maken zowel software als hardware voor gebruik in medische toepassingen, telerobotics, videogames, digitalisatie,… Al hun software is echter volledig commercieel en enkel geschikt voor hun eigen hardware. Voor meer informatie zie http://www.immersion.com

Novint Technologies had vroeger ook een haptics API: e-touch. Deze is echter niet meer beschikbaar. Novint Technologies is wel nog steeds bezig op het vlak van haptics en heeft recent een spelcontroller uitgebracht die samen met Force Dimension werd ontwikkeld. De Novint Falcon is een goedkope haptische controller, speciaal ontwikkeld voor spellen en Novint zal ook zelf de plug-ins voor de spellen voorzien. Voor meer informatie zie http://www.novint.com/

MPB Technologies is net zoals Immersion op verschillende gebieden actief. Hun software is helaas ook volledig closed source, en enkel geschikt voor hun eigen hardware. Voor meer informatie zie http://www.mpb-technologies.ca/

Force Dimension is een ander gespecialiseerd bedrijf dat op vele vlakken bezig is met haptics, ze maken onder andere hun eigen hardware en ze zijn actief in de medische sector, telerobotics en de nanomanipulatie. Voor meer informatie zie http://www.forcedimension.com

## *Conclusie*

Er is op dit moment veel software beschikbaar voor haptische rendering, maar er zijn nog niet erg veel open source toolkits. Ook is er te zien dat SensAble Technologies Incorporated blijkbaar voor een volledige stroming van producten heeft gezorgd; ze zijn vrijwel de grootste speler op de markt en er zijn zelfs open source toolkits die berusten op hun API's.

## Hoofdstuk 6

# Bestaande hardware

Aangezien we reeds veel hebben gesproken over API's en de apparaten waarmee ze samenwerken, geven we hier een kort overzicht van de meest gebruikte haptische apparaten.

### *SensAble PHANToM*

SensAble Technologies maakt vrijwel het meest bekende haptisch apparaat dat er bestaat. De PHANToM komt dan ook in verschillende modellen [Figuur 33]. De PHANToM-apparaten bieden afhankelijk van het model drie of zes vrijheidsgraden.

**Figuur 33: Enkele modellen uit de PHANToM lijn. Links: De PHANToM Omni, Centraal: De PHANToM Desktop, Rechts: De PHANToM PREMIUM [34]**

### *Force Dimension*

Force dimension maakt 3 modellen [Figuur 34], één daarvan is het Omegamodel dat standaard voorziet in drie vrijheidsgraden, de andere twee modellen zijn uit de Deltareeks en zijn respectievelijk voor drie vrijheidsgraden en voor zes vrijheidsgraden.

**Figuur 34: Links De Omega.3, Centraal: De Delta 3,  Rechts: De Delta 6 [35]**

## *Novint*

Novint brengt binnenkort een zeer goedkoop model uit dat in samenwerking met Force Dimension werd ontwikkeld. De Novint Falcon [Figuur 35] zal drie vrijheidsgraden hebben en aangesloten worden via USB 2.0.

**Figuur 35: De Novint Falcon [36]**

## *Conclusie*

We zien dat er verschillende apparaten in tal van vormen beschikbaar zijn, maar in hoofdstuk 5 is er te zien dat er maar één reeks apparaten is die door alle beschikbare API's ondersteund wordt: de SensAble PHANToM lijn. De grote reden hiervoor is ook dat SensAble als eerste op de markt kwam met een dergelijk commercieel apparaat. Dit model geeft dan ook het meeste vrijheid qua gebruikte software.

**Deel II**

# Implementatie

## Hoofdstuk 7

# Demo's haptische API's

We hebben gezien dat er verschillende API's bestaan voor haptics. We zullen nu een aantal van deze API's selecteren en van alle geselecteerde API zullen we een kleine implementatie maken om een idee te kunnen krijgen van hun werking. De selectie bestaat uit volgende toolkits:

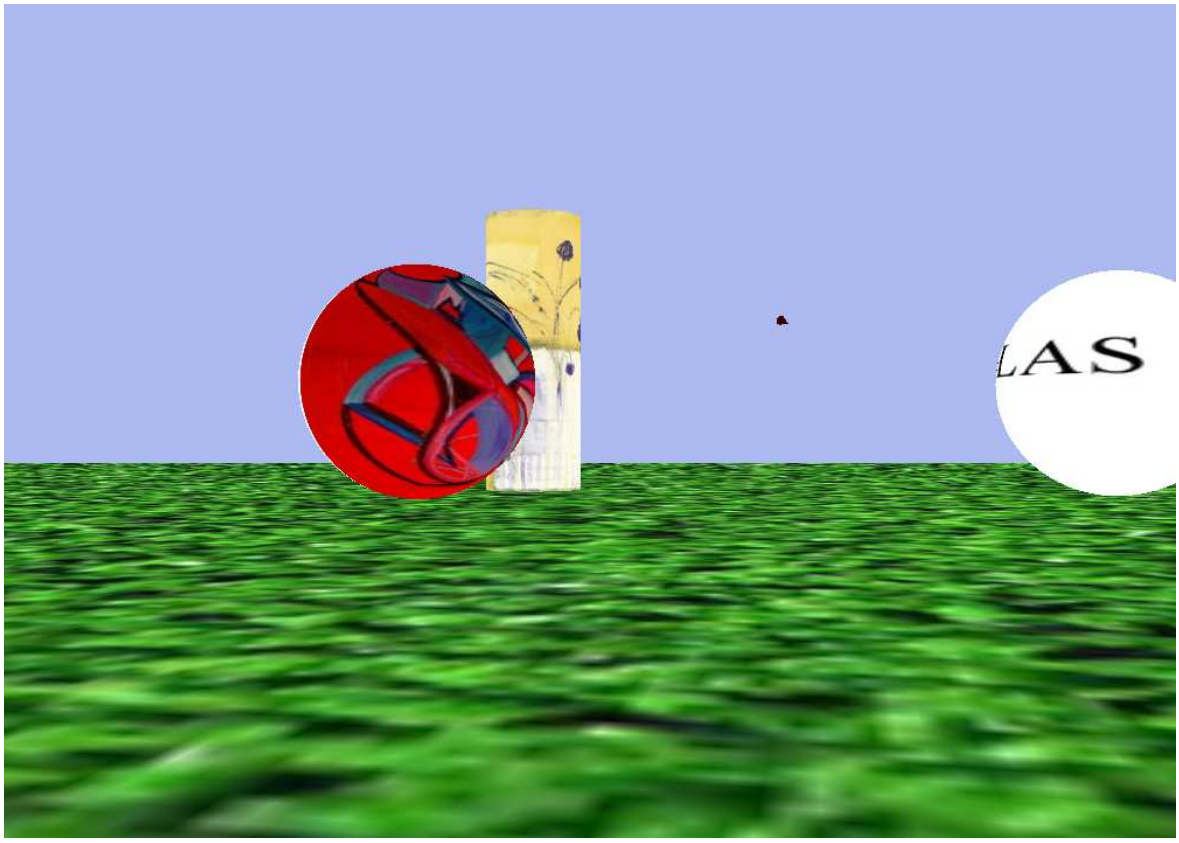
- SensAble OpenHaptics Academic Edition v2.0
- CHAI 3D v1.6.1
- SenseGraphics H3D API v1.5
- HAL v0.1.1

De keuze om deze API's te implementeren werd genomen, gebaseerd op hun voorziene documentatie en natuurlijk ook vanwege het feit dat deze API's allemaal gratis beschikbaar zijn.
We zullen ook een demo maken met HAL, zodat we nadien een beter beeld krijgen van wat we aan deze API kunnen gaan uitbreiden.

### *SensAble OpenHaptics Academic Edition v2.0*

OpenHaptics is in twee verschillende versies beschikbaar. De commerciële editie en de academische versie. De academische versie biedt qua mogelijkheden exact dezelfde features als de commerciële editie, enkel dat hierbij geen ondersteuning wordt gegeven. De academische versie is voor iedereen, gelinkt aan een onderwijsinstelling gratis te downloaden na het doorlopen van een registratieprocedure. Na het downloaden van de toolkit valt al dadelijk op dat er veel documentatie is voorzien. Zo is er onder andere een map te vinden met daarin uitleg over de te gebruiken hardware en ook een readMe-bestand met daarin uitleg omtrent de installatie van de toolkit. Een ander groot pluspunt bij de OpenHaptics toolkit is dat er ook een overvloed aan documentatie online te vinden is. Zo is er een 114 pagina's tellende programmer's guide beschikbaar met daarin uitleg van alle functies van OpenHaptics, zowel tekstueel als met korte voorbeelden in C++. Voorts is ook een API reference van 130 pagina's te vinden met daarin een overzicht van alle klasses en functies. Voor wie dit alles nog niet genoeg is, biedt SensAble op hun site een forum aan, alsook een reeks

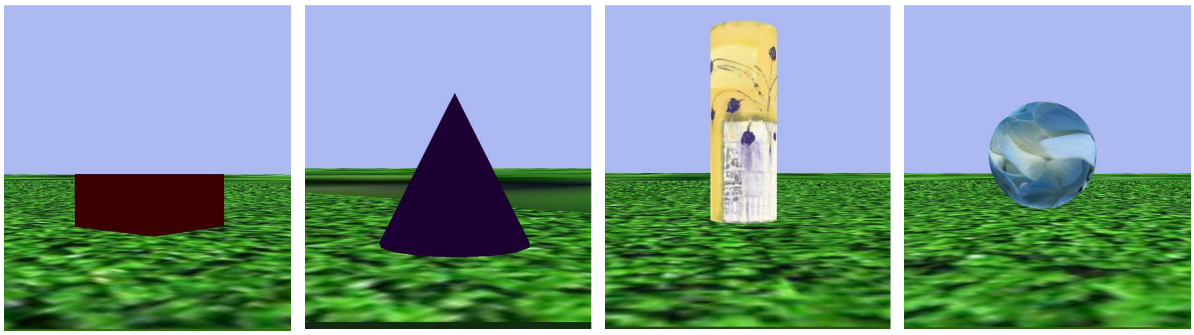
implementatievoorbeelden en een reeks 3rd-party demo's. Dit alles is best opvallend te noemen omdat deze documentatie voor eender welke versie voorzien is.
De OpenHaptics toolkit is te gebruiken door middel van C++. De API is standaard gecompileerd voor MSVC 6, maar de projectbestanden voor latere versies zijn eveneens beschikbaar en zijn zonder noemenswaardige problemen te compileren. Voor onze demo werd gebruik gemaakt van MSVC 7. Zoals reeds vermeld in het overzicht van de API's, beschikt OpenHaptics niet standaard over een grafische renderer. Wel is OpenHaptics zodanig gemodelleerd dat de syntax zeer veel gelijkenissen vertoont met die van OpenGL. Het is dus ook de bedoeling dat OpenHaptics samen met OpenGL gebruikt wordt. Het grafisch gedeelte van onze demo is dan ook in OpenGL gedefiniëerd, als window toolkit hebben we gekozen gebruik te maken van Trolltech QT4[2]. Zowel OpenGL en QT zijn vrij beschikbaar. De door ons opgestelde scene [Figuur 36] heeft als doel verschillende features van de API te testen.




**Figuur 36: Een kleine scene gemaakt met OpenGL en OpenHaptics**

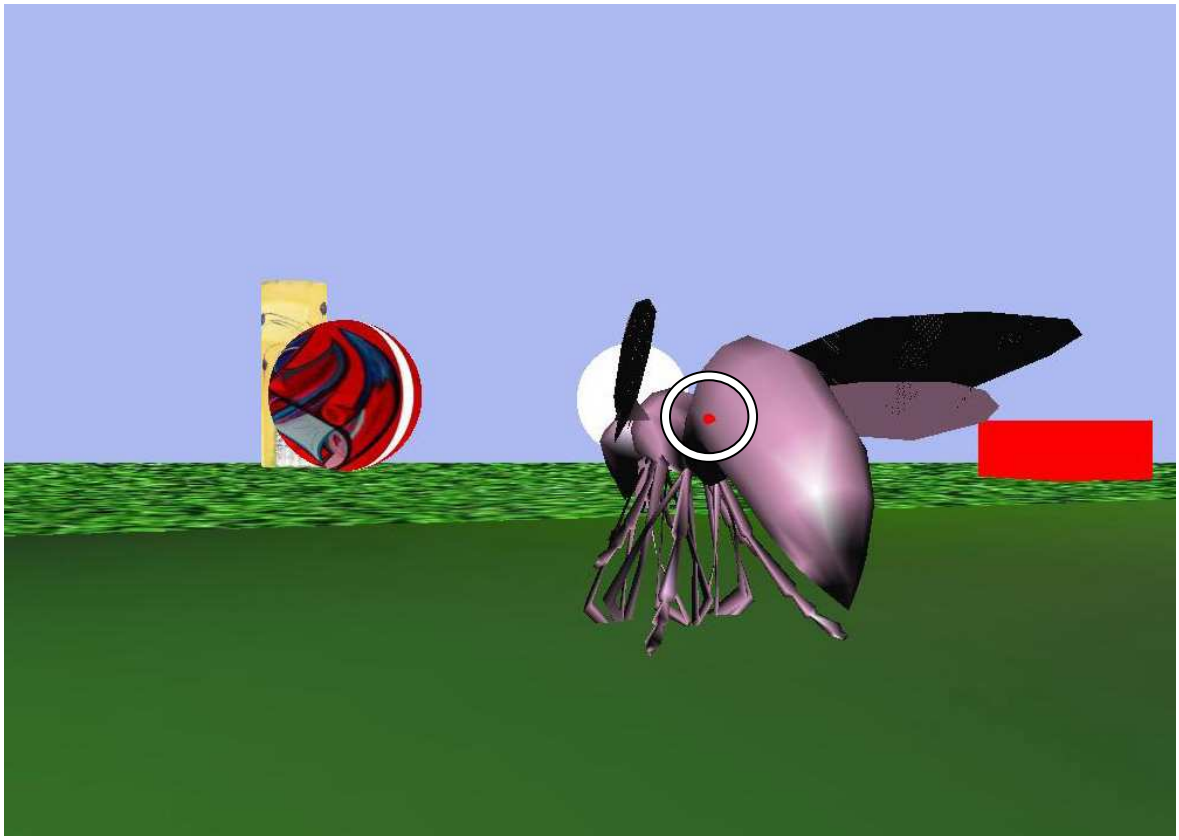
Als eerste hebben we gekeken hoe we een statische scene zowel grafisch als haptisch konden opbouwen. Dit werd gedaan door middel van enkele OpenGL primitieven. In onze demo renderen we een kubus, een cilinder, een kegel en een sphere, allen voorzien van een textuur [Figuur 37].

[2] www.trolltech.com



**Figuur 37: De gerenderde objecten, van links naar rechts: Een kubus, een kegel, een cylinder en een sphere**

Het renderen van een haptisch object gebeurt analoog aan het renderen van een OpenGL object. Net zoals we in OpenGL gebruik maken van glBegin() en glEnd(), gebruiken we in OpenHaptics hlBeginShape() en hlEndShape(). Dit kan gewoon in de OpenGL code worden toegevoegd en is niet enkel mogelijk voor primitieven, maar ook voor displaylists en trianglemeshes [Figuur 38].



**Figuur 38: Een trianglemesh, zowel grafisch als haptisch gerenderd. De virtuele manipulator werd met een cirkel geaccentueerd**

Tenslotte moet er een mapping gebeuren van het fysieke haptische apparaat naar een virtuele manipulator in onze virtuele wereld. Zodoende is er een visuele feedback van de acties van de gebruiker. Het renderen van de manipulator is te zien op [Figuur 38] en de code hiervoor kan de lezer terugvinden in [bijlage 1]. Na het implementeren van deze code kan de scene getest worden. De objecten werden stabiel en accuraat waargenomen met ons haptisch device. Er werden geen storende vibraties of andere anomalieën waargenomen. Vervolgens werd er

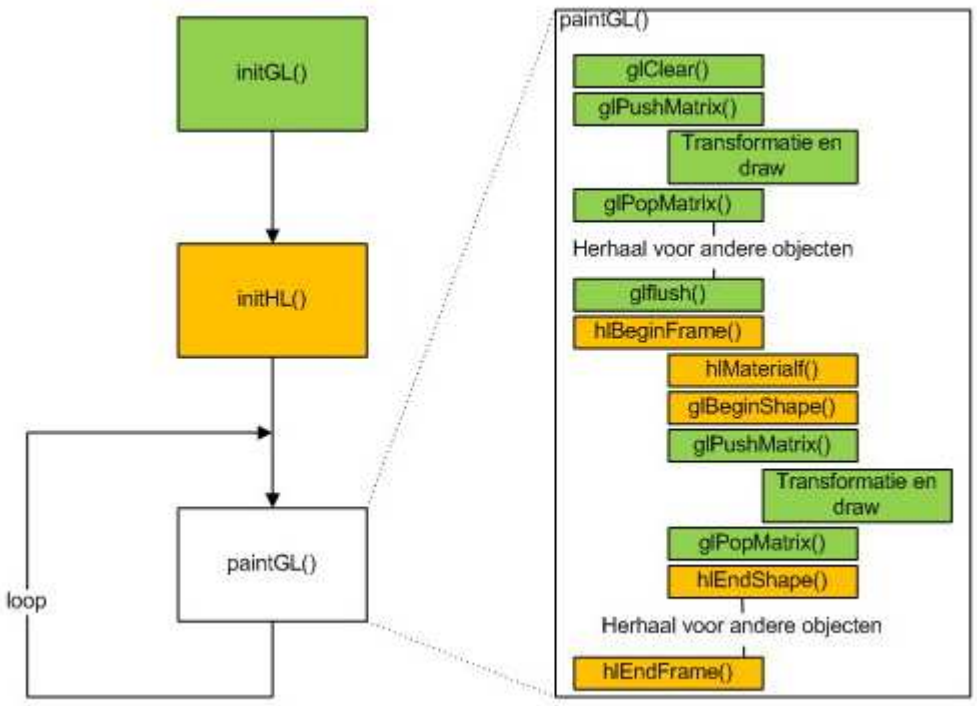
gekeken naar hoe we materiaaleigenschappen konden gebruiken. Verschillende eigenschappen zijn hier mogelijk:

- Stiffness: Bepaalt hoe hard een object aanvoelt.
- Damping: Bepaalt hoe we een botsing met een object gewaar worden.
- Friction: Bepaalt hoe de oppervlakte van een object aanvoelt.
- Popthrough: Bepaalt of we door een object kunnen poppen.

Aan de hand van deze eigenschappen werden enkele objecten opgesteld die identiek waren qua vorm, maar verschillend waren qua materiaal [Figuur 39]. Ook het testen van deze eigenschappen verliep vlot. Door te experimenteren met de materiaaleigenschappen werden we snel gewaar van de effecten van onze instellingen.


**Figuur 39: Enkele spheres voorzien van bepaalde haptische materiaaleigenschappen**

Tenslotte hebben we de scene voorzien van een navigatieklasse zodat we door onze scene konden navigeren. De visuele navigatie in de scene is geheel in OpenGL te implementeren, maar voor het mappen van de verplaatsing met de haptische scene dienen we de procedure in [Bijlage 2] aan te roepen. De algemene manier van werken kunnen we rudimentair voorstellen met het voorbeeldschema op [Figuur 40].




**Figuur 40: Een voorbeeldschema van de integratie van SensAble OpenHaptics in een bestaande OpenGL applicatie**

Het schema op [Figuur 40] toont hoe we SensAble OpenHaptics kunnen integreren in een reeds bestaande OpenGL-applicatie. De OpenGL-code tonen we in het groen, terwijl we de code die nodig is voor OpenHaptics markeren in het oranje. Aan de hand van dit schema zien we duidelijk hoe nauw OpenHaptics verbonden is met OpenGL. Niet alleen gebruikt OpenHaptics een soortgelijke syntax als OpenGL, maar OpenHaptics zal ook OpenGL-code gebruiken als input voor translaties en het renderen van de objecten in de haptische scene. Ook zien we duidelijk hoe de rendering zowel moet gebeuren voor de graphics als voor de haptics. Tenslotte willen we hier nog opmerken dat het schema duidelijk de state-machine-aard van OpenGL en OpenHaptics laat zien. Dit komt er op neer dat we de scene voor elke frame zowel grafisch als haptisch volledig dienen te herdefiniëren.

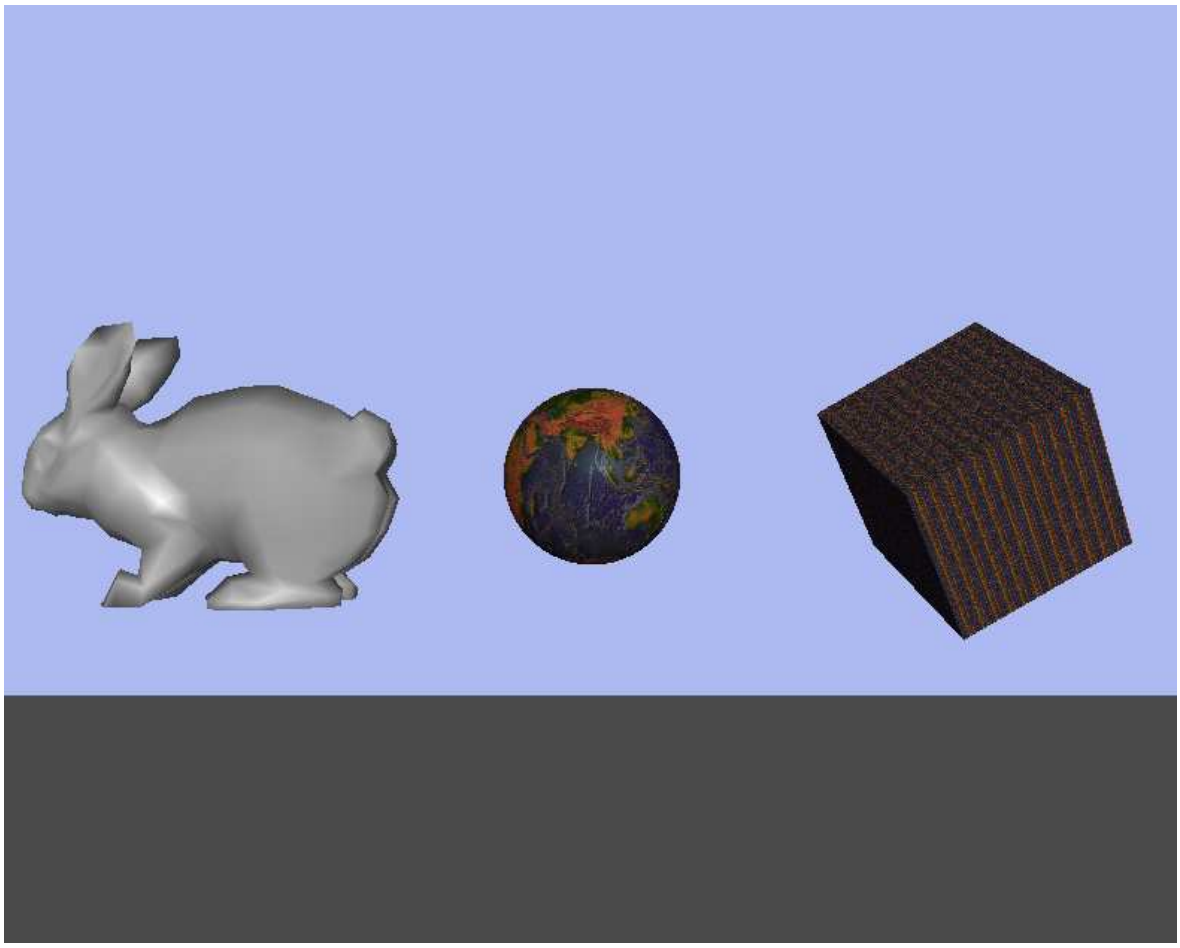
## Conclusie

Er kan besloten worden dat de toolkit van SensAble OpenHaptics zeer gemakkelijk werkt. Een groot voordeel om snel met deze toolkit te kunnen werken is de overvloed aan documentatie en de voorbeelden die er ter beschikking gesteld zijn. Ook de gelijkenissen van de syntax met deze van OpenGL zijn een groot voordeel. Ontwikkelaars met kennis van OpenGL kunnen zo namelijk zeer snel overweg met OpenHaptics. De hier voorgestelde demo was maar een klein voorbeeld van hoe OpenHaptics toegepast kan worden, maar het werd ons wel

snel duidelijk dat OpenHaptics zeer vlot te integreren valt met een reeds bestaande OpenGL-applicatie. Hoewel OpenHaptics niet over een grafische renderer beschikt, moet er toch niet erg veel code worden bijgevoegd om de grafische scene te mappen met deze van de haptics; de scene moet weliswaar dubbel gedefinieerd worden. Deze demo was vlot te maken en er werden geen noemenswaardige problemen gevonden.

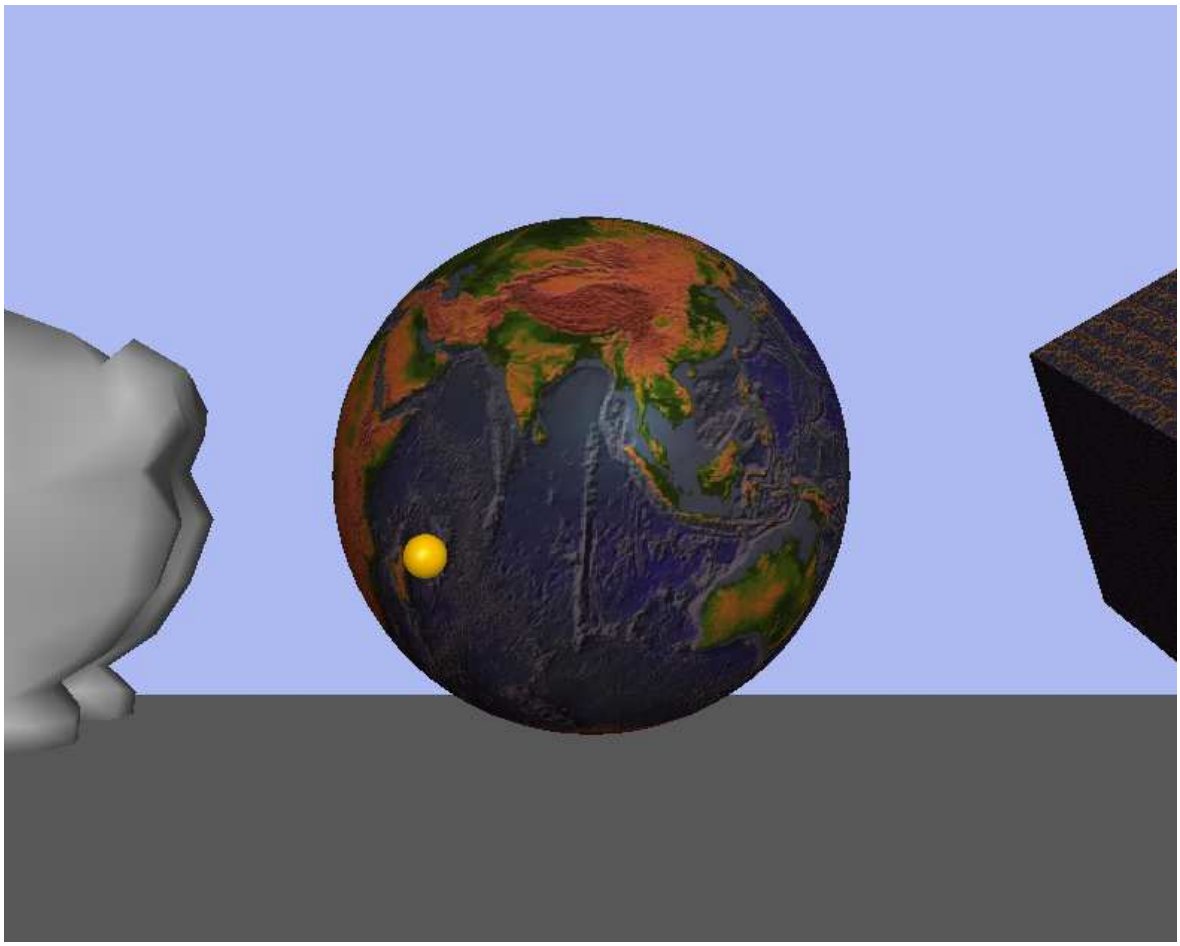
### *CHAI 3D v1.6.1*

CHAI staat voor “Computer Haptics & Active Interfaces”. De toolkit tracht een totaaloplossing te bieden door zowel te voorzien in het haptisch renderen als in het grafisch renderen van een scene. Hiervoor wordt er met een scenegraph gewerkt. Om iets met CHAI 3D te implementeren moet er in C++ gewerkt worden.
Om de demo te ontwikkelen werd er een scene gemaakt gelijkend op deze van SensAble OpenHaptics. Net zoals OpenHaptics is CHAI 3D vrij te verkrijgen op het internet. Wel is hier een groot verschil, CHAI 3D is namelijk volledig open source en biedt ondersteuning aan verschillende types invoerapparaten. Ook voor CHAI 3D is er veel informatie beschikbaar. Zo is er op de website van CHAI 3D een forum en ook is er een API-reference beschikbaar. Na het downloaden van de meest recente versie vinden we in deze bestanden eveneens talrijke voorbeeldapplicaties.
Voor onze demo maken we gebruik van MSVC 7, maar CHAI 3D is ook reeds compatibel met MSVC 8.
De demo-applicatie wordt in C++ gemaakt, als window toolkit gebruiken we GLUT. Om dezelfde scene als in OpenHaptics te implementeren dienen we hier anders te werk te gaan. CHAI 3D werkt voor zijn rendering volledig met behulp van een scenegraph. Om een object te renderen moet dit object dus gedefinieerd worden en als node aan de scenegraph worden toegevoegd. CHAI 3D zal er dan voor zorgen dat dit object zowel grafisch als haptisch gerenderd wordt. CHAI 3D biedt niet zoveel primitieven als OpenHaptics en maakt meer gebruik van een algemene trianglemesh. De gemaakte testscene is zichtbaar op [Figuur 41].



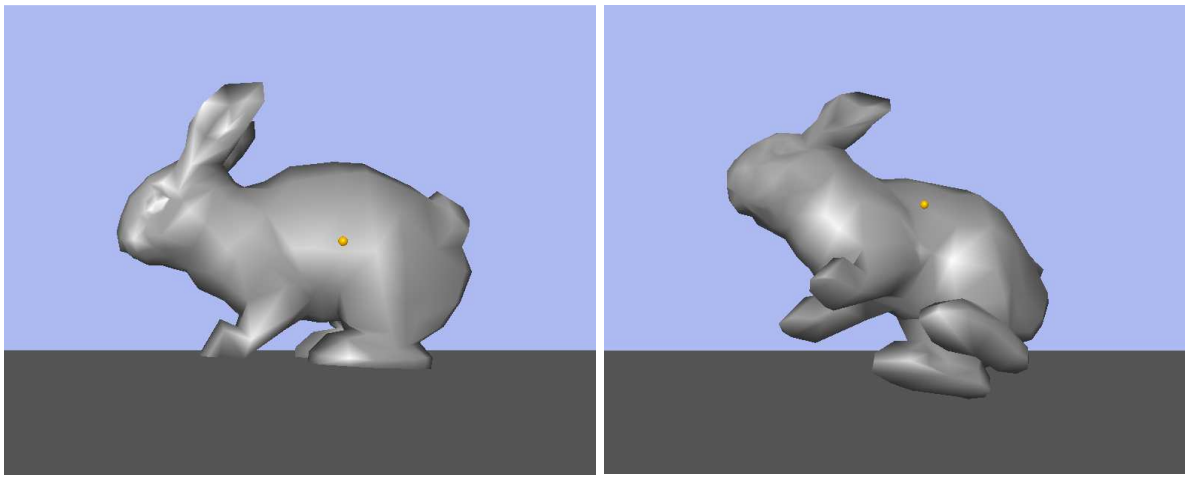
**Figuur 41: Een testscene gemaakt in CHAI 3D**

De objecten in deze scene [Figuur 41] bestaan van links naar rechts uit: Een ingelezen 3D Studio Max bestand, een voorgedefinieerde sphere en een standaard trianglemesh. De sphere is de enige voorgedefinieerde primitieve in CHAI 3D, de kubus is namelijk gedefinieerd als een trianglemesh die nadien voorzien wordt van een vertexlijst voor het vormen van een kubus. Het konijn in de scene is een object, gemaakt in 3D Studio Max. CHAI 3D biedt functionaliteiten aan voor het inlezen van dit type bestanden (*.3ds, *.obj).
Ook het toepassen van grafische texturen dient te gebeuren via CHAI 3D zelf. De manier om dit te doen is analoog aan deze van OpenGL. Primitieven hebben zelf hun textuurcoördinaten, terwijl deze voor trianglemeshes gegenereerd dienen te worden. Zowel de sphere als de kubus (die een trianglemesh is) zijn van een afzonderlijke textuur voorzien in onze scene.
Er werd ook gekeken naar het toepassen van materiaaleigenschappen. Deze zijn beperkter dan deze van SensAble OpenHaptics, enkel stiffness en friction kunnen worden ingesteld.
Tenslotte dient enkel nog de haptische interactor in de virtuele scene te worden gerenderd. Dit wordt gedaan met behulp van de code in [Bijlage 3], de gerenderde cursor is zichtbaar op [Figuur 42].



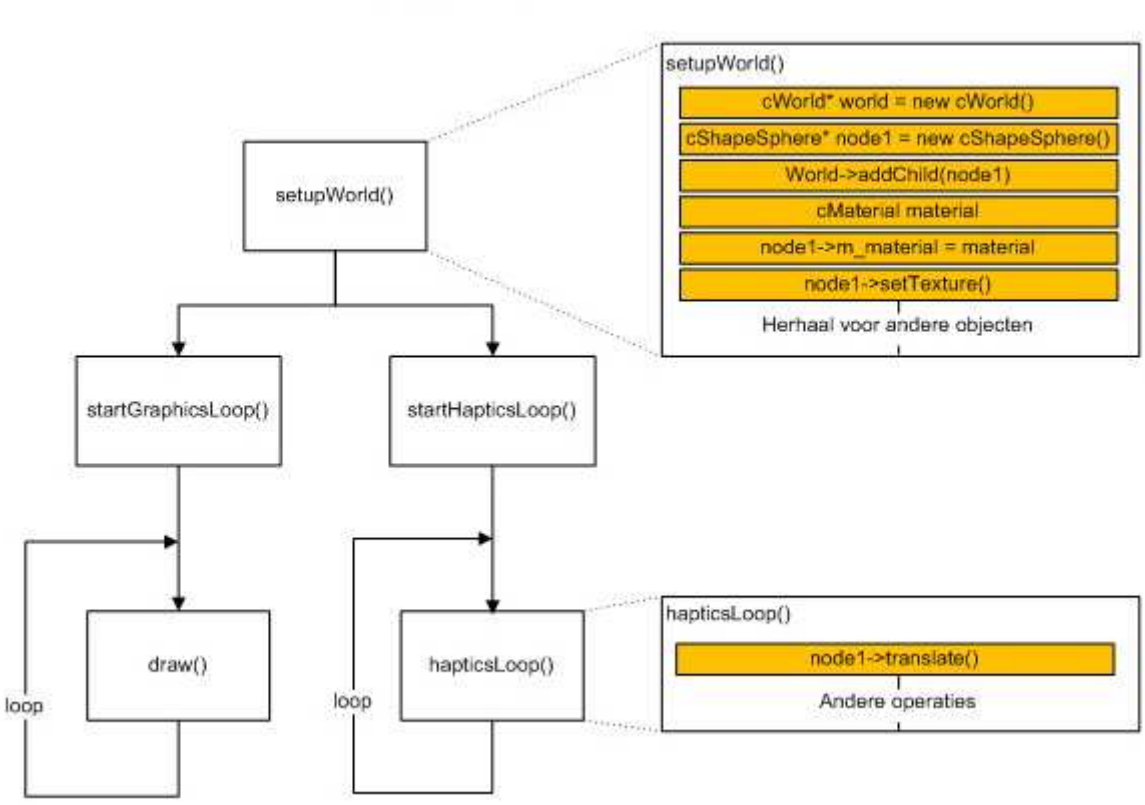
**Figuur 42: De haptische cursor in CHAI 3D**

Navigatie in CHAI 3D is mogelijk door middel van een cameranode in de scenegraph. Deze cameranode is voorzien van methods om de positie en de kijkvector aan te passen.
Ook hebben we in CHAI 3D gekeken naar hoe interacties kunnen worden uitgevoerd op objecten. Zo hebben we in de demo de mogelijkheid om objecten te stoten. Dit was toe te passen door in de haptische loop een lijst op te vragen van de objecten die in contact zijn met de haptische cursor. Deze objecten worden dan geroteerd, afhankelijk van de positie van de cursor. De code hiervoor is terug te vinden in [Bijlage 4]. De resultaten van een dergelijke interactie zijn zichtbaar op [Figuur 43].



**Figuur 43: Links: Een object voor interactie. Rechts: Een object na interactie**

Het volledige proces van een scene, gerenderd met CHAI 3D kunnen we rudimentair voorstellen door [Figuur 44].




**Figuur 44: Een voorbeeldschema van een scene gerenderd met CHAI 3D**

Op [Figuur 44] zien we hoe we een scene eenmalig opstellen en hoe we deze kunnen renderen. Het is duidelijk te zien dat CHAI 3D zowel het grafische als het haptisch gedeelte volledig voor zich neemt. We hebben hier in dit schema een aparte loop voor graphics, omdat de grafische lus veel trager geupdate zal worden in vergelijking met die van de haptics. De transformatie die we in dit schema doen in de hapticsLoop zou bijvoorbeeld een reactie kunnen zijn van een botsing tussen de haptische cursor en een object. We merken hier op dat we duidelijk het verschil

zien met de state-machine-aard die we in de demo met SensAble OpenHaptics zagen. Hier dienen we de scene maar eenmaal te definiëren, waarna CHAI 3D de rendering per frame op zich zal nemen. Enkel de aanpassingen aan objecten dienen aan de scenegraph doorgegeven te worden.

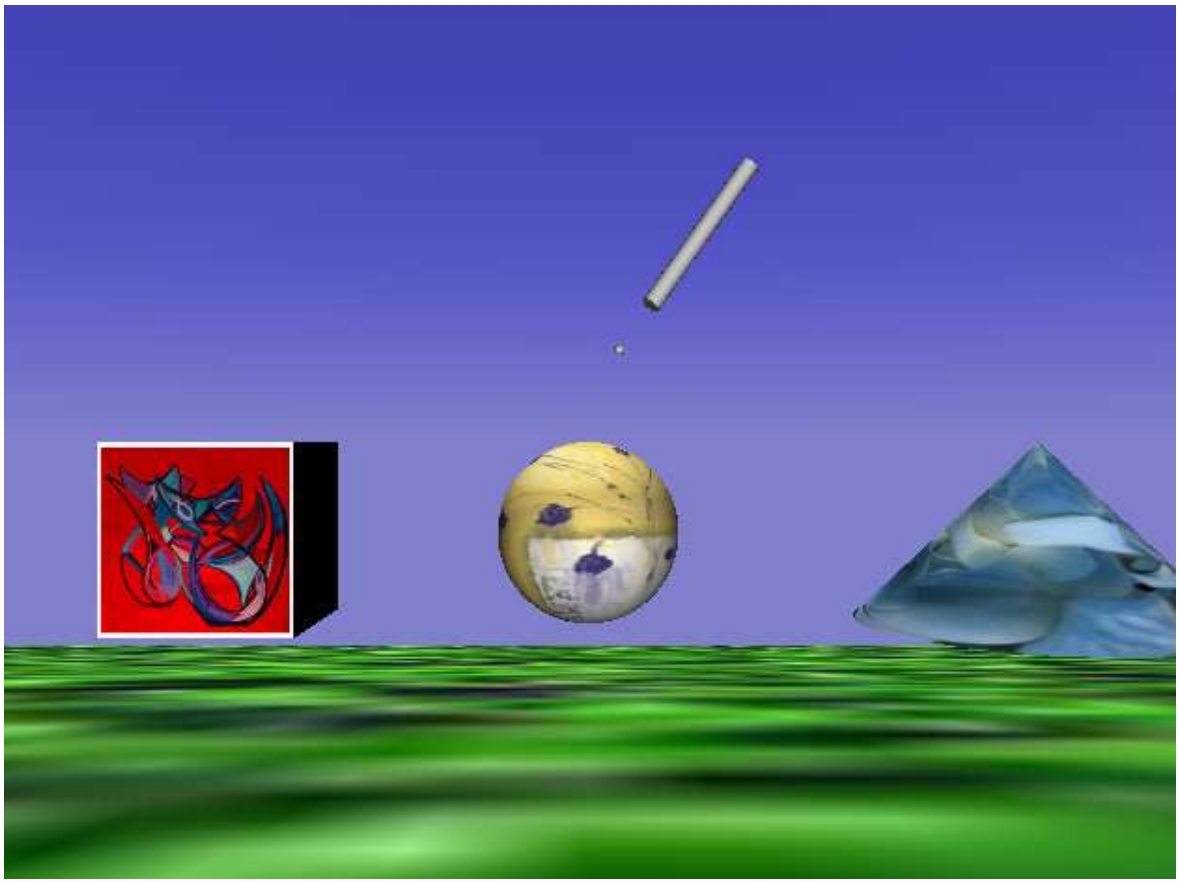
### Conclusie

In tegenstelling tot de OpenHaptics toolkit biedt CHAI 3D een volledig framework waarin we gebruik maken van een scenegraph, die zowel gebruikt wordt voor de graphics als voor de haptics. Een nadeel hiervan is dat CHAI 3D niet in een bestaande OpenGL-applicatie toegepast kan worden. Anderzijds kan er - indien er van nul een applicatie gemaakt dient te worden - sneller een werkende applicatie gemaakt worden, aangezien de grafische en haptische code dezelfde zijn. CHAI 3D is vrij makkelijk aan te leren omdat ook van deze toolkit veel informatie te vinden is. Eén der grote voordelen van CHAI 3D is dat deze API niet gebonden is aan één bepaald type invoerapparaat. Zo biedt CHAI 3D ondersteuning aan tal van haptische apparaten en is deze bovendien volledig open source. Komt er dus een nieuw apparaat op de markt, dan kan CHAI 3D snel worden aangepast zodat het werkt met dit apparaat.

## *SenseGraphics H3D API v1.5*

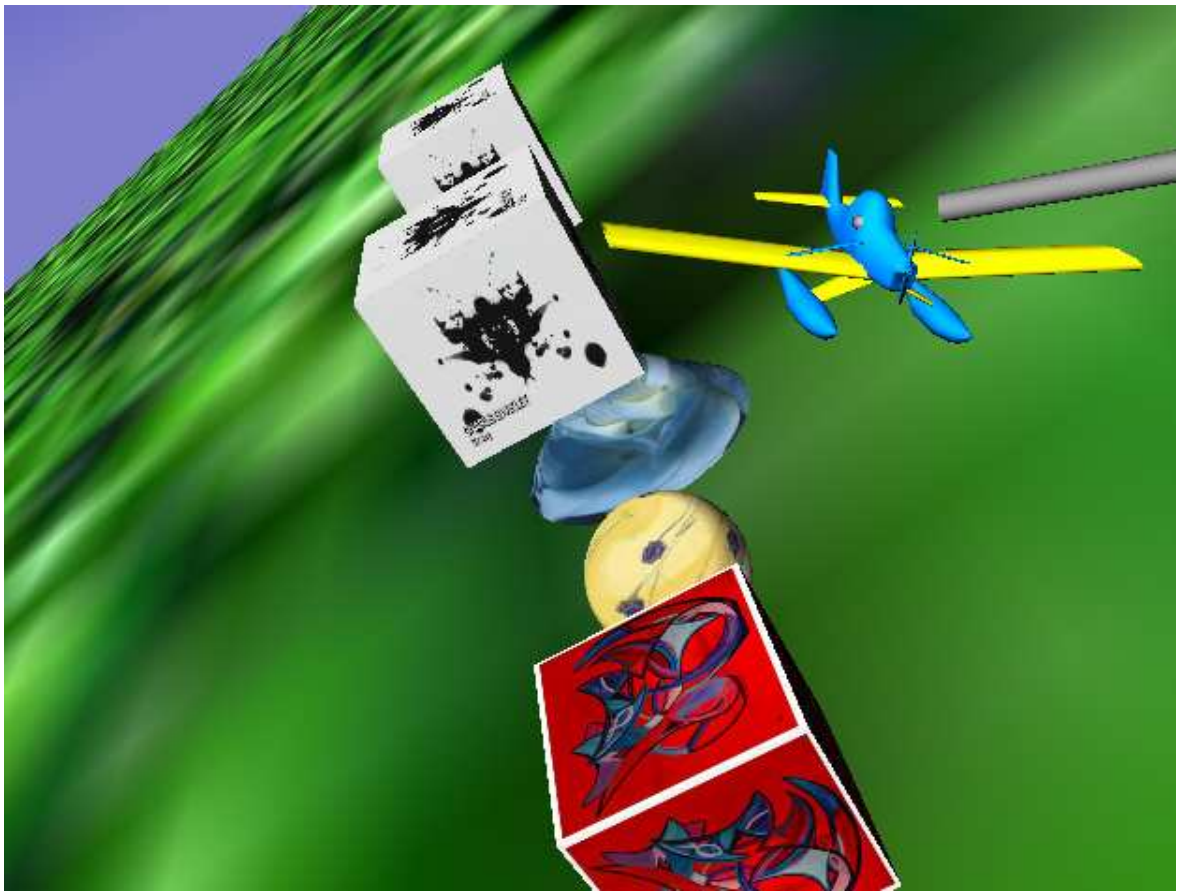
SenseGraphics H3D is een API die net als CHAI 3D een totaaloplossing tracht te bieden. H3D werd in 2004 op de markt gebracht onder een GPL-licentie. Dit komt er in wezen op neer dat H3D volledig open source is en door iedereen gebruikt mag worden. Voor zowel het grafisch renderen als het haptisch renderen wordt er gebruik gemaakt van een scenegraph. Er zijn twee manieren waarop deze graaf aangemaakt kan worden. Een eerste manier is door gebruik te maken van X3D, de opvolger van VRML. X3D laat de gebruiker toe om zijn volledige scene voor te stellen in een xml-compatibel script. Aan dit X3D-script kan extra functionaliteit gegeven worden door dit te koppelen aan een pythonscript. In python kunnen er dan meer geavanceerdere procedures ingesteld worden.
De tweede manier bestaat erin de scenegraph zelf op te bouwen door gebruik te maken van C++. Indien gewenst kan er in C++ nog altijd een X3D-bestand worden ingelezen. Voor het haptisch renderen is H3D volledig afhankelijk van de SensAble OpenHaptics toolkit. Deze toolkit dient dus altijd samen met H3D geïnstalleerd te zijn.
De H3D toolkit is na registratie vrij te downloaden op de H3D website. Net zoals bij de vorige toolkits is ook de H3D website een goede bron van informatie. Zo is er

een online forum, een API reference, en er is ook een starters guide beschikbaar. Na het downloaden van de API zien we eveneens dat ook hier tal van voorbeelden meegeleverd worden.
Voor deze demo werd er geopteerd gebruik te maken van X3D in combinatie met python. We hebben in X3D getracht een gelijksoortige scene op te stellen als in de andere demo's. De opgebouwde scene is te zien op [Figuur 45].



**Figuur 45: Een scene opgesteld in X3D voor H3D**

Het X3D-bestand is in te lezen met een bijgeleverde tool: H3DLoad. Deze tool zal ons bestand gaan parsen en zal onze scene zowel haptisch als grafisch gaan renderen. De tool zorgt ook voor een virtuele haptische interactor die op [Figuur 45] te zien is. Daarnaast voorziet H3DLoad in een basisnavigatie door middel van de muis. Een volgende stap was het instellen van haptische materiaaleigenschappen: ieder object in onze scene werd voorzien van een specifieke eigenschap. De mogelijke eigenschappen waren in te stellen door middel van een stiffnessfactor. Ook andere eigenschappen zoals een magnetisch oppervlak behoren tot de mogelijkheden. Er werd ook een krachtveld ingesteld, dat zowel globaal als lokaal toe te passen is. In onze scene werken we met texturen, die zeer gemakkelijk in te lezen zijn door simpelweg een verwijzing naar het beeldbestand te maken in het X3D-bestand. Een volgende stap bestond eruit te kijken hoe er met trianglemeshes wordt omgegaan. Ook dit is met X3D te doen. Een voorbeeld van een scene met een trianglemesh is te zien op [Figuur 46].



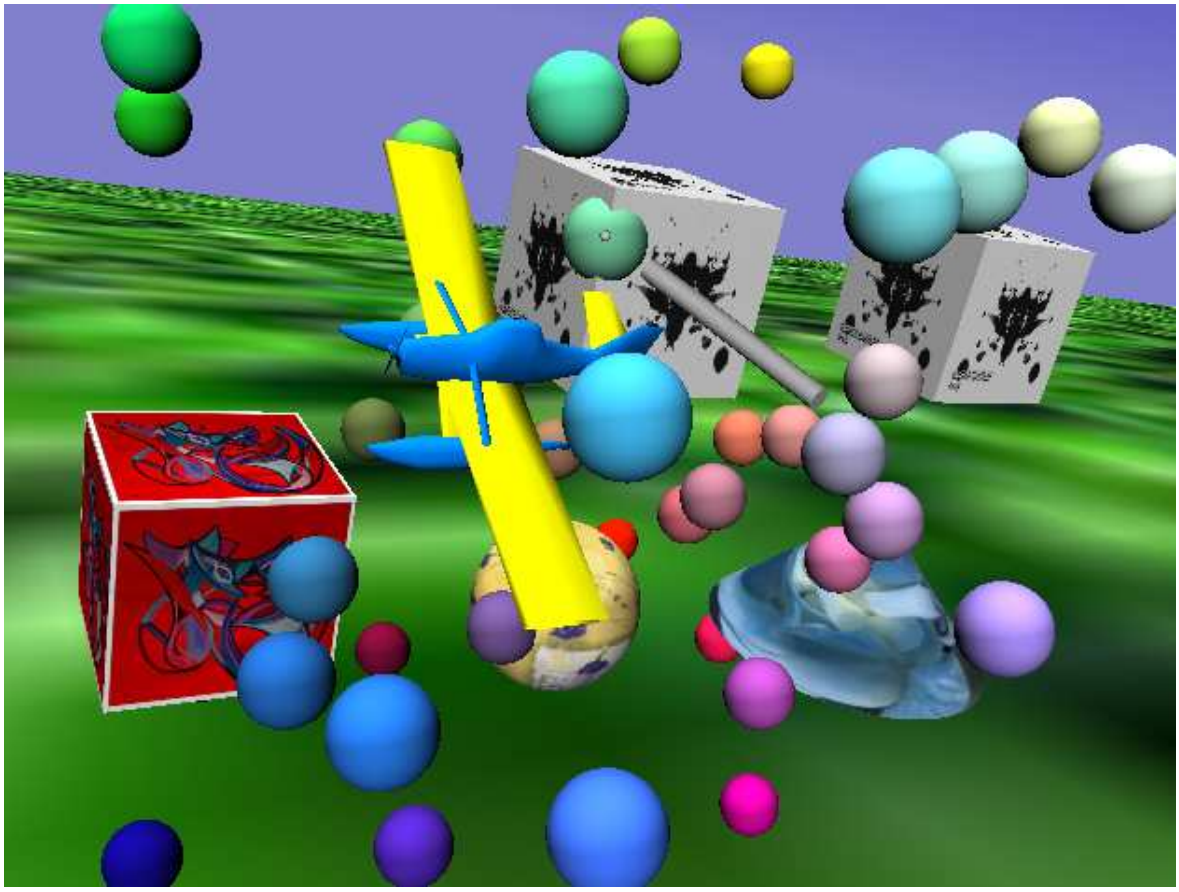
**Figuur 46: De scene waarvan het vliegtuigje ingelezen werd als een trianglemesh**

Een voordeel van het werken met X3D is dat we met verschillende X3D-bestanden kunnen werken, in onze demo maken we gebruik van één enkel X3D-bestand, maar er kunnen ook verwijzingen gemaakt worden naar andere bestanden met daarin bijvoorbeeld een bepaalde trianglemesh. X3D is een populair bestandsformaat en de objecten kunnen worden ontworpen in reeds bestaande en in waarde gevestigde applicaties zoals:

- Blender
- Seamless3d
- X3D-weaver
- Maya 3D
- Flux Studio

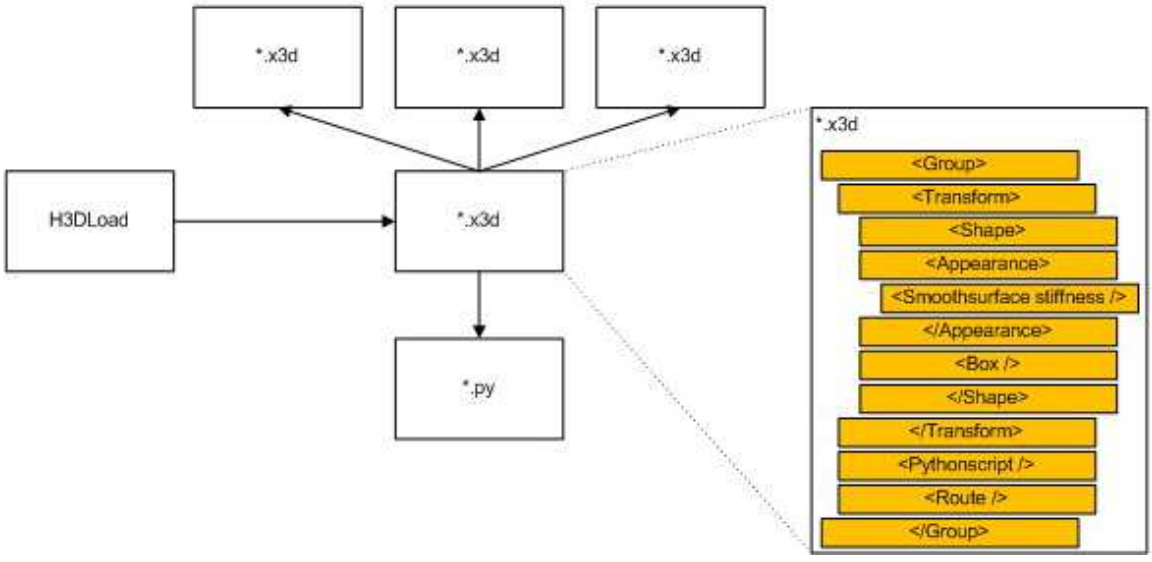
Er zijn zelfs plug-ins mogelijk waarmee X3D-bestanden geëxporteerd kunnen worden van de unreal-editor [29]. Het door ons gebruikte object (het vliegtuig in [Figuur 46]), is te vinden bij de meegeleverde voorbeeldbestanden van de toolkit. Een laatste stap in onze demo bestond uit het maken van de koppeling tussen het X3D-bestand en een pythonscript. In X3D kunnen we “routes” creëren die nodes in X3D verbinden met procedures in een pythonscript en vice versa, dit zorgt ervoor dat er geavanceerdere procedures mogelijk zijn. In onze demo kan er door

middel van een muisklik een object in de scenegraph worden toegevoegd. Het resultaat van een reeks muisklikken is te zien op [Figuur 47].



**Figuur 47: Het resultaat van een reeks acties in X3D gekoppeld aan een pythonscript**

De routes en het pythonscript om dit te verwezenlijken zijn te vinden in [Bijlage 5]. De werking van een applicatie met H3D via X3D en pythonscript wordt op [Figuur 48] schematisch voorgesteld.




**Figuur 48: De werking van H3D met X3D en pythonscript**

Op [Figuur 48] zien we de aparte opzet van H3D. We zien dat H3DLoad start met een X3D-bestand, dat verwijzingen kan bevatten naar verschillende andere X3D-bestanden en pythonscripts. Ieder genest X3D-bestand kan zo verder verwijzingen bevatten. Via een X3D-bestand kunnen dan via een pythonscript en de bijhorende routes meer geavanceerde procedures worden geïmplementeerd. H3DLoad zal de rendering van zowel de grafische als de haptische scene volledig voor zijn rekening nemen. Bij H3D moeten we een scene daarom maar éénmalig definiëren.

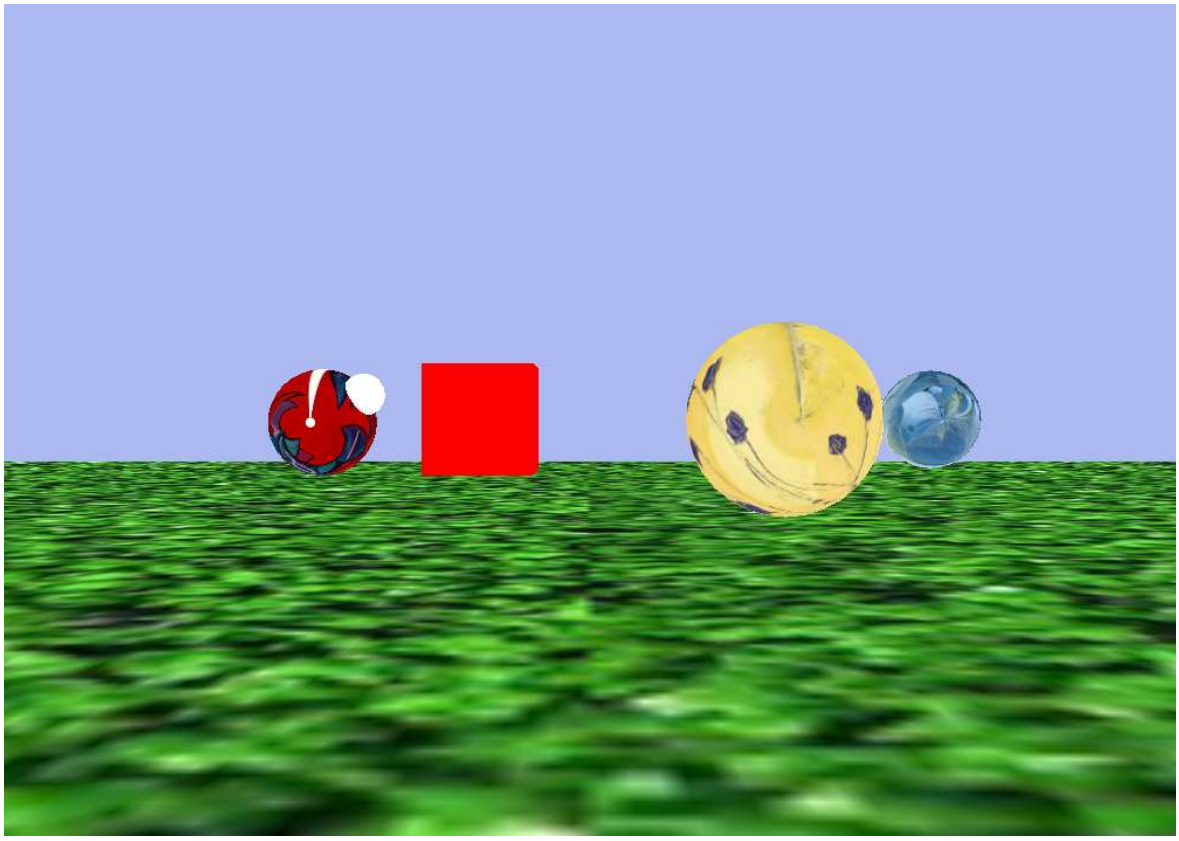
### Conclusie

Na het maken van deze demo zien we dat H3D tal van mogelijkheden bezit. Net als bij CHAI 3D is H3D gemaakt voor applicaties die van nul ontwikkeld dienen te worden. Bij het ontwikkelen in H3D is te zien dat het maken van een scene zeer snel kan gebeuren door gebruik te maken van X3D-bestanden, en eveneens dat er voor X3D zeer veel ondersteuning te vinden is. Indien er iets niet gemaakt kan worden met de reeds standaard voorziene functionaliteiten, kan er gebruik gemaakt worden van C++ om zo zelf nieuwe nodes in de scenegraph te definiëren en hier dan extra functionaliteit aan te koppelen. De meest opvallende eigenschap was dat H3D zeer vlot werkt en dat het zeer goed te gebruiken is voor rapid prototyping, iets wat op de site van H3D ook duidelijk wordt geaccentueerd. In onze demo hadden we snel objecten in X3D, maar de koppeling van de routes en het pythonscript was wat onduidelijk, grotendeels door de door onze gebrekkige kennis van pythonscript. De krachten en de koppeling met het haptische apparaat waren wel zeer precies en hier werden geen problemen ondervonden.

## *HAL v0.1.1*

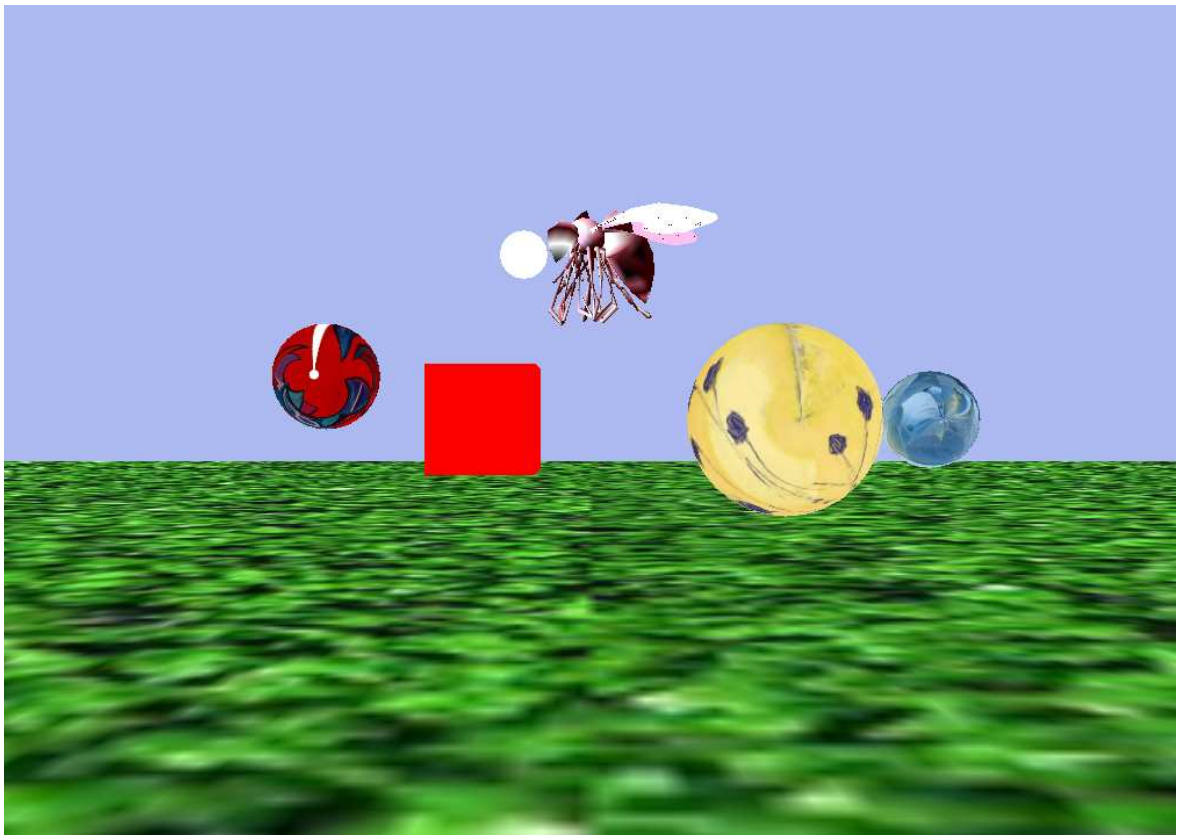
De laatste API waarmee we een demo zullen maken is die van HAL. HAL is beschikbaar onder de GPL en is dus ook open source. HAL is een toolkit die net als SensAble OpenHaptics enkel voorziet in haptisch renderen. Om met HAL te werken dient er met C/C++ geprogrammeerd te worden. HAL ondersteunt momenteel enkel het PHANToM apparaat en doet dit via de GHOST I/O wrapper. Indien hier geen beschikking over bestaat, kan er gewerkt worden met een pseudodevice dat bediend kan worden met het toetsenbord. De demo die we zullen maken zal een scene voorstellen die lijkt op die van de vorige demo's. Als window toolkit gebruiken we opnieuw Trolltech QT, voor het renderen van de grafische scene maken we gebruik van OpenGL en de haptics zullen door HAL

worden uitgevoerd. De door ons initieel opgebouwde scene is zichtbaar op [Figuur 49].



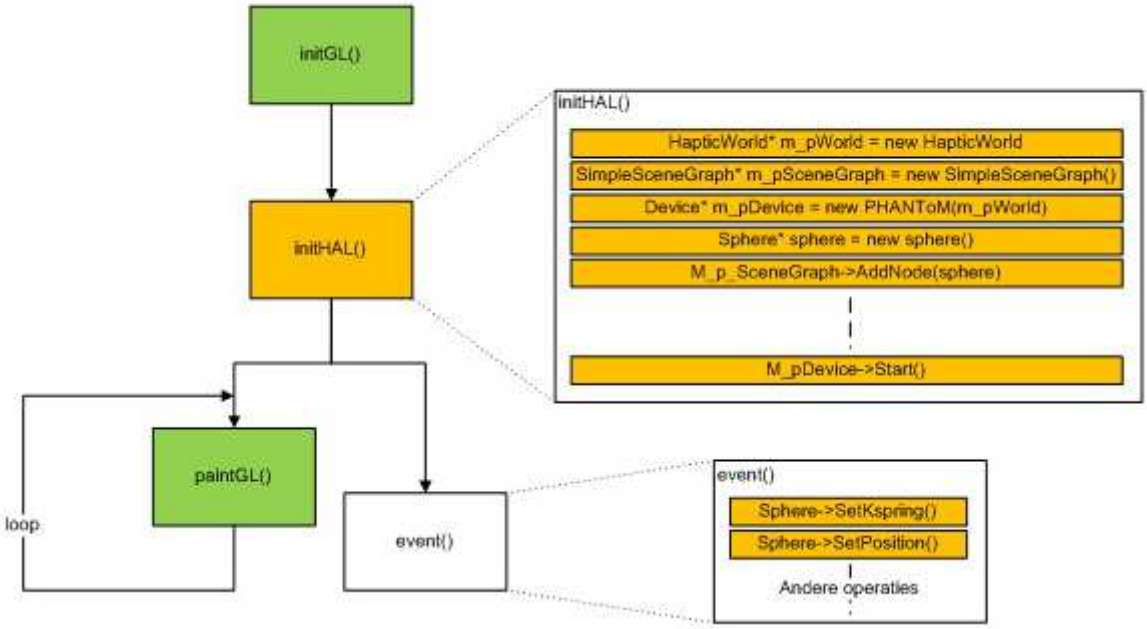
**Figuur 49: Een scene voorzien van haptics met HAL**

De scene op [Figuur 49] is voorzien van enkele primitieven, die bestaan uit een reeks statische spheres en een kubus. Voor het grafisch renderen zijn we hier volledig gebonden aan de mogelijkheden van OpenGL, nadien wordt er met HAL een haptische scenegraph opgebouwd, waar we de objecten van de grafische scene in zullen stockeren. HAL ondersteunt standaard een reeks primitieven bestaande uit: Een sphere, een kubus, een triangle en een polymesh. In de scene op [Figuur 49] dienen we enkel nog de haptische cursor te renderen. De code hiertoe is te vinden in [Bijlage 6]. Het opbouwen van de grafische scene was analoog aan deze van SensAble OpenHaptics. Het opbouwen van de haptische scene vertoonde daarentegen meer gelijkenissen met die van CHAI 3D. We dienen wel op te merken dat HAL meer primitieven ondersteunt dan CHAI 3D. Een volgende stap bestond erin een polymesh te renderen. Een polymesh in HAL is eigenlijk niets meer dan een normale trianglemesh. Het resultaat van het renderen van zo'n object is zichtbaar op [Figuur 50].



**Figuur 50: Een scene voorzien van een polymesh**

Vervolgens werd er naar de materiaaleigenschappen gekeken. De grafische materiaaleigenschappen zijn volledig in te stellen via OpenGL, de haptische eigenschappen zijn mogelijk gemaakt door middel van twee methods. Zo kan er op de node van een object in HAL een veerfactor worden ingesteld; "`SetKSpring()`" en een dempingsfactor; "`SetKDamping()`". Een andere opmerking die we dienen te maken is dat alle maten van OpenGL gedeeld moeten worden door een factor 10 om correct te mappen met de wereld van HAL. Een sphere met radius 10 in OpenGL stemt dus overeen met een sphere van radius 1 in HAL. Tenslotte werd er ook nog een object geanimeerd in de scene, de uiterst linkse sphere op [Figuur 49] en [Figuur 50] beweegt op en neer. In OpenGL dient dit per frame aangepast te worden en dit wordt gemapt door de corresponderende node in HAL per paint-event in OpenGL te transleren. Ook navigatie werd in een elementaire vorm voorzien, maar deze is geheel gekoppeld aan OpenGL en dus niet relevant voor deze haptische studie. De werking van een applicatie die gebruik maakt van HAL wordt geschetst in [Figuur 51].




**Figuur 51: Een applicatie gebruikmakend van HAL**

In het groen schetsen we de processen voor het grafisch renderen. In de demo gebruiken we hier dus OpenGL voor, maar er kan zonder probleem een andere API gebruikt worden, zolang er maar een correcte mapping gedaan wordt. In het oranje duiden we de HAL code aan, zoals te zien moeten we de statische scene van HAL eenmalig definiëren en enkel de veranderingen aan de scene dienen nadien te worden doorgegeven aan de scenegraph. Het event in onze schets kan van verschillende aard zijn, bijvoorbeeld van een keyboardevent of een animatieloop. Op de schets merken we dat HAL uitsluitend gebruikt wordt voor haptics en er dus een zeer losse koppeling is met andere API's. Van alle besproken API's in de demo's is HAL bijvoorbeeld de enige die rechtstreeks voor Microsoft Direct3D gebruikt zou kunnen worden.

## Conclusie

Een scene in HAL implementeren is te vergelijken met de combinatie van SensAble OpenHaptics en CHAI 3D. Het grafisch renderen staat volledig los van het haptisch renderen. In onze demo hebben we voor het grafisch renderen gebruik gemaakt van OpenGL, maar ook andere API's kunnen worden gebruikt. Zo zou het eveneens mogelijk zijn om HAL in combinatie te gebruiken met Microsoft Direct3D. Er kan geopteerd worden voor eender welke grafische API zolang er maar een correcte mapping gemaakt wordt met HAL. Het haptisch renderen werkt volledig met scenegraphs, dit werkt eigenlijk vlot en we hebben hier geen problemen bij het implementeren gehad. Qua mogelijkheden biedt HAL vrijwel hetzelfde als de andere API's, buiten het feit dat er geen eigenschappen mogelijk zijn voor het instellen van de frictie op een object. We zien dat HAL veel

vrijheid toelaat omtrent de toolkits waarmee het in combinatie gebruikt wordt. De enige beperking momenteel is dat enkel de PHANToM als haptisch apparaat gebruikt kan worden.

## *Featurevergelijking en benchmarks*

Nu we met iedere toolkit wat ervaring hebben opgedaan, kunnen we een vergelijking gaan maken. We zullen de toolkits vergelijken op het vlak van:

- Documentatie
- Gebruiksgemak
- Features
- Performantie

Na het bespreken van iedere toolkit per item zullen we tot een conclusie trachten te komen, om zo te zien welke van de API's we het meest kunnen aanraden.

### Documentatie

Bij documentatie zullen we voor iedere gebruikte toolkit bespreken wat er qua documentatie te vinden is. Een voorbeeld hiervan zijn online manuals en fora.

#### *SensAble OpenHaptics Academic Edition v2.0*

De SensAble OpenHaptics toolkit beschikt ruwweg over de grootste hoeveelheid aan documentatie van alle toolkits die we besproken hebben. De reden hiervoor is wellicht dat SensAble het langst op deze markt actief is en tevens de fabrikant is van het meest bekende haptische apparaat: de PHANToM. Op de site van SensAble vinden we een overvloed aan informatie. Zo beschikt de gebruiker na registratie over verschillende online fora, dit zowel voor software- als voor hardwarediscussies. De fora voor OpenHaptics zijn ook vrij actief en er is een grote userbase van meer dan 1000 leden. Anderzijds is er veel statische informatie te vinden; zo is er een programmer's guide en een API reference te downloaden. Voorts worden bij de toolkit verschillende voorbeelden meegeleverd, dit zowel voor HDAPI als voor de HLAPI. Op de website staan daarenboven nog een groot aantal 3rd-party voorbeelden. Tenslotte zijn er ook verschillende papers

en boeken aan de toolkit gewijd; de verwijzingen hiernaar worden op de website van SensAble gemaakt.

### *CHAI 3D v1.6.1*

Net als voor OpenHaptics is er voor CHAI 3D een site met informatie te vinden, zij het met een kleinere userbase. De site biedt een forum aan, maar dit is blijkbaar al even buiten gebruik, de laatste post dateert namelijk van november 2006. De laatste versie van CHAI 3D (v 1.6.1) is gelukkig recenter, deze dateert van juni 2007. Verder is er wel voldoende statische informatie op de site te vinden. Zo zijn er zowel op de site als bij het downloaden van de API tal van voorbeelden. Op de website staat ook nog een API reference. Er zijn ook tal van presentaties te vinden, maar deze zijn geen echte meerwaarde voor het leren werken met de toolkit. Anderzijds moeten we wel opmerken dat CHAI 3D volledig open source is waardoor er ook altijd naar de code zelf gekeken kan worden.

### *SenseGraphics H3D API v1.5*

Documentatie voor de SenseGraphics H3D API is gratis te verkrijgen via de website van H3D. Op de site zijn er verscheidene fora te vinden met een actieve en grote userbase. Ook statische informatie wordt aangeboden, zo kan er een API reference gedownload worden, alsook een kleine starters guide. De code van H3D is net als die van CHAI 3D open source en kan dus ook als bron van informatie dienen. Indien dit niet voldoende zou zijn, biedt SenseGraphics verschillende betaalde ondersteuningsformules aan. Deze bieden onder andere een e-maildienst voor problemen en voor de premium versie is er ook telefonische hulp.

### *HAL v0.1.1*

De HAL library biedt vrij weinig documentatie. Er is wel een website waar de API gedownload kan worden, maar er is verder geen informatie beschikbaar op deze site. De library zelf biedt gelukkig wel een voorbeeld waar we vrijwel alle informatie uit kunnen halen. Eveneens is net zoals CHAI 3D en H3D ook deze toolkit volledig open source, zodat hier ook direct naar de code gekeken kan worden of via doxygen een API reference kan worden gecreëerd.

### *Conclusie*

Qua documentatie kunnen we besluiten dat de API van SensAble OpenHaptics de beste ondersteuning geeft. Bij OpenHaptics worden de meeste voorbeelden geleverd en het grootste pluspunt is dat SensAble beschikt over de grootste userbase. Ook SenseGraphics H3D beschikt over goede documentatie, ook hier biedt de site veel informatie en ondersteuning. CHAI 3D is iets minder om mee te leren werken, aangezien de fora hier niet erg actief zijn, en er wordt niet veel gedaan aan de API zelf. Hoewel er wel recent een nieuwe versie van CHAI 3D beschikbaar gesteld is, is deze versie grotendeels een compatibiliteitsupdate voor nieuwe compilers. De vorige update is reeds een jaar oud en de updates voordien waren veel frequenter dan nu het geval is. HAL biedt het minste informatie, maar de voorbeelden bieden wel een goed uitgangspunt.

## Gebruiksgemak

Bij gebruiksgemak zullen we bespreken hoe makkelijk een API werkt. Ook wordt er gekeken of de API vlot te integreren valt en wat het gebruiksgemak is.

### *SensAble OpenHaptics Academic Edition v2.0*

Een der grootste pluspunten van SensAble OpenHaptics is dat de toolkit zeer vlot te integreren valt met reeds bestaande OpenGL applicaties. De code van OpenHaptics is gemaakt om op OpenGL te gelijken, wat er voor zorgt dat mensen die reeds ervaren zijn met OpenGL sneller kunnen werken met OpenHaptics. Deze eigenschappen zijn niet enkel bruikbaar voor integratie in reeds bestaande OpenGL applicaties, maar ook voor applicaties die met OpenGL en OpenHaptics zullen werken die van nul worden ontworpen. Anderzijds is de grote samenhang van OpenHaptics ook een van de nadelen, hoewel OpenGL een industrie standaard gevestigde API is, zijn er toch andere 3D toolkits. Zo is er Microsoft Direct3D, dat aan een sterke opmars bezig is. De API biedt niet enkel beperkingen ten opzichte van 3D API’s, maar ook ten opzichte van het gebruik van objectgeoriënteerde scenegraphs voor graphics. Er wordt getracht dit te overkomen door middel van nieuwe API’s zoals H3D die gebruik maken van HDAPI voor scenegraphs. Ook biedt OpenHaptics geen voorgedefiniëerde functies voor het inlezen van meshes of texturen; de programmeur dient zelf in dit alles te voorzien. De toolkit voorziet ook bestanden voor alle recente versies van MSVC. Een ander nadeel aan de toolkit is dat deze enkel met de haptische apparaten van SensAble zelf werkt.

### *CHAI 3D v1.6.1*

CHAI 3D biedt een uniforme aanpak voor zowel graphics als haptics. In tegenstelling tot SensAble OpenHaptics kan de scene hier eenmalig gedefinieerd worden, waarna CHAI 3D voor de verschillende renderings zal zorgen. Ook voorziet CHAI 3D zelf in functies voor het inlezen van 3D objecten zoals die van 3D Studio Max en Maya (*.3ds, *.obj). Ook hier zitten we net als bij OpenHaptics met de voor- en nadelen van een scenegraph. CHAI 3D is volledig gebonden aan zijn eigen scenegraph. CHAI 3D kan dus niet zomaar gebruikt worden in een reeds bestaande applicatie. Er kan weliswaar aan scenegraph matching gedaan worden, maar dit is moeilijker toe te passen en komt de performantie niet ten goede. Anderzijds biedt CHAI 3D dan wel weer de mogelijkheid om te werken met verschillende haptische apparaten. Sinds versie v1.6.1 zijn ook voor alle veelgebruikte compilers bestanden voorzien.

### *SenseGraphics H3D API v1.5*

H3D heeft een soortgelijke aanpak als CHAI 3D, enkel dat er hier verder gebouwd werd met SensAble OpenHaptics. H3D zorgt ervoor dat er zeer snel applicaties kunnen worden gemaakt door gebruik te maken van X3D-bestanden en H3DLoad. Zo kunnen ook mensen die niet erg veel kennis hebben van programmeren toch een prototype maken. Op de website wordt deze eigenschap ook specifiek aangeprezen. Een groot voordeel is ook dat de grafische objecten met veel tools gemaakt kunnen worden, zolang de tools maar X3D ondersteunen. Voor meer geavanceerde functies kan er gebruik worden gemaakt van pythonscript; de bruikbaarheid hiervan is natuurlijk afhankelijk van de kennis die de programmeur heeft van deze scripttaal. Voor verdere functionaliteiten kan er ook altijd met C++ gewerkt worden.

### *HAL v0.1.1*

Het gebruiksgemak van de HAL toolkit is te vergelijken met dat van CHAI 3D maar dan puur voor de haptische scene. Voordeel hiervan is dat eender welke grafische API kan worden gekozen, zolang de scenes maar gesynchroniseerd worden. Het haptische apparaat dat er gebruikt wordt, wordt aangesproken via de Ghost I/O wrapper, maar dit houdt in dat er enkel gebruik kan worden gemaakt van de PHANToM apparaten. Voor het compileren met verschillende compilers waren er geen noemenswaardige problemen.

***Conclusie***

Het gebruiksgemak is afhankelijk van verschillende factoren. Zo is het altijd de vraag of er reeds een applicatie bestaat of niet, is er reeds een bestaande applicatie met OpenGL gemaakt, dan kan er best gebruik worden gemaakt van SensAble OpenHaptics of van HAL. Is er een bestaande applicatie met een andere grafische API, dan biedt HAL het beste alternatief. Moet er snel een prototype gemaakt worden, dan is SenseGraphics H3D de beste oplossing. Wordt er een applicatie van nul ontworpen, dan kan er een keuze worden gemaakt tussen alle API's die volledig afhankelijk is van wat de eisen zijn van de applicatie en die wellicht zeer afhankelijk is van de voorkeuren van de programmeurs. Eveneens kan de rol van het te gebruiken haptische apparaat meespelen.

## Features

In deze sectie zullen we een tabel opstellen met alle mogelijke features per API. We zullen op die manier trachten een overzicht te geven van welke API welke features al dan niet ondersteunt.

| | **SensAble OpenHaptics Academic Edition v2.0** | **CHAI 3D v1.6.1** | **Sense-Graphics H3D v1.5** | **HAL v0.1.1** |
|---|---|---|---|---|
| **Open source** | Neen | Ja | Ja | Ja |
| **Scenegraph voor haptische rendering** | Neen | Ja | Ja | Ja |
| **Scenegraph voor grafische rendering** | Neen | Ja | Ja | Neen |
| **Unified scenegraph structuur** | Neen | Ja | Ja | Neen |
| **Ondersteunde primitieven** | Alle primitieven via de OpenGL API | Triangles, torussen en spheres | Kubussen, spheres, cilinders, kegels, torussen, triangles en lijnen | Triangles, kubussen en spheres |

| | | | | |
|---|---|---|---|---|
| **Trianglemeshes** | Ja, via OpenGL | Ja | Ja | Ja |
| **Inlezen objecten** | Neen | Ja, *.xml, *.3ds, *.obj | Ja, via *.x3d | Neen |
| **Materiaalproperties** | Ja, stiffness, damping, friction en popthrough | Ja, stiffness, dynamic friction, static friction | Ja, friction, magnetisme | Ja, veerconstante en damping |
| **Forcefields** | Ja | Ja | Ja | Ja |
| **Ondersteunde haptische apparaten** | SensAble PHANToM | SensAble PHANToM, Force dimension Omega en Delta, MPB Freedom 6, FCS Haptic Master, “thinking about haptics”, I/O boards | SensAble PHANToM, Force dimension Omega en Delta | SensAble PHANToM |
| **Mogelijkheid te werken zonder haptisch apparaat** | Neen | Ja | Ja | Ja |
| **Ondersteuning pseudodevice** | Neen | Ja | Neen | Ja |
| **Directe apparaatcontrole** | Ja, via HDAPI | Ja | Ja, via HDAPI | Ja, via Ghost I/O wrapper |
| **Gebruikte programmeertaal** | C/C++ | C/C++ | C/C++ | C/C++ |
| **Ondersteuning scripttalen** | Neen | xml | Python, X3D | Neen |
| **Ondersteunde besturingsystemen** | Win32, Linux, Mac OS X | Win32, Linux, Mac OS X, QNX, RTAI | Windows XP, Linux, Mac OS X | Win32 |

**Tabel 1: De featurelisting per haptische toolkit**

### *Conclusie*

Na het maken van dit overzicht, zien we dat de grootste verschillen tussen de API’s de volgende zijn:

De API beschikt al dan niet over grafische rendering: SensAble OpenHaptics en HAL hebben geen grafische scenegraph, ze doen ook geen grafische rendering. Dit kan een voor- of nadeel zijn. In [42] is er sprake dat sommige programmeurs resoluut voor een scenegraph kiezen, terwijl anderen dit niet doen. OpenHaptics gebruikt nergens een scenegraph en is tegelijk sterk verbonden met OpenGL. Wordt er geen gebruik gemaakt van OpenGL, dan moet er HDAPI gebruikt worden, en moet er zelf voor de collisiedetectie gezorgd worden. HAL geeft meer vrijheid op dit vlak, want het gebruikt een scenegraph voor het haptisch renderen van de scene, maar laat alle vrijheid voor het grafisch renderen.

De API ondersteunt niet alle haptische apparaten: SensAble OpenHaptics ondersteunt enkel de PHANToM reeks, ook HAL ondersteunt enkel de PHANToM. De andere API's ondersteunen verschillende apparaten. Afhankelijk van het type apparaat dat er gebruikt wordt, kan dit tot een groot nadeel leiden.

De API kan gebruikt worden zonder haptisch apparaat en voorziet eventueel in een pseudodevice: Bij het maken van de demo's was het zonder haptisch apparaat weliswaar altijd mogelijk om de applicatie te compileren, maar niet om deze dan te testen. SensAble OpenHaptics zal zonder haptisch apparaat zelfs gewoon niet werken. De andere toolkits werkten wel zonder haptisch apparaat, wat altijd handig was om even de huidige staat van de applicatie te testen. HAL en CHAI 3D bieden hier als enigen een pseudodevice aan dat zeer handig was voor het ontwikkelen van een applicatie indien er niet altijd een haptisch apparaat ter beschikking stond.

De API ondersteunt speciale objecten en scripttalen: We zien ook dat de API's die beschikken over grafische rendering veelal voorzien zijn van extra features voor het inlezen van 3D objecten, of om zelfs een volledige scene via een bestand in te lezen. Dit komt CHAI 3D en SenseGraphics H3D ten goede omdat deze een totaalpakket trachten te bieden. Bij SensAble OpenHaptics en HAL moet dit zelf worden voorzien, maar afhankelijk van het doel van de toolkit kunnen we dit niet echt als voor- of nadeel bestempelen.

Voorts zijn er nog kleine verschillen tussen de API's, maar deze zijn niet onoverkomelijk. Voorbeelden hiervan zijn de verschillende primitieven die er ondersteund worden, die immers altijd via een trianglemesh kunnen worden ingelezen.

## Performantie

De laatste vergelijking van de geïmplementeerde demo's dient ertoe een beeld te schetsen van de prestaties van iedere toolkit. De toolkits zullen met een aangepaste demo getest worden en we zullen zo trachten weer te geven welke van de API's het best presteert op vlak van resources. Voor elke benchmark zal een zo exact mogelijke scene gerenderd worden, dit zowel grafisch als haptisch. Er werden op deze manier drie scenario's ontwikkeld:

- Een scene met 1 kubus zonder textuur.
- Een scene met 1 kubus en textuur.
- Een scene met 100 kubussen zonder textuur.

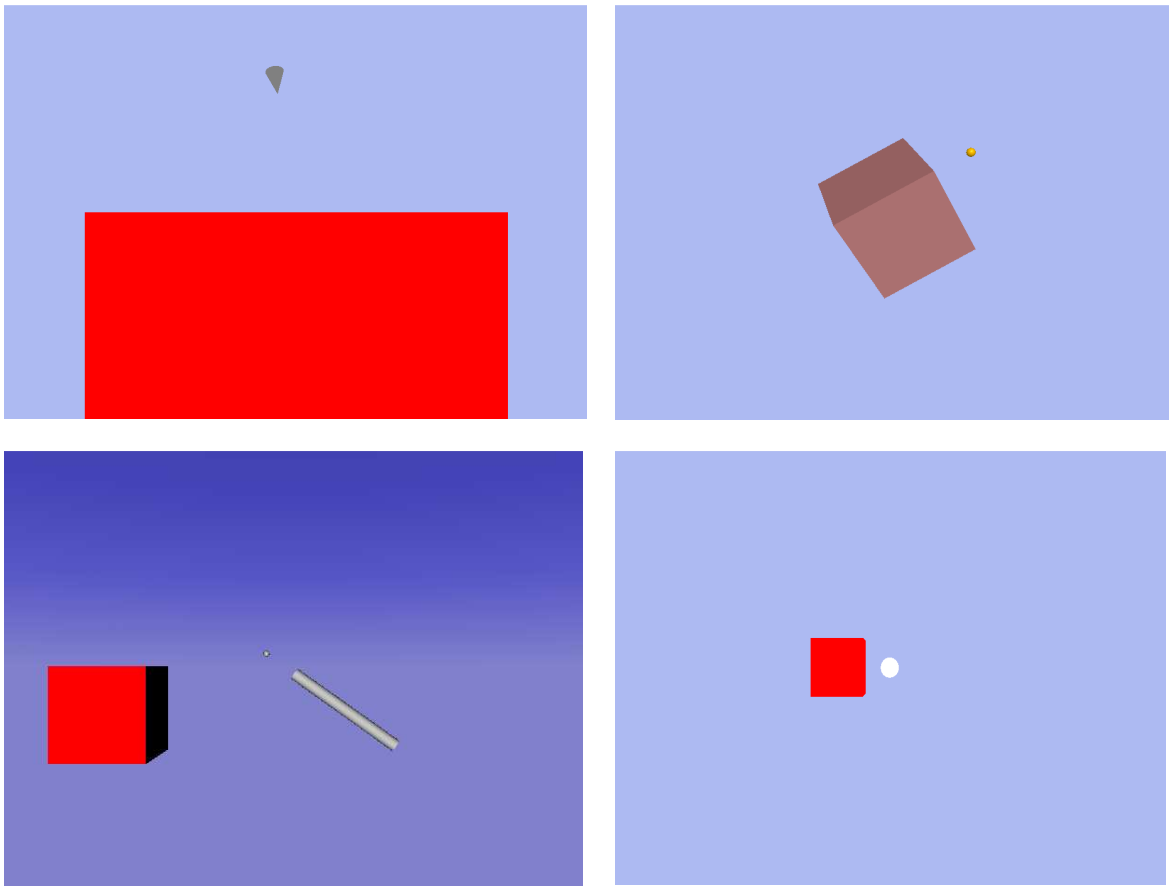
Deze drie scenario's dienen ertoe een beeld te scheppen van hoe iedere toolkit omspringt met de beschikbare resources op vlak van cpu-vermogen en de memory footprint. Voor ieder van deze scenario's werd een demo van al de API's ontwikkeld die zo licht mogelijk gemaakt werd, hiermee bedoelen we dat de demo is ontdaan van overbodige code en klasses. Zo is er in de benchmarkdemo's bijvoorbeeld al geen navigatie meer mogelijk. De pc die er gebruikt werd voor het uitvoeren van deze benchmarks is voorzien van volgende hardware:

- Dual Intel Xeon cpu 2.4 Ghz, 512 Kb cache
- Front side bus 533 Mhz
- 2 x 512 Mb DDR RAM 266 Mhz
- nVidia Quaddro4 900 XGL vga-kaart, AGP, 128 Mb DDR
- Microsoft Windows XP Professional Edition, 32-bit versie
- Western Digital WD1200JB-75CRA Harde schijf, 7200 rpm, 120 Gb, 4 Mb cache
- SensAble PHANToM Premium 1.5

De cpu load en de memory footprint werden gemeten met de Windows taskmanager processes en performance monitor. Voor ieder scenario zullen deze parameters gemeten worden met de haptische manipulator zowel in contact met het object als niet in contact. Ook zal de haptische loop gemeten worden. We zullen deze meten bij al de API's buiten SenseGraphics H3D, omdat deze API getest werd met scripts en niet via C++, en ook omdat SenseGraphics H3D reeds gebruik maakt van SensAble OpenHaptics.

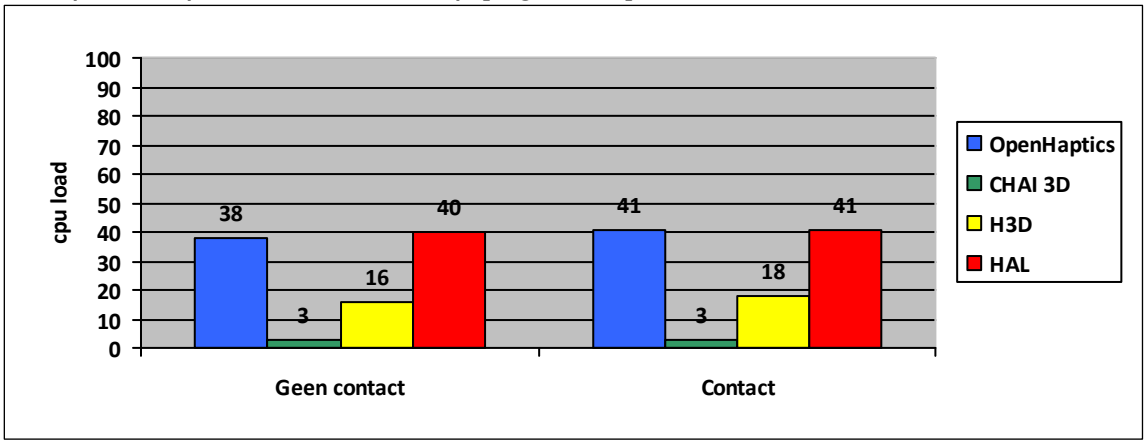
### *Scenario 1: 1 kubus zonder textuur*

Scenario 1 zal de performance meten van iedere API met een zo simpel mogelijk object en zonder texturen. De benchmarkdemo's voor iedere API zijn te zien op [Figuur 52].



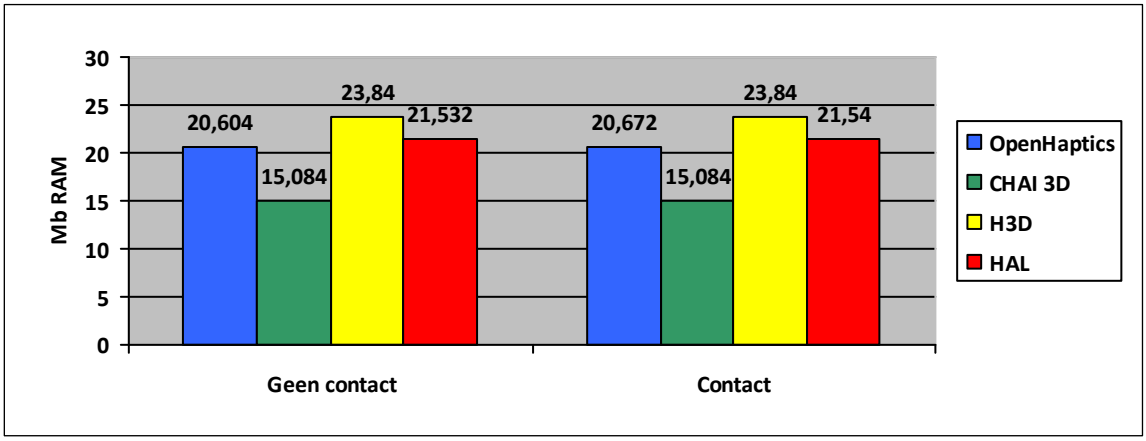
**Figuur 52: Linksboven: SensAble OpenHaptics. Rechtsboven: CHAI 3D. Linksonder: SenseGraphics H3D. Rechtsonder: HAL**

De cpu load per API is te zien op [Figuur 53].




**Figuur 53: De cpu load voor de verschillende API's in scenario 1**

Op [Figuur 53] is duidelijk te zien dat CHAI 3D spaarzamer omgaat met cpu cycles. We moeten hier wel het onderscheid maken tussen de API's die zelf voorzien in het grafisch renderen. CHAI 3D en SenseGraphics H3D hebben duidelijk een lagere cpu load, beide API's zien ook zelf toe op hun grafische rendering. Het verschil in cpu load is wellicht deels te verklaren door de optimalisatie die deze API's doen voor de grafische rendering. Vergelijken we Sensable OpenHaptics en HAL, dan zien we dat deze niet erg veel verschillen qua cpu load, we merken wel dat de cpu load bij alle API's met uitzondering van HAL en CHAI 3D toeneemt indien er een botsing optreedt. Ook is er een opvallend verschil tussen de API's met eigen grafische rendering; zo is de cpu load van H3D veel hoger dan deze van CHAI 3D. [Figuur 54] zal een zicht geven op de memory footprint van de API's in scenario 1.




**Figuur 54: De memory footprint van de verschillende API's in scenario 1**

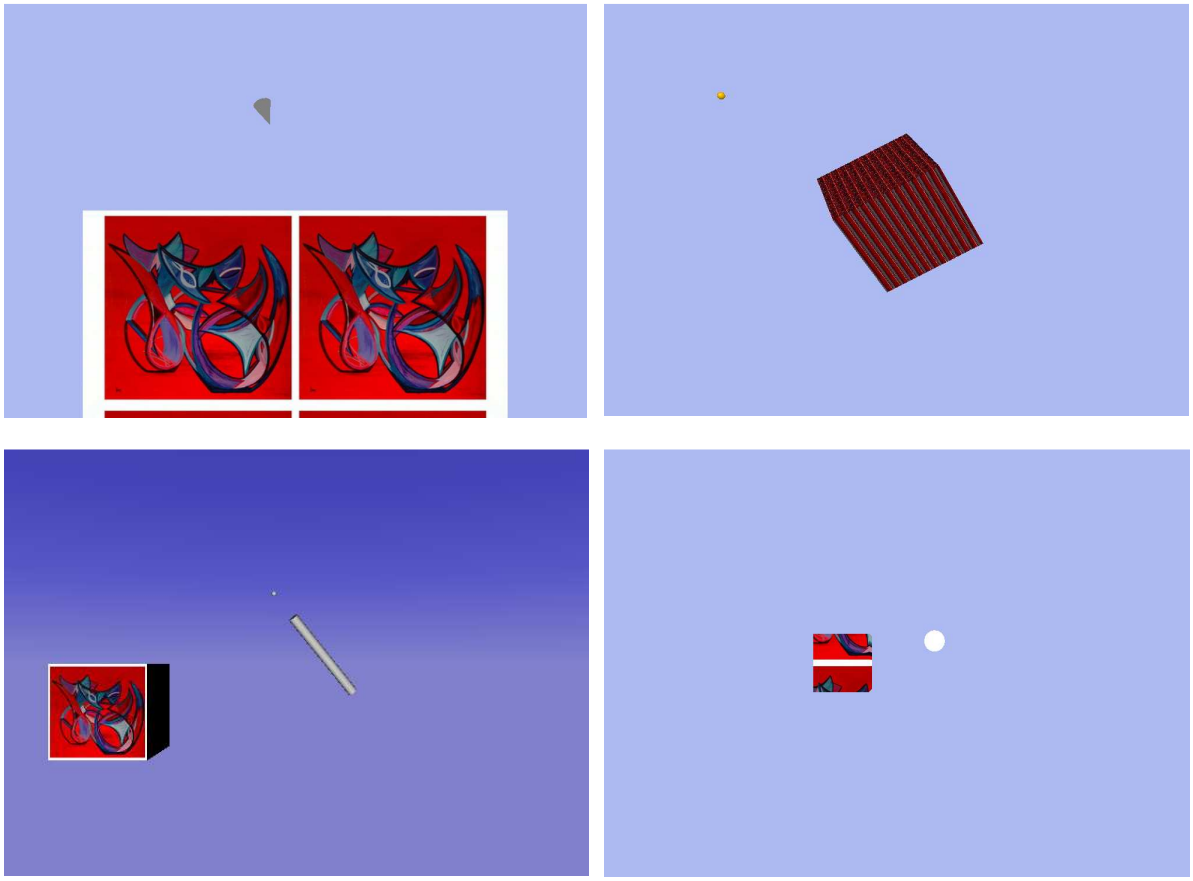
Op [Figuur 54] zien we dat de memory footprint vrij constant blijft, of er nu contact is of niet. Voorts zien we dat CHAI 3D ook hier het beste omspringt met de resources. Het grote geheugengebruik van SenseGraphics H3D is te verklaren door het feit dat H3D de volledige H3DLoad-applicatie altijd moet starten voor eender welke scene. Waar we bij de andere applicaties bijvoorbeeld de code van navigatie kunnen schrappen voor deze test, zal de H3DLoad-applicatie deze toch inladen. Ook de hogere cpu load is hier wellicht door te verklaren. Op [Figuur 55] wordt de duur van de haptische loop in kaart gebracht.


**Figuur 55: Het timen van de haptische loop in scenario 1**

De duur van de haptische loop die hier weergegeven wordt is het gemiddelde van een totaal van 250 gemeten opeenvolgende waarden. Zoals te zien op [Figuur 55] duurt het uitvoeren van de haptische loop het langst bij SensAble OpenHaptics, de haptische procedure neemt het minste tijd in beslag voor CHAI 3D, en die van HAL zit tussen de twee andere API's in. We zien ook dat bij alle hier gemeten API's de haptische loop gemiddeld langer duurt indien er contact tussen de objecten wordt gemaakt. We zien bij deze test ook het effect van het door ons geïmplementeerde grafisch renderen bij SensAble OpenHaptics en HAL. Zo is duidelijk te zien dat waar in [Figuur 53] de API's qua cpu load dicht bij mekaar lagen, deze API's op vlak van pure haptics toch een groter prestatieverschil vertonen.

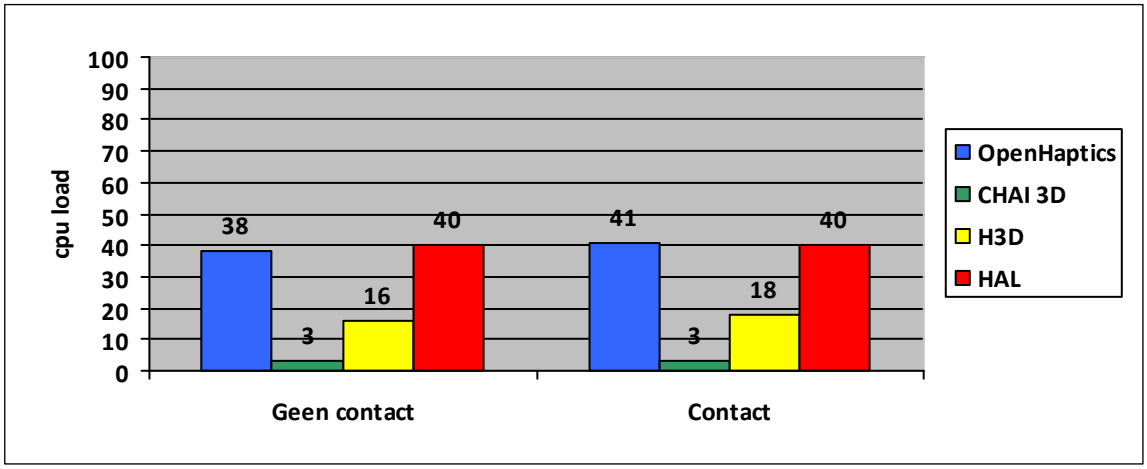
### *Scenario 2: 1 kubus met textuur*

Scenario 2 dient er vooral toe te kijken naar de voordelen van de API's met grafische rendering en deze zonder. Zo zullen we door texturen te gebruiken zien hoe efficiënt de API's met beeldbestanden omgaan ten opzichte van de eigen implementering met OpenGL. Het beeldbestand dat we gebruiken is een PNG-bestand van 24,8 Kb. [Figuur 56] toont de benchmarkdemo's.



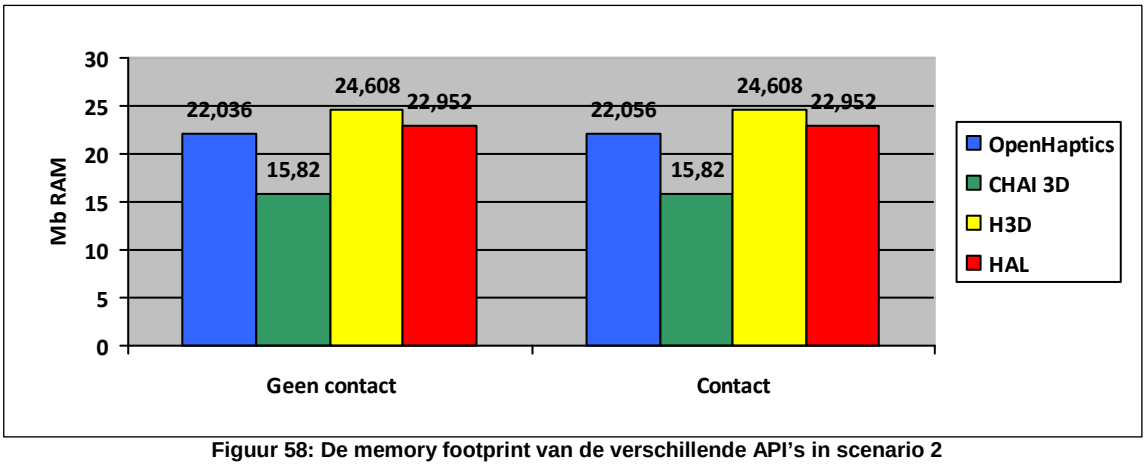
**Figuur 56: Linksboven: SensAble OpenHaptics. Rechtsboven: CHAI 3D. Linksonder: SenseGraphics H3D. Rechtsonder: HAL**

Op [Figuur 57] is de cpu load te zien van de API's in scenario 2.




**Figuur 57: De cpu load voor de verschillende API's in scenario 2**

We zien op [Figuur 57] dat er vrijwel geen verschil is qua cpu load in vergelijking met het scenario waarbij er geen texturen worden gebruikt. Op [Figuur 58] zien we de memory footprint van de toolkits in scenario 2.



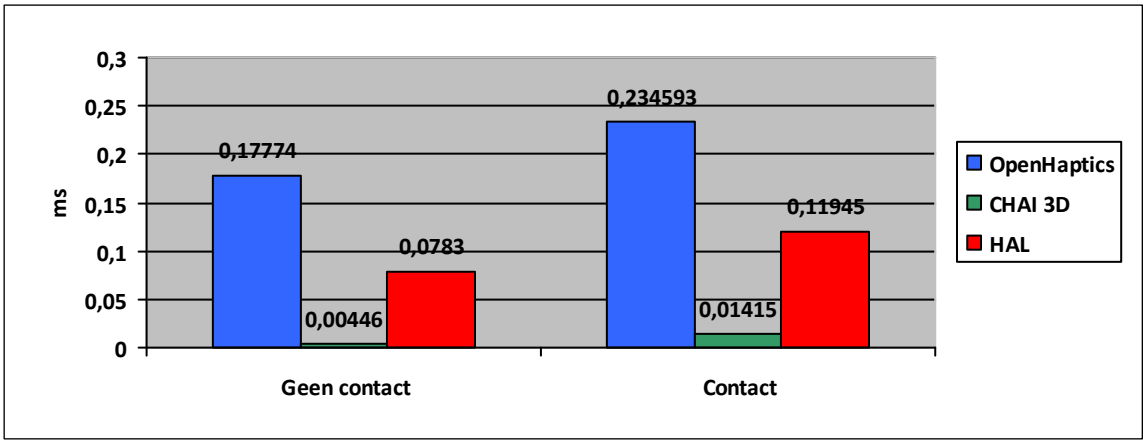

**Figuur 58: De memory footprint van de verschillende API's in scenario 2**

Op [Figuur 58] zien we dat het verschil tussen de API's - in contactmode of niet - vrijwel nihil is. Dit viel te verwachten aangezien het een volledig grafisch feature betreft. Het verschil in memory footprint met de test in scenario 1 zien we op [Tabel 2].

| | **Scenario 1** | **Scenario 2** | **Verschil** |
|---|---|---|---|
| **SensAble OpenHaptics** | 20,604 Mb | 22,036 Mb | 1,432 Mb |
| **CHAI 3D** | 15,084 Mb | 15,820 Mb | 0,736 Mb |
| **SenseGraphics H3D** | 23,840 Mb | 24,608 Mb | 0,768 Mb |
| **HAL** | 21,532 Mb | 22;952 Mb | 1,420 Mb |

**Tabel 2: De verschillen in memory footprint voor scenario 1 en 2**

In [Tabel 2] zien we dat de API's met geïntegreerde grafische rendering duidelijk veel efficiënter omspringen met het beeldbestand. De zelf geïmplementeerde grafische rendering neemt zowat dubbel zoveel resources in voor het beeldbestand. Wellicht kunnen we deze code nog optimaliseren, maar dit toont reeds aan dat SenseGraphics H3D en CHAI 3D zeer efficiënt zijn op dit vlak. We merken hier ook op dat we in [Tabel 2] de waardes genomen hebben voor in het geval er geen contact zou zijn. Op [Figuur 59] zien we de gemiddelde duur van de haptische loop.




**Figuur 59: Het timen van de haptische loop in scenario 2**

Net zoals bij [Figuur 55] zijn ook deze waardes de gemiddelden van 250 opeenvolgend gemeten waardes. De algemene resultaten zijn ook hier te vergelijken met deze van scenario 1. Zo duurt de haptische loop van SensAble OpenHaptics het langst, gevolgd door HAL en dan door CHAI 3D. Om een beter beeld te kunnen scheppen in vergelijking met scenario 1 tonen we de verschillen in [Tabel 3].

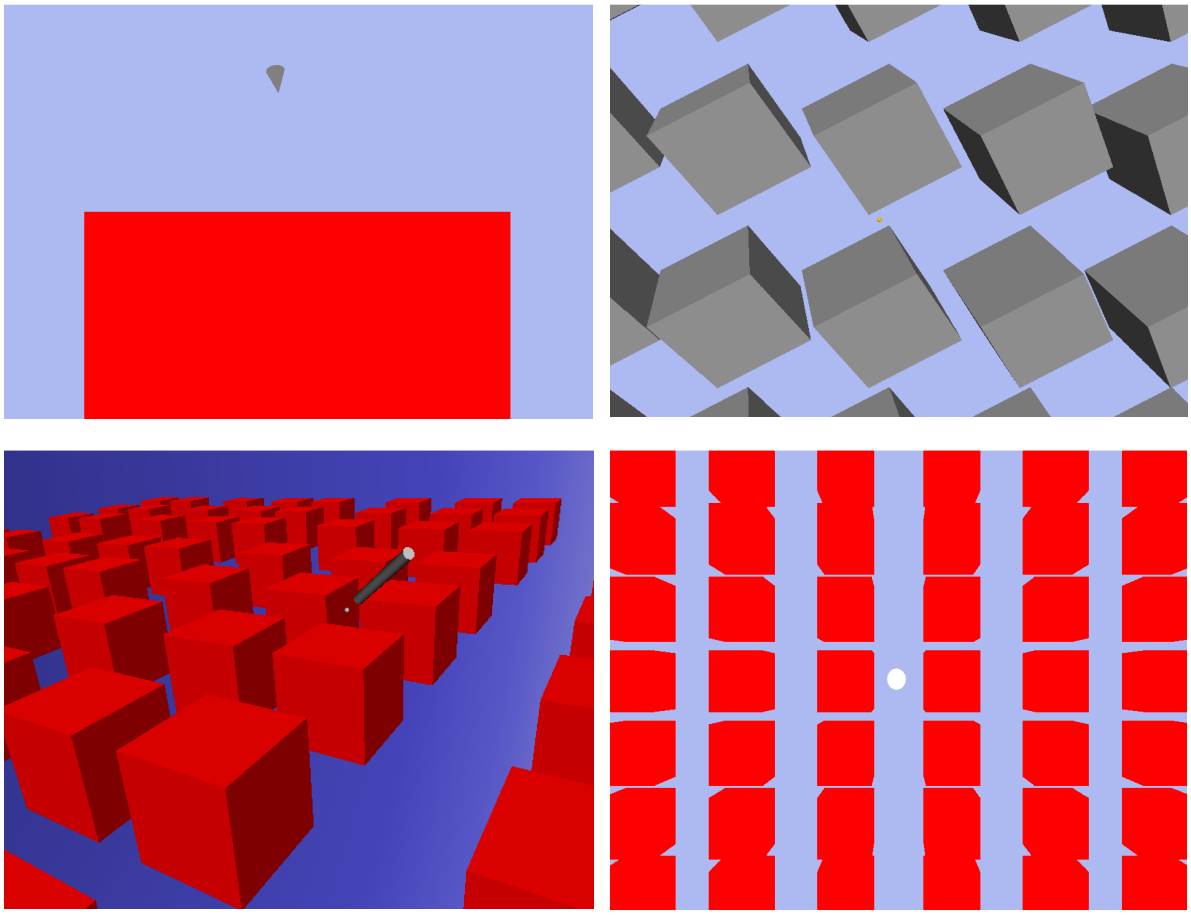
| | **Scenario 1** | **Scenario 2** | **Verschil** |
|---|---|---|---|
| **SensAble OpenHaptics** | 0,188 ms | 0,177 ms | -0,011 ms |
| **CHAI 3D** | 0,003015 ms | 0,00446 ms | 0,001445 ms |
| **HAL** | 0,086762 ms | 0,0783 ms | -0,008462 ms |

**Tabel 3: De verschillen in duur van de haptische loop voor scenario 1 en 2**

In [Tabel 3] zien we de verschillen tussen de haptische loops wanneer er geen contact is. Het grootste verschil bedraagt 8 microseconden, dus zoals reeds vermeld zijn deze verschillen miniem en te verwaarlozen. Het gebruik van texturen heeft volgens ons slechts een klein effect op het geheugengebruik.

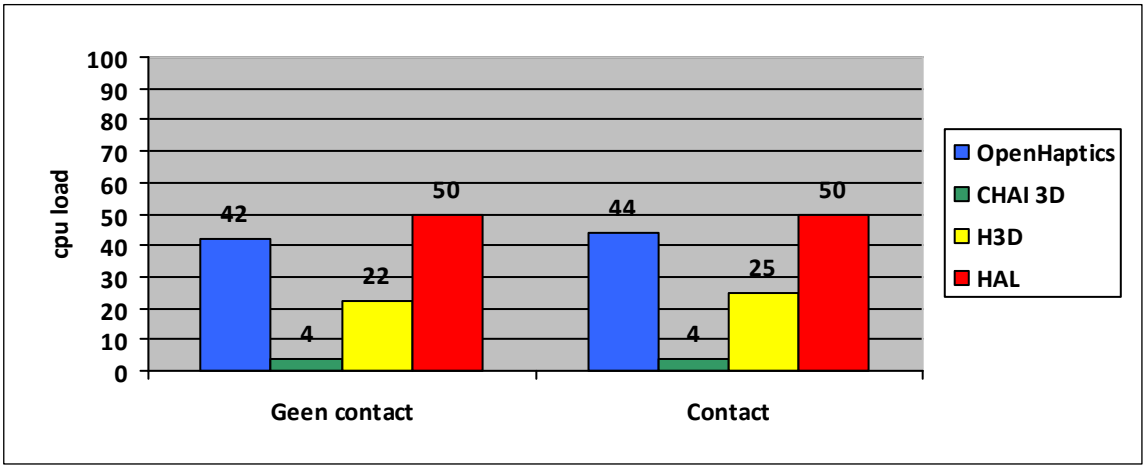
### *Scenario 3: 100 kubussen zonder textuur*

Scenario 3 dient ertoe meerdere objecten te renderen. We hebben reeds in scenario 2 gezien dat het effect van texturen enkel invloed heeft op de grafische loop, vandaar dat we bij dit scenario geen texturen zullen gebruiken. De scene van dit scenario is te zien op [Figuur 60].



**Figuur 60: Linksboven: SensAble OpenHaptics. Rechtsboven: CHAI 3D. Linksonder: SenseGraphics H3D. Rechtsonder: HAL**

We merken op dat het beeld van SensAble OpenHaptics op [Figuur 60] identiek is aan dit van scenario 1, omdat de andere kubussen uit het zicht staan en er geen navigatieklasse meer aanwezig is om dit beeld anders te oriënteren. Tevens staat de kubus zodanig opgesteld dat we deze toch konden aanraken met de haptische cursor. Op [Figuur 61] zien we de cpu load van scenario 3.



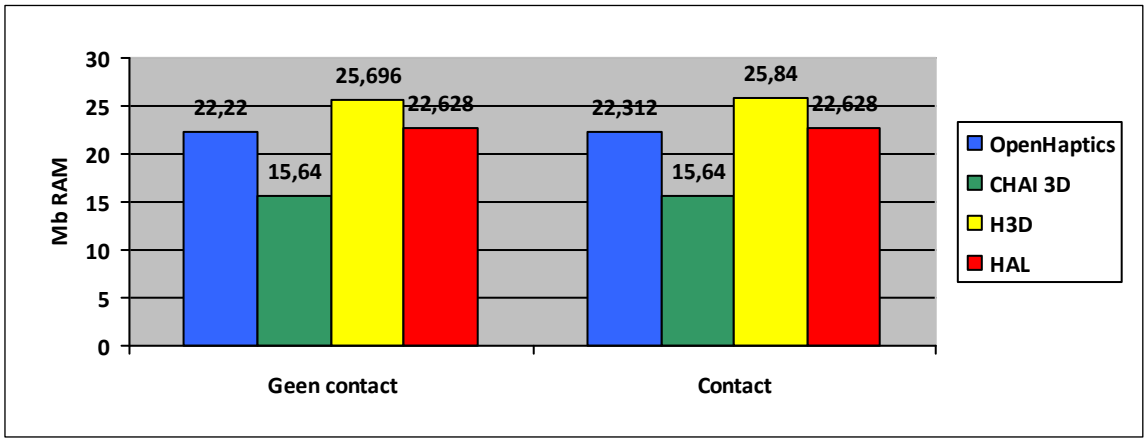

**Figuur 61: De cpu load voor de verschillende API's in scenario 3**

Op [Figuur 61] zien we dat de cpu load voor SensAble OpenHaptics en H3D licht toeneemt wanneer er contact is. CHAI 3D blijft constant, maar HAL heeft continu 50% cpu load. Tijdens de test werken de haptics bij alle API's, buiten die van HAL. Bij nader onderzoek zien we dat HAL 1 cpu thread volledig benut. HAL heeft duidelijk zijn maximum bereikt en aangezien de haptische loop niet snel genoeg meer verwerkt kan worden zijn de haptics uitgeschakeld. We zijn nog verder gaan testen naar het precieze punt waarop HAL zijn haptics uitschakelt. Zo blijkt dat HAL een maximum van 76 kubussen haptisch kon renderen op dit specifieke computersysteem. De verschillen in cpu load voor de verschillende scenario's zijn terug te vinden in [Tabel 4].

| | **Scenario 1 – geen contact** | **Scenario 3 – geen contact** | **Verschil** | **Scenario 1 - contact** | **Scenario 3 - contact** | **Verschil** |
|---|---|---|---|---|---|---|
| **SensAble OpenHaptics** | 38 % | 42 % | 4 % | 41 % | 44 % | 3 % |
| **CHAI 3D** | 3 % | 4 % | 1 % | 3 % | 4 % | 1 % |
| **SenseGraphics H3D** | 16 % | 22 % | 6 % | 18 % | 25 % | 7 % |
| **HAL** | 40 % | 50 % | 10 % | 41 % | 50 % | 9 % |

**Tabel 4: De verschillen in cpu load voor scenario 1 en 3**

Wat het meest opvalt is dat SenseGraphics H3D met meer objecten in de scene aanzienlijk meer load geeft indien de haptische manipulator contact maakt. Het verschil tussen contact of niet bedraagt 3 % voor senario 3, wat toch wel een opvallende toename is. Het andere grote punt was dat HAL deze test niet meer haptisch kon renderen. Op [Figuur 62] wordt de memory footprint van scenario 3 getoond.



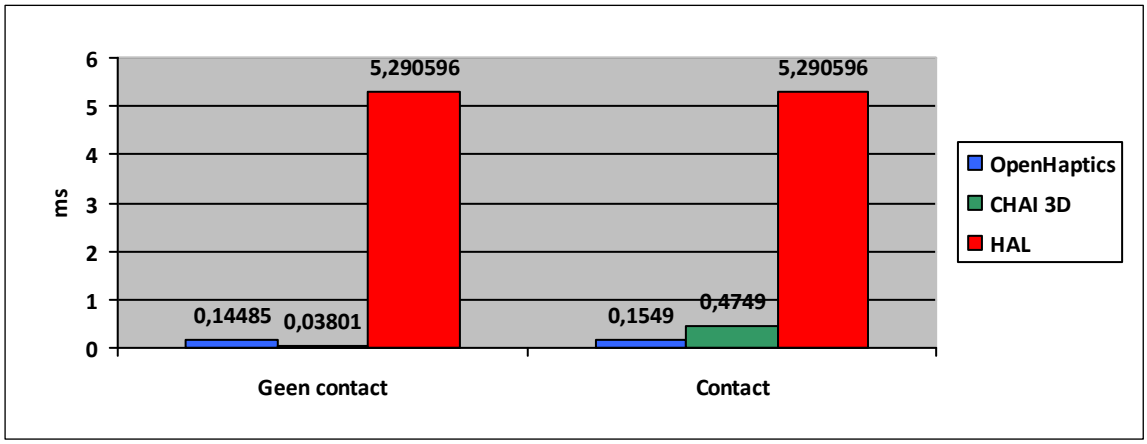

**Figuur 62: De memory footprint van de verschillende API's in scenario 3**

We zien dat voor CHAI 3D en HAL de memory footprint vrijwel constant blijft. Maar bij SensAble OpenHaptics en SenseGraphics H3D zien we een kleine toename wanneer er contact wordt gemaakt. De reden dat deze twee API's deze toename vertonen is wellicht te verklaren door het feit dat SenseGraphics H3D gebruik maakt van SensAble OpenHaptics voor zijn haptics. Op [Tabel 5] zien we de verschillen tussen scenario 1 en scenario 3.

| | **Scenario 1 – geen contact** | **Scenario 3 – geen contact** | **Verschil** | **Scenario 1 - contact** | **Scenario 3 - contact** | **Verschil** |
|---|---|---|---|---|---|---|
| **SensAble OpenHaptics** | 20,604 Mb | 22,220 Mb | 1,616 Mb | 20,672 Mb | 22,312 Mb | 1,640 Mb |
| **CHAI 3D** | 15,084 Mb | 15,640 Mb | 0,556 Mb | 15,084 Mb | 15,640 Mb | 0,556 Mb |
| **SenseGraphics H3D** | 23,84 Mb | 25,696 Mb | 1,856 Mb | 23,84 Mb | 25, 840 Mb | 2,000 Mb |
| **HAL** | 21,532 Mb | 22,628 Mb | 1,096 Mb | 21,54 Mb | 22,628 Mb | 1,088 Mb |

**Tabel 5: De verschillen in memory footprint voor scenario 1 en 3**

In [Tabel 5] zien we dat vrijwel iedere API meer geheugen in gebruik zal nemen naarmate het aantal objecten toeneemt, wat vrij normaal is. Opvallend is dat SenseGraphics H3D de grootste toename van geheugen neertekent. Op [Figuur 63] wordt de timing van de haptische loop in kaart gebracht.




**Figuur 63: Het timen van de haptische loop in scenario 3**

Op [Figuur 63] zien we duidelijk waarom HAL zijn haptics heeft uitgeschakeld. HAL heeft namelijk meer dan 5 milliseconden nodig om zijn haptische loop uit te voeren, ver boven het maximum van 1 milliseconde om stabiel krachten aan 1 Khz te genereren. We zien ook dat SensAble OpenHaptics en CHAI 3D wel voldoen aan deze maximumtijd, maar we zien ook dat CHAI 3D opvallend veel tijd nodig heeft voor het uitvoeren van de haptische loop indien er contact is. De verschillen ten opzichte van scenario 1 zijn opgesomd in [Tabel 6].

| | **Scenario 1 – geen contact** | **Scenario 3 – geen contact** | **Verschil** | **Scenario 1 - contact** | **Scenario 3 - contact** | **Verschil** |
|---|---|---|---|---|---|---|
| **SensAble OpenHaptics** | 0,188 ms | 0,14485 ms | -0,04315 ms | 0,222 ms | 0,1549 ms | -0,0671 ms |
| **CHAI 3D** | 0,003015 ms | 0,03801 ms | 0,034995 ms | 0,008696 ms | 0,4749 ms | 0,466204 ms |
| **HAL** | 0,086762 ms | 5,290596 ms | 5,203834 ms | 0,113291 ms | 5,290596 ms | 5,177305 ms |

**Tabel 6: De verschillen in duur van de haptische loop voor scenario 1 en 3**

In [Tabel 6] zien we duidelijk hoe de haptische loop van SensAble OpenHaptics vrij stabiel blijft bij de verschillende scenario's. Als we verder zouden gaan met objecten toevoegen, dan zal deze API het waarschijnlijk het langst uithouden. CHAI 3D toont in geval van geen contact goede prestaties, maar in geval van contact is er duidelijk te zien hoe de API meer tijd nodig heeft om alles te verwerken. Na verdere analyse bleek overigens dat CHAI 3D vanaf ongeveer 450 kubussen onstabiel werd, maar dat in tegenstelling tot HAL de haptics ingeschakeld bleven. Ook is duidelijk te zien hoe HAL niet meer voldoet aan de voorwaarden om stabiel krachten te renderen, en hoe deze API als gevolg zijn haptics uitschakelt.

***Conclusie***

Na het uitvoeren van deze benchmarks zien we dat de API met de beste algemene prestaties CHAI 3D is, CHAI 3D presteerde in vrijwel alle scenario's het beste. Maar, we dienen op te merken dat SensAble Openhaptics – ondanks het ietwat gulzige omspringen met resources - de beste prestatie biedt qua pure haptics. Zo was er te zien dat de haptische loop van deze API over alle scenario's zo goed als stabiel bleef. CHAI 3D is spaarzamer, maar zal naarmate de scene complexer wordt, meer moeite hebben om alles gerenderd te krijgen. De test is weliswaar moeilijk te vergelijken, aangezien er bij SensAble OpenHaptics nog zelf voorzien zal moeten worden in het grafisch renderen. De stabielere performance van SensAble OpenHaptics kunnen we verklaren door betere algoritmen zoals beschreven in hoofdstuk 4.
Een andere opmerking is dat SenseGraphics H3D vrij veel resources inneemt, dit is te verklaren doordat deze toolkit eerst de applicatie van H3DLoad moet starten, waardoor er veel meer moet worden ingeladen ten opzichte van de andere API's. Zo bleef bij H3D bijvoorbeeld de navigatie altijd mogelijk, terwijl dit was geschrapt voor de andere demo's. Een laatste opmerking die we moeten maken is dat de API's zonder eigen grafische rendering veel meer cpu cycles vroegen, dit wellicht door onze eigen grafische code. Hier moesten we een afweging maken, of we gebruikten onze eigen grafische code, die wellicht minder efficiënt is, of we gebruikten geen grafische code, maar dan kregen we natuurlijk ook geen correct beeld. Om hier een zo goed mogelijk beeld van te kunnen maken werd dus de haptische loop gemeten.

## *Conclusie*

Door het maken van een implementatie in iedere toolkit hebben we een veel beter beeld gekregen van wat er momenteel mogelijk is op het vlak van haptics. We hebben voor de implementatie de toolkits gebruikt die het bekendst zijn. We hebben bij geen enkele toolkit grote problemen ondervonden, maar het is opvallend hoe uiteenlopend de visie per toolkit is. In een presentatie van CHAI 3D staan de verschillende haptische API's op dit moment beschreven als de toren van Babel. Na het maken van de demo's kunnen we dit inderdaad beamen. De vier API's die we hebben gebruikt zijn alle vier weliswaar goed in hun eigen opzet, maar iedere API biedt voor- en nadelen. Er is geen enkele API die goed is voor eender welke situatie. Iedere toolkit heeft zijn eigen specifieke doelpubliek en opzet. Zo is SensAble OpenHaptics een goede toolkit, die voorzien is van veel features, maar ze is bedoeld om samen te werken met OpenGL, wat zowel een plus- als een minpunt kan zijn. Om met OpenHaptics zonder OpenGL te werken moet er gebruik worden gemaakt van HDAPI, maar deze biedt dan weer niet de

features aan van de andere toolkits. Anderzijds is OpenHaptics gebonden aan SensAble, met als gevolg dat enkel de haptische apparaten van SensAble ondersteund worden. CHAI 3D biedt een alternatief door eender welk apparaat toe te laten, maar tegelijk legt CHAI 3D restricties op door gebruik te maken van zijn eigen grafische rendering. Hoewel dit praktisch is om snel te ontwikkelen, en het de prestaties zeker ten goede kwam, kan er niet zomaar gekozen worden voor een andere grafische toolkit. Net als bij OpenHaptics kan er ook hier direct met het apparaat gecommuniceerd worden om zo zelf grafisch te kunnen renderen, maar dan is er geen ondersteuning meer voor de speciale features. SenseGraphics H3D gebruikt een totaal andere opzet door gebruik te maken van X3D en pythonscript. Zo kan er nog sneller een werkende applicatie gemaakt worden en de programmeur heeft zelfs geen echte programmeerkennis nodig. Anderzijds moet hij of zij dan wel goed overweg kunnen met X3D en pythonscript om een gelijkwaardig alternatief te bekomen. H3D biedt ook nog altijd de mogelijkheid om te ontwikkelen in C++, maar dit echter enkel indien X3D en pythonscript niet voldoende zijn. Een pluspunt van H3D is dat - hoewel deze toolkit steunt op OpenHaptics - het toch alternatieve haptische apparaten kan gebruiken. Tenslotte is er HAL, een API die enerzijds tracht het beste van de haptische rendering te nemen, en anderzijds de gebruiker de volledige vrijheid geeft bij het grafisch renderen. Welke van deze API’s de beste is, is dan ook volledig afhankelijk van de eisen van het haptische project.

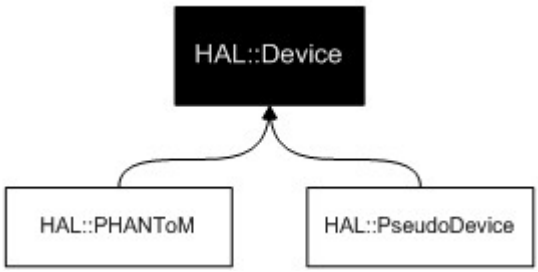
## Hoofdstuk 8

# Uitbreiding HAL

In het laatste onderdeel van deze thesis zullen we een minpunt van HAL selecteren en trachten weg te werken. Daar we in hoofdstuk 7 HAL hebben opgenomen in onze demo's, kunnen we nu een selectie maken met de minpunten van de HAL toolkit. Zoals we eerder zagen is HAL een van de API's zonder grafische rendering. Het ontbreken van deze rendering zien we echter niet als een groot minpunt. Het feit dat HAL puur instaat voor haptische rendering en dit doet via een scenegraph, zorgt er immers voor dat HAL in combinatie met eender welke andere grafische API gebruikt kan worden. Het grootste nadeel van HAL is volgens ons dat de library momenteel gebonden is aan de haptische apparaten van SensAble. We hebben daarom besloten om dit minpunt weg te werken.

### *HAL extentie 1: Het elimineren van de gebondenheid aan enkel de haptische apparaten van SensAble*

Om dit minpunt weg te kunnen werken, moeten we eerst een beeld scheppen van de communicatie van HAL met de haptische hardware. Op [Figuur 64] tonen we een onderdeel van het klassediagram waar we zien hoe momenteel de haptische hardware geïmplementeerd is.



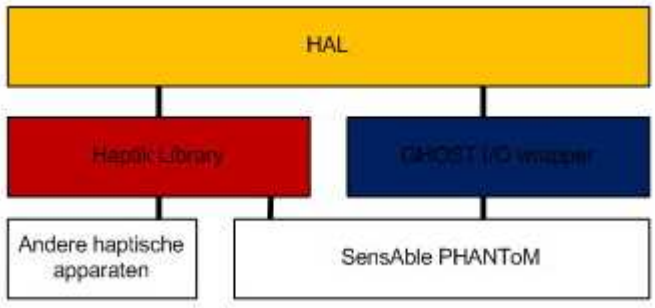

**Figuur 64: De HAL Device klasse en zijn kinderen**

We zien dat HAL werkt met de klasse Device voor de haptische hardware. Dit kan een pseudodevice zijn, wat wil zeggen dat er eigenlijk geen haptisch apparaat is en dat er dus met het toetsenbord gewerkt zal worden. Anderzijds zal dit meestal een object van het type PHANToM zijn, dat de haptische hardware van SensAble zal aanspreken. De PHANToM klasse werkt momenteel met de GHOST I/O

wrapper om te communiceren met de hardware. De GHOST I/O wrapper is een onderdeel van de GHOST SDK en zoals we zagen in in hoofdstuk 5 kan deze enkel werken met de apparaten van SensAble. Deze beperking gaan we dus elimeneren, om zo toe te laten dat er met andere apparatuur dan die van SensAble gewerkt kan worden. Om dit te doen zullen we een nieuwe klasse maken die overerft van HAL::Device. De oorspronkelijke PHANToM klasse heeft in wezen 5 functies:

- Constructor: Deze method zal het haptisch device initialiseren.
- Destructor: Deze method zal het haptisch device stoppen en de resources releasen.
- Start(): Deze method zal de haptische loop van het apparaat starten.
- Stop(): Deze method zal de haptische loop stoppen.
- Loop(): Deze method zal in de haptische loop worden aangeroepen voor het verkrijgen van de huidige positie van het haptische apparaat en eveneens zal hier de toe te passen kracht worden doorgegeven naar het apparaat.

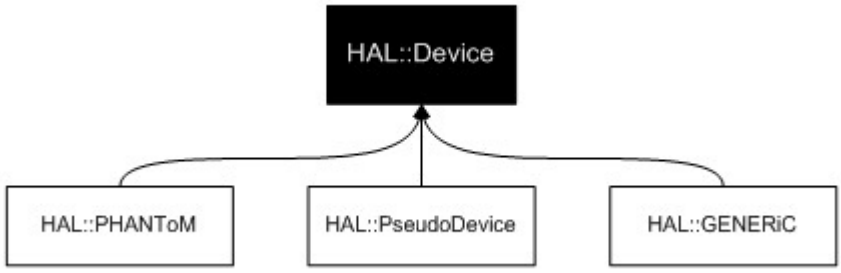
In ieder van deze methods wordt er met de GHOST I/O wrapper gewerkt. We dienen hier nu een alternatief voor te vinden. In hoofdstuk 5 hebben we een kort overzicht gegeven van de Haptik Library, een componentgebaseerde architectuur voor uniforme toegang tot haptische devices. De Haptik Library biedt een laag aan die tussen de hardware en de applicatielaag van HAL dient te komen. Deze laag laat het toe om met dezelfde broncode zowat ieder op dit moment beschikbare haptisch apparaat te gebruiken. Indien er ooit een nieuw apparaat op de markt zou komen, dan kan de besturing voor dit apparaat ook snel toegevoegd worden aan de Haptik Library. Het laagmodel van een applicatie die gebruik maakt van HAL zal er door gebruik te maken van de Haptik Library uitzien zoals aangegeven op [Figuur 65].




**Figuur 65: Het laagmodel na de Haptik Library integratie**

De nieuwe klasse die we hebben gemaakt noemen we HAL::GENERiC, deze klasse maakt geen gebruik meer van de GHOST I/O wrapper en zal deze dus vervangen door de Haptik Library. Alle methods zijn aangepast om dit te doen. Er is ook in de Start() method een callback gezet van de haptische loop van de Haptik Library naar de Loop() method van de HAL::GENERiC klasse. In de Loop()

method zullen we de data van het haptisch apparaat uitlezen en de nieuwe krachten toepassen. We hebben hier ook code moeten schrappen in vergelijking met de originele HAL::PHANToM klasse. Zo biedt de Haptik Library in tegenstelling tot de GHOST I/O wrapper geen functionaliteiten voor het waarnemen van de temperatuur van het haptisch apparaat. Met de nieuwe device klasse kunnen we het klassediagram van [Figuur 64] aanpassen tot het klassediagram op [Figuur 66].




**Figuur 66: De nieuwe versie van het klassediagram**

Het aanspreken van de Haptik Library verliep zonder problemen, enkel dat het assenstelsel van deze library iets verschilt van deze van HAL. We moesten namelijk de Z-as inverteren om een correcte mapping te verkrijgen. Om een applicatie gebruik te laten maken van de nieuwe HAL::GENERiC klasse, dient er aan deze applicatie nog wel een configuratiebestand te worden toegevoegd. Dit bestand “Haptik.config.txt” geeft aan welke haptische apparaten er mogelijk zijn, alsook hun specifieke configuraties. De inhoud van een dergelijk bestand is te vinden in [Bijlage 7]. Tenslotte is er nog een verschil tussen het uitvoeren van een applicatie die gebruik maakt van de HAL::GENERiC klasse in vergelijking met wanneer er met de HAL::PHANToM klasse gewerkt wordt. Bij het starten van de applicatie zal deze namelijk eerst één maal moeten gecalibreerd worden, daarna zal alles naar behoren werken. De calibratie wordt nog getoond op [Figuur 67].

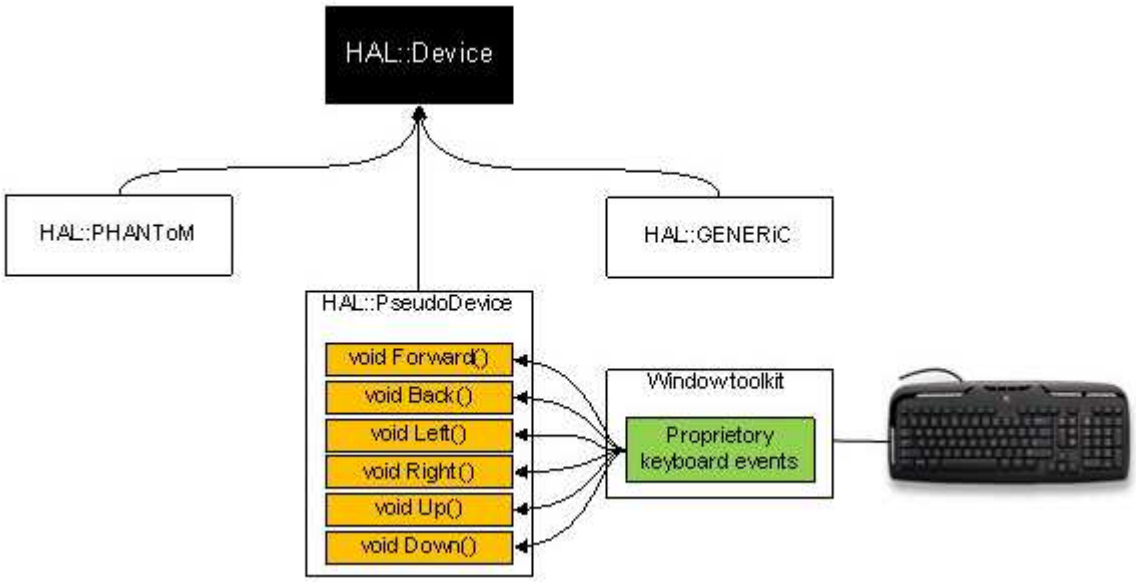
**Figuur 67: De calibratie van de Haptik Library**

### Conclusie

Het resultaat van deze uitbreiding is dat HAL nu ieder commercieel te verkrijgen apparaat ondersteunt. Als er ooit nog een nieuw apparaat op de markt komt, dan kan dit nadien altijd worden toegevoegd zonder dat de code van de applicatie zelf moet wijzigen. De Haptik Library community is hier vrij actief mee bezig. Eveneens blijkt dat volgens [49] het toevoegen van deze laag vrijwel geen vertraging zal teweegbrengen ten opzichte van het directe gebruik van de GHOST I/O wrapper. Met deze implementatie is HAL de enige haptische toolkit die volledige vrijheid biedt op het vlak van grafische rendering en die tegelijk ook alle haptische apparaten ondersteund, terwijl het gebruik maakt van een haptische scenegraph voor het renderen van de haptische scene.

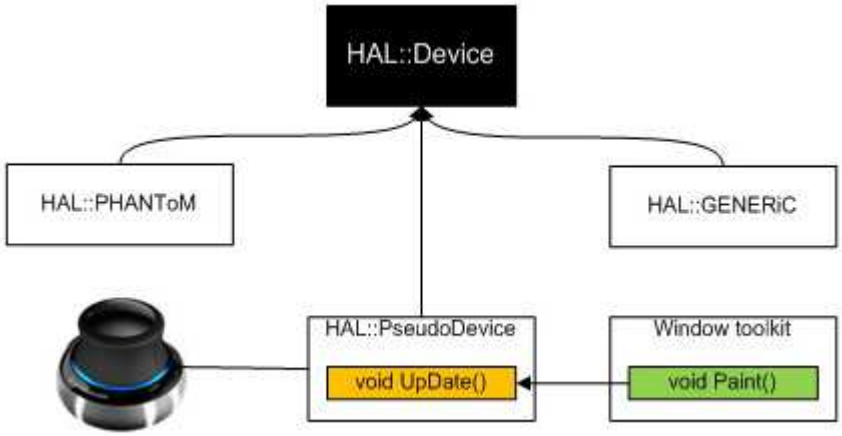
## *HAL extentie 2: Het uitbreiden van het pseudodevice met een spacemouse*

Een andere uitbreiding die ons handig leek was het uitbreiden van het pseudodevice. Het huidige pseudodevice zorgt voor een alternatief indien er geen haptisch apparaat ter beschikking staat. Dit pseudodevice zal dan de inputacties die de gebruiker normaal doet met het haptische apparaat vervangen door de inputacties van het toetsenbord. Deze opzet is echter volledig afhankelijk van de gebruikte window toolkit. De programmeur moet immers met deze toolkit de keyboardevents zelf opvangen en ze doorgeven aan de gepaste method van het pseudodevice. Een schematisch voorbeeld van deze opzet is te zien op [Figuur 68].




**Figuur 68: Een schematisch voorbeeld van de werking van het pseudodevice door middel van een keyboard**

Het nadeel aan deze opzet is dat er alleen gebruik van het keyboard kan worden gemaakt en dat de koppeling hiermee afhankelijk is van de gebruikte window toolkit. De door ons gemaakte extentie implementeert een spacemouse die automatisch de koppeling met de positiemethods zal maken. Voor onze implementatie werd er gebruik gemaakt van een spacemuse van 3D Connexion: de “3D Connexion SpaceNavigator SE”. Om dit te realiseren hebben we gebruik gemaakt van de 3D Connexion SDK 2.0.4, die ervoor zorgt dat alle modellen van deze fabrikant gebruikt kunnen worden. Indien er een spacemouse aanwezig is zal ze geïnitialiseerd worden en kan ze gebruikt worden om de virtuele manipulator mee te positioneren. Indien er geen zulk type apparaat aanwezig is kan er nog steeds met de standaard keyboardinterface gewerkt worden. De door ons gemaakte implementatie werkt door het apparaat te pollen. Deze polling moet wel zelf gelinkt worden door de UpDate-method van het pseudodevice aan te roepen. Dit is bijvoorbeeld mogelijk door deze aan te roepen in de Paint-call. Een voorbeeldschema van huidige implementatie is te zien op [Figuur 69].




**Figuur 69: Een schematisch voorbeeld van de werking van het pseudodevice door middel van een spacemouse**

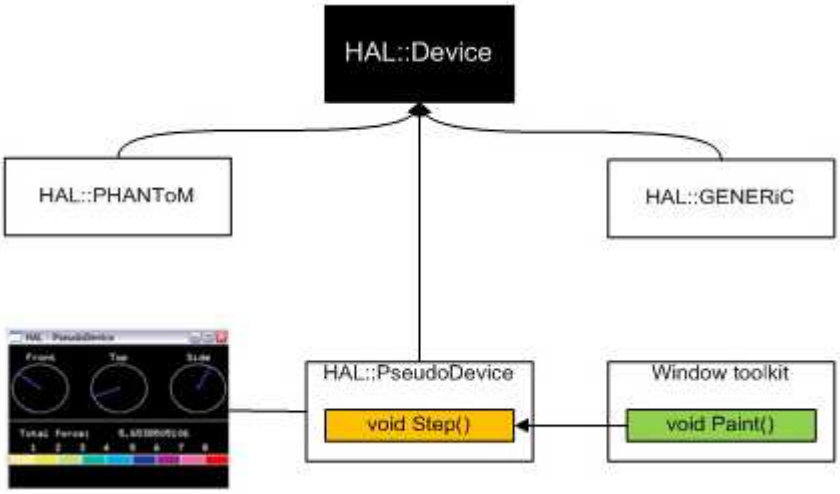
We dienen wel op te merken dat de window toolkit de windowhandle dient door te geven in de constructor van het pseudodevice om met de spacemouse te kunnen werken. Indien dit niet wordt gedaan kan er enkel met het keyboard gewerkt worden.

### Conclusie

Deze uitbreiding zorgt voor een meer algemenere toepassing van het pseudodevice. Zo kunnen er nu apparaten worden aangesloten die qua interface meer geschikt zijn voor 3D-navigatie. Eveneens is de toegang tot dit apparaat makkelijker te implementeren, aangezien er minder methods gelinkt dienen te worden aan specifieke methods van de gebruikte window toolkit.

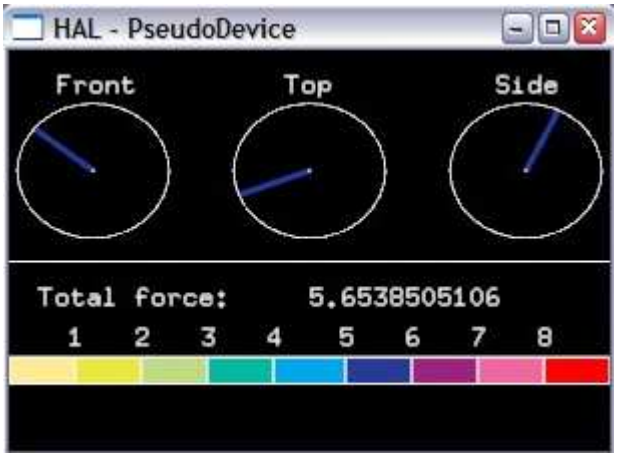
## *HAL extentie 3: Het uitbreiden van het pseudodevice met een grafisch front-end*

De derde en laatste uitbreiding die er voor HAL werd gemaakt is een tweede uitbreiding op het pseudodevice. Daar het de opzet van het pseudodevice is om een haptisch apparaat te simuleren, werd er hier een grafisch front-end voor gemaakt; een pseudodevice beschikt namelijk niet over de mogelijkheid om haptics fysiek te renderen. De bedoeling van het grafisch front-end is om de gebruiker toch te voorzien van enige feedback over de haptische krachten die zijn acties teweeg zouden brengen. De extentie zal een apart venster openen met daarin grafische feedback over de gerenderde krachten. Het venster wordt gemaakt door middel van MFC, terwijl de haptische info gerenderd wordt via de OpenGL API. Een schematisch voorbeeld van de opzet van deze extentie is te zien op [Figuur 70].



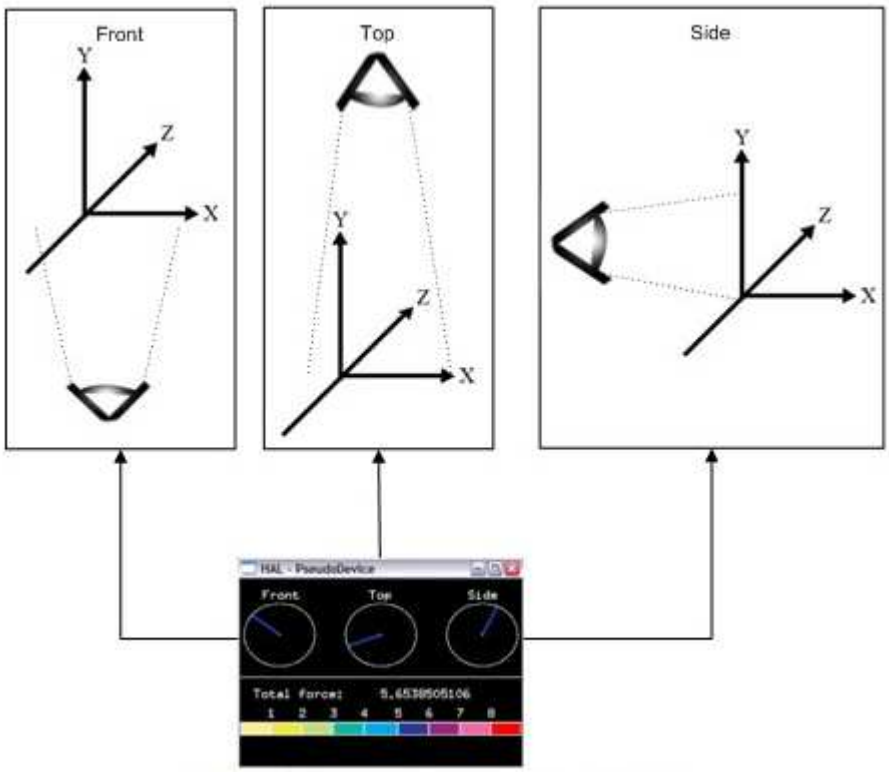

**Figuur 70: Een schematisch voorbeeld van de werking van het pseudodevice met het grafisch front-end**

Zoals te zien op [Figuur 70] dient de gebruiker het grafisch front-end zelf te updaten. Deze update kan bijvoorbeeld aangeroepen worden met de paintCall van de hoofdapplicatie. Het aparte venster dat gevisualiseerd wordt voor de gebruiker is te zien op [Figuur 71].




**Figuur 71: Het grafisch front-end voor het pseudodevice**

Op [Figuur 71] is te zien dat de krachtvector die normaal door middel van het haptische apparaat haptics voorziet, nu gevisualiseerd wordt via drie wijzerplaten. De eerste wijzerplaat zal de krachtvector visualiseren in het XY-vlak, de tweede wijzerplaat doet de visualisatie in het XZ-vlak en de derde wijzerplaat zal het ZY-vlak visualiseren. [Figuur 72] zorgt voor een duidelijker beeld van hoe dit geïnterpreteerd moet worden.




**Figuur 72: De visualisatie van de krachtvector**

Ook zal de krachtvector tekstueel worden weergeven, en tevens zal de totale kracht van deze vector geaccentueerd worden met behulp van kleurcodes.

## Conclusie

Het grafisch front-end voor het pseudodevice zorgt voor extra informatie om zo zonder haptisch apparaat toch tests te kunnen uitvoeren. De gebruiker kan nu zonder extra code dadelijk de gerenderde krachten zien; zo wordt er zonder eigenlijke haptics een zo goed mogelijk vervangmiddel aangeboden.

## Hoofdstuk 9

# Conclusie

### *Algemene conclusie*

Haptics zijn voor de gewone gebruiker vrij onbekend, maar indien er even bij wordt stilgestaan, valt er te zien dat haptics reeds alomtegenwoordig zijn en dat de ontwikkeling ervan niet stilstaat. Zo zijn haptics reeds lang in gebruik in verschillende vakgebieden. We kunnen zien hoe haptics gebruikt wordt in tal van simulatoren, hoe haptics een revolutie ontketend heeft in de medische wereld, en zelfs hoe haptics de mens toelaat nieuwe nanoscopische werelden haptisch te verkennen. Niet alleen in de nieuwe technologieën zien we de opkomst van haptics, maar ook in de bedrijfswereld wordt het steeds meer aangewend. Zo is er te zien hoe er een meerwaarde kan gegeven worden bij 3D modellering en hoe er in trainingsdoeleinden gebruik van gemaakt kan worden om zo kosten te besparen, maar toch tegelijk de veiligheid er mee sterk te vergroten. Ook is er te zien hoe haptics steeds meer en meer bij de gewone mens terecht komt, zo maakt tegenwoordig iedere recente spelconsole gebruik van controllers voorzien van force feedback.

De sterke opkomst van haptics heeft ertoe geleid dat er reeds tal van bedrijven actief mee bezig zijn. De meest bekende bedrijven zijn SensAble en Immersion Corporation. Beide bedrijven hebben van haptics hun core business gemaakt. Door deze groei in interesse in haptics zien we steeds meer spelers op de markt. Dit resulteert onder andere in het feit dat ook de haptische apparatuur steeds meer in het bereik komt van de gewone mens. Zo heeft SensAble sinds kort een nieuw goedkoper haptisch apparaat op de markt gebracht dat aan te sluiten is via een firewire-connectie: de PHANToM Omni. Maar ook andere bedrijven komen met betaalbare toestellen: zo heeft Novint net zijn Falcon uitgebracht, het eerste haptische apparaat met een verkoopprijs van minder dan 150 €.

Deze toestroom van fabrikanten heeft ertoe geleid dat ook de software voor haptics in een stroomversnelling zit. Iedere fabrikant brengt zijn eigen devices uit met de daarbij horende interface en Software Development Kit. Het grote probleem met deze zogenaamde haptische API’s is dat iedere fabrikant werkt met proprietary software en drivers. Om een applicatie te ontwikkelen met haptics, moet er dus een keuze gemaakt worden voor welk haptische apparaat dat er gebruikt zal worden, komt er nadien een ander, beter apparaat op de markt, dan

dient de applicatie voor dit device hermaakt te worden. Momenteel is er dus een toevloed aan verschillende haptische API's met ieder hun specifiek ondersteunde hardware. Momenteel is er hier wel een positieve ontwikkeling om deze hardwareondersteuning zo uniform mogelijk te maken, het grootste voorbeeld hiervan is de Haptik Library.

In deze thesis werd eerst besproken wat het nut is van haptics en op welke markten er gebruik van wordt gemaakt, met de bijhorende voordelen. Vervolgens werd er in hoofdstuk 4 een overzicht gegeven van hoe de software voor deze API's te werk moet gaan. Haptics zijn namelijk zeer cpu-intensief en er dienen dus speciaal ontworpen algoritmen voor ontwikkeld te worden. Door de niet uniforme software voor haptics dienen veel van deze algoritmen verschillende keren apart voor verschillende toolkits te worden geïmplementeerd. In hoofdstuk 5 geven we dan ook een overzicht van de verschillende haptische API's die er beschikbaar zijn, waarna we in hoofdstuk 6 de verschillende soorten hardware ervoor bespreken.

Uiteindelijk was het de bedoeling verschillende API's met mekaar te gaan vergelijken, niet alleen theoretisch, maar ook door in ieder van deze API's een demo te maken om zo hun features te kunnen toetsen. In hoofdstuk 7 hebben we op die manier 4 API's met mekaar vergeleken. Eerst werd er een demo gemaakt om zo hun werking en features te achterhalen, waarna we de demo's later meer in detail zijn gaan vergelijken. Ook werden de demo's vergeleken door ze met elkaar te benchmarken. Na hoofdstuk 7 kunnen we besluiten dat al de API's hun voor- en nadelen hebben. Sommige API's zoals SenseGraphics H3D en CHAI 3D trachten een totaaloplossing te bieden, maar verliezen hierdoor een groot deel van hun flexibiliteit. Andere API's zoals SensAble OpenHaptics en HAL richten zich in tegenstelling tot de anderen puur op het haptisch gebeuren. De conclusie is dan ook dat de keuze voor welke API te gebruiken volledig afhankelijk is van de volgende factoren:

- De programmeertaal waarmee er gewerkt zal worden.
- Het haptische apparaat dat ervoor gebruikt zal worden.
- De persoonlijke voorkeuren en de ervaring van de programmeurs.
- De vraag of er een integratie in een reeds bestaande applicatie dient te gebeuren of is er een volledig nieuwe applicatie nodig.

Dit zijn de belangrijkste vragen wanneer er een applicatie zal ontwikkeld worden die gebruik maakt van haptics. Afhankelijk van deze vragen kan er een haptische toolkit geselecteerd worden. We kunnen hier de ultieme conclusie trekken dat er geen uniform aan te raden haptische API bestaat.

Tenslotte werden er in hoofdstuk 8 uitbreidingen gemaakt voor HAL. Na hoofdstuk 7 hadden we een goed beeld over de grootste tekortkomingen van HAL en in

hoofdstuk 8 wordt het grootste pijnpunt hiervan opgelost. Het integreren van de Haptik Library voor een uniforme toegang tot eender welk haptisch apparaat zorgt er immers voor dat de toolkit altijd goed gebruikt kan worden en ook future-proof is als er nieuwe apparaten op de markt komen. Een tweede uitbreiding zorgt ervoor dat het pseudodevice ook gebruikt kan worden met een spacemouse. Indien er dus geen beschikking bestaat over een haptisch apparaat kan er nu gebruik worden gemaakt van een spacemouse of een toetsenbord. De derde uitbreiding werkt het pseudodevice nog verder uit, door ervoor te zorgen dat de gebruiker beschikt over een visuele feedback van de krachtvector.

We kunnen afsluiten door te zeggen dat de ontwikkelingen in de haptische wereld volop bezig zijn en dat het zeer interessant wordt om te zien hoe de haptische API's in de toekomst zullen evolueren, zeker nu er meer en meer goedkopere apparaten op de markt komen.

## *Future work*

In deze thesis kunnen verschillende aspecten uitgebreid worden.

Zo kunnen de benchmarks van de verschillende API's uitgebreid worden. Momenteel werden deze op één enkel systeem getest, maar er zijn verschillende scenario's mogelijk, voor deze thesis beschikten we toch wel over een zeer krachtig systeem. De vraag is natuurlijk hoe al deze API's presteren op een normale desktopcomputer voor consumenten. Zo kunnen we de benchmarks uitbreiden waarin we de verschillende demo's testen op verschillende pc's. Eveneens kan er hier verder worden gekeken naar het effect van meerdere processoren. Het is tegenwoordig de trend van de chipbakkers om meer cores op één enkele cpu te zetten. Met deze benchmark zou het mogelijk zijn om te kijken welke van de API's hier het beste op schaleert.

Natuurlijk kunnen er ook meerdere API's getest worden; er zijn nog verschillende API's die gratis beschikbaar zijn, maar er zijn er ook die te betalen zijn. De vraag hier zou zijn of de betaalde API's een grote meerwaarde bieden ten opzichte van de reeds gratis verkrijgbare alternatieven, dit niet alleen qua support, maar ook qua gebruiksgemak en performantie.

Voorts zou het ook mogelijk zijn om een vergelijking te maken van de verschillende haptische apparatuur zelf. Er zijn momenteel meerdere fabrikanten op deze markt actief, en de prijs van de apparaten varieert sterk. Het lijkt ons vooral interessant om de goedkopere apparaten te testen, zoals bijvoorbeeld de recent uitgebrachte Novint Falcon en de PHANToM Omni.

Over het uitbreiden van de HAL library moet er goed worden nagedacht, zo was er bijvoorbeeld het idee om HAL te koppelen aan een grafische rendering. Het probleem hierbij is dat HAL dan een van zijn grootste troeven zou verliezen - de volledige vrijheid om te kiezen waarmee de library in combinatie mee gebruikt kan worden. Weliswaar zijn er andere uitbreidingen mogelijk, zoals het toevoegen van meerdere primitieven of van speciale voorgedefinieerde haptische effecten. Tenslotte zou een website met extra voorbeelden, een forum en een programmer's guide het gebruik van de library flink kunnen stimuleren.

# Bibliografie


1. Brian T. Bethea, Allison L. Okamura, Masaya Kitagawa, Torin P. Fitton, Stephen M. Cattaneo, Vincent L. Gott, William A. Baumgartner en David D. Yuh. Technical Report: Application of haptic feedback to robotic surgery. Journal of Laparoendoscopic & advanced surgical techniques, Volume 14, Number 3, 2004, pagina 192 – 193.

2. J. Allard, S. Cotin, F. Faure, P.-J. Bensoussan, F. Foyer, C. Duriez, H. Delingette, L. Grisoni. SOFA – an Open Source Framework for Medical Simulation. CIMIT Sim Group – Harvard Medical School, INRIA – Evasion, Alcove, and Asclepios teams, juni 2006.

3. Kevin Montgomery, Eric Herbranson, Paul Brown. A haptically Enabled Dental Simulator. National Biocomputation Center, Stanford University, Stanford CA – Brown and Herbranson Imaging, Portola Valley, CA, april 2005.

4. Michael K. Zyskowski. Aircraft Simulation Techniques Used in Low-Cost, Commercial Software. Microsoft Corporation Redmond WA. AIAA Modeling and Simulation Technologies Conference and Exhibit, 11-14 augustus 2003, Austin, Texas.

5. John D. Ianni, Daniel Repperger, Robert W. Baker, Robert L. Williams. Human Interfaces For Robotic Satellite Servicing. Air Force Research Laboratory, Ohio University, juni 2002.

6. Antoine Ferreira, Member IEEE en Constantinos Mavroidis, Member IEEE. Virtual Reality and Haptics for Nano Robotics: A Review Study. IEEE Robotics and Automation Magazine, 2006.

7. Sam-ha Choi, he-Dong Chang, Kyung-Sik Kim. Development of Force-Feedback Device for PC-Game using vibration. Department of Computer Engineering, Hoseo University, 2004.

8. Current and Next Generation Game Console Feature Study. Ipsos-Insight. 22 september, 2006.

9. Qi Wang, Vincent Levesque, Jerome Pasquero, Vincent Hayward. A haptic Memory Game using the STRESS Tactile Display. Haptics Laboratory McGill University, Canada, april 2006.

10. Dan Morris, Neel Joshi, Kenneth Salisbury. Haptic Battle Pong: High-Degree-of-Freedom Haptics in a Multiplayer Gaming Environment. Computer Science Department, Stafford University en University of California, San Diego.

11. Margaret L. McLaughlin, Joao Hespanha en Gauray S. Sukhatme. Touch in Virtual Environments: Haptics and the design of interactive Systems, hoofdstuk 3 en 4, december 2001.

12. Chrisopher M. Smith. Human Factors in Haptic Interfaces, Crossroads, volume 3, issue 3, Special issue on human computer interaction, pagina 14 – 16, 1997.

13. Stéphane Redon, Abderrahmane Kheddar en Sabine Coquillart. Fast Continuous Collision Detection between Rigid Bodies. I3D – INRIA – France, CEMIF-SC – Iniversité d'Evry – France, 2002.

14. Daniela Constantinescu, Septimiu E. Salcudean, Elizabeth A. Croft. Haptic rendering of rigid contacts using impulsive and penalty forces. IEEE Transactions on Robotics, volume 21, issue 3, pagina 309 – 323, juni 2005.

15. Miguel A. Otaduy en Ming C. Lin. Introduction to Haptic Rendering. Department of Computer Science. University of North Carolina at Chapel Hill, 2005.

16. S. Gottschalk, M. C. Lin en D. Manocha. OBBTree: A Hierarchical Structure for Rapid Interference Detection. Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599-3175, pagina 171 – 180, 1996.

17. Wilson, E. Larsen, D. Manocha en M. C. Lin. Partitioning and Handling Massive Models for Interactive Collision Detection. Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599-3175, september 1999.

18. Avneesh Sud, Naga Govindaraju, Russel Gayle, Ilknur Kabul en Dinesh Manocha. Fast Proximity Computation among Deformable Models Using Discrete Voronoi Diagrams. Dept of Computer Science, University of North Carolina at Chapel Hill, pagina 1144 – 1153, 2006..

19. Karljohan Lundin. Natural Haptic Feedback from Volumetric Density Data. Department of Science and Technology, Linköping Universiy, SE-601 74 Norrköping, Sweden, december 2001.

20. Minsky M. Computational Haptics: The Sandpaper System for synthesizing Texture for a Force-Feedback Display. Media Arts and Sciences, MIT, 1995.

21. M. Anitescu en G. Hart. A Hard-Constraint Time-Stepping Approach for Rigid Multibody Dynamics with Joints, Contact and Friction. Proc Conf Divers Comput, Atlanta, GA, pagina 34 – 41, 2003.

22. Diego C. Ruspini, Krasimir Kolarov en Oussama Khatib. The haptic display of Complex Graphical Environments. Stanford University en Interval Research Corporation, Proceedings of the 24th annual conference on Computer graphics and interactive techniques, pagina 345 – 352, 1997.

23. Herman J. Haverkort. Results on geometric networks and data structures, Phd thesis. Utrecht University, hoofdstuk 1, pagina 14, 2004.

24. Cagatay Basdogan, Chih-Hao Ho. Haptic Rendering in Virtual Environments. Laboratory for human and Machine Haptics, MIT. Virtual Environments HandBook, hoofdstuk 4, 2001.

25. Miguel A. Otaduy. Sensation Preserving Simplification for 6-DoF Haptic Display. ETH-Zurich. Siggraph 2005.

26. Rajshree Chabukswar, Adam T. Lake, Mary R. Lee. Multi-threaded Rendering and Physics Simulation. Intel Software Solutions Groups. februari 2005.

27. M. de Pascale, G. de Pascale, D. Prattichizzo, F. Barbagli. A GPU-friendly Method for Haptic and Graphic Rendering of Deformable Objects. Robotics & Systems Lab, University of Siena, Italy, Eurohaptics 2004.

28. H3D API Manual, Blz 12.
http://www.h3dapi.org/uploads/api/H3D%20API%20Manual.pdf
17 augustus 2007.

29. David Arendash. The Unreal Editor as a Web 3D Authoring Environment. Quantum Leap Computing, Web3d, pagina 119 – 126, 2004.

30. Henry König, Thomas Strothotte. Fast Collision Detection for Haptic Displays Using Polygonal Models. Otto-von-Guericke University Magdeburg. Proceedings of the Conference on Simulation and Visualizaton, SCS—Europe BVBA, Ghent, pages 289-300, Belgium, 2002.

31. Cristian J. Luciano. Haptics-Based Virtual Reality Periodontal Training Simulator. University of Illinois at Chicago, Master thesis, 2006.

32. Martin Held. Bounding-Volume Hierarchy.
http://www.cosy.sbg.ac.at/~held/projects/collision/bvt.html
17 augustus 2007.

33. OpenHaptics Toolkit Datasheet.
http://www.sensable.com/documents/documents/FFM_FFMP_datasheet_lo.pdf
17 augustus 2007.

34. SensAble Technologies Incorporated.
http://www.sensable.com/products-haptic-devices.htm
17 augustus 2007.

35. Force dimension.
http://www.forcedimension.com/
17 augustus 2007.

36. Novint.
http://www.novint.com/
17 augustus 2007.

37. Immersion Corporation.
http://www.immersion.com
17 augustus 2007.

38. Intuitive Surgical.
http://www.intuitivesurgical.com/
17 augustus 2007.

39. Alsim.
http://www.alsim.com/
17 augustus 2007.

40. Defenselink.
http://www.defenselink.mil/transformation/images/photos/2006-02/Hi-Res/OCPA-2006-02-27-083502.jpg
17 augustus 2007.

41. Kenneth E. Hoff III. Bounding Volume Hierarchies and Spatial Partitoning. COMP-236 lecture, UNC Chapel Hill, slide 19, 2000.

42. Dirk Reiners. Scene Graph Rendering. OpenSG Forum, pagina 1 – 15, 5 maart 2002.

43. Wes Bethel, J. Dean Brederson. Hierarchical Parallellism in a Scene Graph. The 2001 IEEE Symposium on Parallel and Large-Data Visualisation and Graphics (PVG 2001), pagina 1 – 2, 2001.

44. Bruce F. Naylor. A tutorial on Binary Space Partitioning Trees. Spatial Labs Inc.

45. Wikipedia.org
http://en.wikipedia.org/wiki/BSP_tree
17 augustus 2007.

46. Samuel Ranta-Eskola. Binary Space Partitioning Trees and Polygon Removal in real Time 3D Rendering. Master Thesis. Information Technology Computing Science Department Uppsala University, Sweden, 19 januari 2001.

47. Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomeranets, Markus Gross. Optimized Spatial Hashing for Collision Detectionof Deformable Objects. Computer Graphics Laboratory, ETH Zurich. 19 – 21 november 2003.

48. Bruno Heidelberger, Matthias Tschner, Markus Gross. Volumetric Collision Detection for Deformable Objects. Computer Graphics Laboratory, ETH Zurich, 8 april 2003.

49. Maurizio de Pascale and Domenico Prattichizzo. The Haptik Library: a component based Architecture for Uniform Access to Haptic Devices. Dipartimento di Ingegneria dell'Informazione, University of Siena. Pagina 7 - 8.

# Bijlage 1

```
void cRenderWidget::drawCursor()
{
    static const double kCursorRadius = 0.5;
    static const double kCursorHeight = 1.5;
    static const int kCursorTess = 15;
    HLdouble proxyxform[16];

    GLUquadricObj *qobj = 0;

    glPushAttrib(GL_CURRENT_BIT | GL_ENABLE_BIT | GL_LIGHTING_BIT);
    glPushMatrix();

    if (!gCursorDisplayList)
    {
        gCursorDisplayList = glGenLists(1);
        glNewList(gCursorDisplayList, GL_COMPILE);
        qobj = gluNewQuadric();

        gluCylinder(qobj, 0.0, kCursorRadius, kCursorHeight,
                    kCursorTess, kCursorTess);
        glTranslated(0.0, 0.0, kCursorHeight);
        gluCylinder(qobj, kCursorRadius, 0.0, kCursorHeight / 5.0,
                    kCursorTess, kCursorTess);

        gluDeleteQuadric(qobj);
        glEndList();
    }

    hlGetDoublev(HL_PROXY_TRANSFORM, proxyxform);

    glMultMatrixd(proxyxform);
    glScaled(gCursorScale*2, gCursorScale*2, gCursorScale*2);

    glEnable(GL_COLOR_MATERIAL);
    glColor3f(1.0, 0.0, 0.0);

    glCallList(gCursorDisplayList);
    glDisable(GL_COLOR_MATERIAL);

    glPopMatrix();
    glPopAttrib();
}
```

# Bijlage 2

```
void cRenderWidget::updateWorkspace()
{
    GLdouble modelview[16];
    GLdouble projection[16];
    GLint viewport[4];

    glGetDoublev(GL_MODELVIEW_MATRIX, modelview);
    glGetDoublev(GL_PROJECTION_MATRIX, projection);
    glGetIntegerv(GL_VIEWPORT, viewport);

    hlMatrixMode(HL_TOUCHWORKSPACE);
    hlLoadIdentity();

    hluFitWorkspace(projection);

    gCursorScale = hluScreenToModelScale(modelview, projection,
    viewport);
    gCursorScale *= CURSOR_SIZE_PIXELS;
}
```

# Bijlage 3

```
cursor = new cMeta3dofPointer(world, 0);
world->addChild(cursor);
cursor->setPos(0.0, 0.0, 0.0);
cursor->setWorkspace(1.0,1.0,1.0);
cursor->setRadius(0.01);
cursor->initialize();
cursor->start();
```

# Bijlage 4

```
for (unsigned int i=0; i<cursor->m_pointForceAlgos.size(); i++)
    {
            cProxyPointForceAlgo* cur_proxy =
dynamic_cast<cProxyPointForceAlgo*>(cursor->m_pointForceAlgos[i]);
            if ((cur_proxy != NULL) && (cur_proxy->getContactObject() !=
NULL))
            {
                  cGenericObject* temp = cur_proxy->getContactObject();

                  if(temp != object && temp != object2)
                  {
                        cVector3d objectPos = temp->getGlobalPos();
                        cVector3d cursorForce = cursor-
>m_lastComputedGlobalForce;
                        rotVelocity.add( cMul(-10.0 * increment,
cCross(cSub(cursorPos, objectPos), cursorForce)));
                        rotVelocity.mul(1.0 - increment);
                        if (rotVelocity.length() > CHAI_SMALL)
                        {
                              temp->rotate(cNormalize(rotVelocity),
increment * rotVelocity.length());
                        }
                  }
            }
      }
```

# Bijlage 5

```
//De routes in X3D
<MouseSensor DEF="MS"/>
<Group DEF="GROUP"/>
<PythonScript DEF="PS" url="H3Ddemo.py">
      <Group USE="GROUP" containerField="references"/>
</PythonScript>
<ROUTE fromNode="MS" fromField="rightButton" toNode="PS"
toField="add_item"/>

//Het pythonscript
#rootnode van de scenegraph zodat we items kunnen toevoegen
group, = references.getValue()

#geometrie zetten, kan ook andere shape zijn natuurlijk, of een mesh (of
via url een andere x3d-file)
currentGeometry = createX3DNodeFromString( """<Sphere radius="0.02" />
""")[0]

#functie die we roepen via de routes in ons x3d-bestand
class AddObject( AutoUpdate( SFBool ) ):
  def update( self, event ):
    if( event.getValue() ):
      temp, dn = createX3DNodeFromString( """\
      <Transform>
         <Shape DEF="SHAPE" >
            <Appearance> <SmoothSurface />
              <Material DEF="MATERIAL" />
            </Appearance>
         </Shape>
      </Transform>""")
      color = RGB( random.random(), random.random(), random.random() )
      dn["MATERIAL"].diffuseColor.setValue( color )
      dn["SHAPE"].geometry.setValue( currentGeometry )
      temp.translation.setValue( Vec3f(  color.r * 0.5 - 0.25,
                                         color.g * 0.5 - 0.25,
                                         color.b * 0.5 - 0.25 ) )
      group.children.push_back( temp )
    return event.getValue()

#onze functie instantieren
add_item = AddObject()
```

# Bijlage 6

```
void cRenderWidget::drawCursor()
{
	static float blue[3] = { 0, 0, 1 };
	static float red[3] = { 1, 0, 0 };
	static float white[3] = { 1, 1, 1 };

	glPushMatrix();
		HAL::Vector position = m_pDevice-
>GET_VALUE(SCP).GetPosition();
		glTranslated( position.X() * 10, position.Y() * 10,
position.Z() * 10 );
		glMaterialfv( GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, blue
);
		auxSolidSphere(.1);
	glPopMatrix();

	glPushMatrix();
		position = m_pDevice->GET_VALUE(Location).GetPosition();
		glTranslated( position.X() * 10, position.Y() * 10,
position.Z() * 10 );
		glMaterialfv( GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, red
);
		auxSolidSphere(.1);
	glPopMatrix();
}
```

# Bijlage 7

```
Delta.LeftHanded  =     TRUE
Delta.MaxForce    =     4.0000

HaptikLibrary.numberOfPlugins =     3
HaptikLibrary.plugin0_0 =     Haptik.Phantom42.dll
HaptikLibrary.plugin0_1 =     Haptik.Phantom40.dll
HaptikLibrary.plugin0_2 =     Haptik.Phantom31.dll
HaptikLibrary.plugin1_0 =     Haptik.Delta.dll
HaptikLibrary.plugin2_0 =     Haptik.Spectre.dll

MouseSpectre.Scale.force      =     1.000000
MouseSpectre.Scale.moveX      =     0.200000
MouseSpectre.Scale.moveY      =     0.200000
MouseSpectre.Scale.moveZ      =     1.000000
MouseSpectre.Scale.rotateX    =     1.000000
MouseSpectre.Scale.rotateY    =     1.000000
MouseSpectre.Scale.rotateZ    =     0.100000

MouseSpectre.Window.Opacity   =     0.800000
MouseSpectre.Window.PositionX =     10
MouseSpectre.Window.PositionY =     10
MouseSpectre.Window.Transparent     =     FALSE

MouseSpectre.Workspace.maxX   =     500.000000
MouseSpectre.Workspace.maxY   =     500.000000
MouseSpectre.Workspace.maxZ   =     500.000000
MouseSpectre.Workspace.minX   =     -100.000000
MouseSpectre.Workspace.minY   =     -100.000000
MouseSpectre.Workspace.minZ   =     -100.000000

MouseSpectre.Frequency  =     1000.000000
MouseSpectre.LeftHanded =     TRUE
MouseSpectre.MaxForce   =     50.000000

Phantom.LeftHanded      =     TRUE
Phantom.MaxForce  =     4.000000
```

# 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, de gevraagde informatie in te vullen (en de overeenkomst te ondertekenen en af te geven).*

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
**Extending an Haptic API**
Richting: **Master in de informatica** Jaar: **2007**
in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik 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 mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Ik ga akkoord,

**Pascal Porta**

Datum: **22.08.2007**