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 met

Titel: Ambient projection / stuurbare projectie met spiegels
Richting: 2de masterjaar in de informatica - Human Computer Interaction Jaar: 2009

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,

SLENDERS, Jonathan

Datum: 14.12.2009

Ambient projection / stuurbare projectie met spiegels

Jonathan Slenders

promotor :
Prof. dr. Kris LUYTEN

Eindverhandeling voorgedragen tot het bekomen van de graad master in de informatica Human Computer Interaction

Universiteit Maastricht

universiteit hasselt

Samenvatting

Traditionele werkomgevingen gebruiken een vaste monitor met toetsenbord en muis, waarbij de gebruiker ervoor gaat zitten om met het systeem te interageren. Een alternatief is een stuurbare projectie; een projectiesysteem dat elk oppervlak uit de omgeving in een interactief scherm kan veranderen. Het realiseren hiervan vereist onder andere een correctie van schuin geprojecteerde beelden en hardware welke de projectie kan richten. Verder kan er invoer nodig zijn in de vorm van het herkennen van gebaren of tracking van de gebruiker. Een stuurbare projectie verkent vakgebieden als augmented reality, informatievisualisatie en interactietechnieken binnen de Human Computer Interaction. Onderwerpen die aan bod komen zijn: peephole displays, kalibratie van de projectie, het combineren van meerdere projectors en het software design.

Voorwoord

Het onderwerp van deze thesis, Ambient Projection, kwam als voorstel van prof. dr. Kris Luyten. Hij wist dat ik interesse had voor elektronica – mijn bachelorthesis bevatte immers ook een stukje elektronica – dus dit zou een interessant vakgebied zijn voor mijn thesis. Mij leek het realiseren van een zogenaamde 'stuurbare projectie' ook erg interessant, dus heb ik het voorstel graag geaccepteerd.

Om te weten in hoeverre alles haalbaar zou zijn, ben ik al erg vroeg begonnen met het maken van een prototype. Al snel werd duidelijk dat niet alles even vlot zou verlopen; de servomotors die ik gebruikte in mijn constructie waren niet altijd even nauwkeurig ondanks hun kwaliteit.

Deze thesis heb ik uitgewerkt gedurende mijn twee masterjaren. Het academiejaar (2007–2008) bestond voornamelijk uit onderzoek en het testen van een eerste prototype. Het ontwerpen van de meeste software en ook het realiseren van een tweede prototype is in het tweede jaar (2008–2009) gebeurd.

Hier wil ik ook mijn begeleider, Tom De Weyer, bedanken voor de hulp die hij heeft gegeven en de tijd die hij gebruikt heeft om het eerste prototype te bouwen. Ook begeleidster Karoline Petermans bedank ik voor haar hulp bij deze thesis en het nakijken van deze tekst. Verder dank ik heel graag prof. dr. Kris Luyten om alles op te volgen en de nodige hulp te geven wanneer nodig en ook mijn ouders voor hun steun en in het bijzonder mijn mama om de hele tekst te controleren op spelfouten.

Inhoudsopgave

1 Inleiding **1**

1.1 Scenario 1

1.2 Toepassingen 2

1.3 Uitdagingen 3

2 Literatuurstudie en onderbouw **4**

2.1 The everywhere display 4

2.2 Toepassingen 6

2.2.1 Augmented reality 6

2.2.2 Een stuurbare projector als resolutieverbeteraar (subregion enhencement) 6

2.2.3 Games 8

2.2.4 Het ideale kantoor 8

2.3 Peephole displays 10

2.4 Interactie met de projectie 12

2.4.1 Een oppervlakte als interactief touchscreen 12

2.4.2 Andere interactietechnieken 14

2.4.3 Tracking van de gebruiker 15
2.5 Beperkingen van projectors 16
2.6 Kalibratie 18
2.6.1 Geometric calibration 18
2.6.2 Photometric calibration 20
2.7 Software design 21
2.8 Privacy 22

3 Vereisten voor het bouwen van een projectiesysteem 23
3.1 Enkele aandachtspunten 23

4 Architectuur voor een everywhere display projection 25
4.1 Software 25
4.1.1 De rendering engine 26
4.1.2 Toepassen van de beeldtransformatie 27
4.1.3 Applicaties 29
4.1.4 De verschillende componenten 31
4.1.5 Configuratie van de ED-projectie 35
4.1.6 Voorbeelden van applicaties 37
4.1.7 Ontwerpen van een Hello World-applicatie 42
4.1.8 Meerdere projectiesystemen combineren 43
4.2 Hardware 44
4.2.1 LCD output 44

4.2.2 Button input 45
4.2.3 De mechanische besturing van de projector 46
4.2.4 Besturing van de servomotors 49
4.2.5 Kalibratie tussen hard- en software 53
4.3 Mogelijke uitbreidingen 54
4.3.1 ED over een netwerk 54
4.3.2 Security 55
4.3.3 Privacy 56
4.3.4 Geavanceerde invoer 56
4.3.5 Projectie op plafond en vloer 57
4.4 Toekomstig werk 59

5 Evaluatie en conclusie **60**

6 Bijlagen **64**

6.1 Application interface 64
6.2 Omgevingsmodel 66
6.3 Lengte van de draden 69

Lijst van figuren

2.1 Het tweede prototype van IBM, november 2001 (Bron: [16]) 5

2.2 Subregion enhencement (Bron: [6]) 7

2.3 Een reeks van zelfconfigurerende projectors die samen een groter, hoog resolutiebeeld vormen. (Bron: [18]) 8

2.4 Interactief spel waarbij de speler moet proberen de kikker te tikken. (Bron: [8]) 9

2.5 Detectie van gebruikersinteractie door verschillen tussen opeenvolgende frames te analyseren. (Bron: [16].) 13

2.6 Geometrische kalibratie 20

2.7 Luminantie (bij een Proxima 6850-projector) in functie van de invoerwaarde voor de rode, groene en blauwe componenten. (Bron: [24].) . 21

4.1 Componenten van een beeldtransformatie 28

4.2 Prototype van de beeldtransformatie, geïmplementeerd in C# 29

4.3 Componentendiagram 32

4.4 UML-schema 34

4.5 Tonen van pan- en tilthoek van een bepaalde projector op een muur van de ED-projectie. 38

4.6 Pong 39

4.7 Een videofilm die afspeelt op een muur van de ED 40

4.8 Remote keyboard client. De keyboard input van dit venster wordt over het netwerk verstuurd naar een bepaald venster in de projectie volgens een input mapping. 41

4.9 Het renderen van output naar een scherm op een muur 41

4.10 Pan- en tiltoutput op een LCD-scherm 45

4.11 Bovenaanzicht van de output/inputmodule 46

4.12 Tekening van het eerste prototype 47

4.13 Prototype in actie 48

4.14 Onderkant van het prototype. Hier zien we de drie wielen die elk één draad besturen en zo de spiegel kunnen richten. 49

4.15 Een alternatief voor ons prototype 50

4.16 Tweede prototype 50

4.17 Tweede prototype, close-ups 51

4.18 Controller voor de servomotors. Deze heeft een USB-ingang en vier uitgangen voor de servomotors. (Bron: http://www.phidgets.com/) . 52

4.19 Pulse width modulation – servobesturing (Bron: http://servocity.com) 53

4.20 Het microprocessorcircuit dat eerder al gebruikt werd, is nu uitgebreid om de servo's te besturen over ethernet. 54

4.21 De XWand als invoerapparaat (Bron: [23].) 57

4.22 Verplaatsen van de projectie tussen verschillende vlakken 58

6.1 De drie draden 70

Listings

4.1 Berekenen van de beeldtransformatie 29
4.2 Configuratie van een projectievlak (muur) 35
4.3 Configuratie van een projector 35
4.4 Configuratie van een applicatie 36
4.5 Configuratie van een window mapping 36
4.6 Configuratie van een invoerapparaat 36
4.7 XAML code voor een Hello World-applicatie 42
4.8 C# code voor een Hello World-applicatie 42
6.1 Application interface 64
6.2 Het omgevingsmodel 66

Hoofdstuk 1

Inleiding

In de hedendaagse computeropstellingen wordt gewoonlijk gebruik gemaakt van een vaste monitor, vast toetsenbord en een vaste muis. Bij laptops is dit 'vast zijn' veel minder aanwezig doordat het apparaat zelf draagbaar is, maar de locatie van het beeldscherm is nog steeds afhankelijk van de positie van de hardware.

In deze thesis wordt een oplossing besproken die het beeld op een flexibele manier kan verplaatsen doorheen de ruimte, en dit volgens een volledig softwarematige methode. Er wordt geconcentreerd op opstellingen waarbij projectors (beamers) gebruikt worden om een gerichte projectie te realiseren. Deze projectie zou van elk stuk oppervlak uit de omgeving – bijvoorbeeld een gedeelte van een kast of tafel – gebruik moeten kunnen maken om informatie aan de gebruiker over te brengen. Dit wordt ook een ambient projection genoemd.

1.1 Scenario

Wanneer Harry aankomt in de kantoorruimte van zijn werk, detecteert het kantoor automatisch zijn aanwezigheid. Hij hoeft niet in zijn agenda te kijken want op de grote muur voor hem wordt al meteen de dagplanning geprojecteerd. Het eerste agendapunt is een vergadering met enkele collega's. Nadat hij met een drukgebaar een virtuele knop van de projectie met als label 'gidsen naar vergaderruimte' aan-

klikt, verplaatst de projectie zich naar de vloer en verandert in een pijl. Harry verlaat het kantoor en volgt de pijl – die netjes voor hem uit beweegt – door de gangen tot aan de vergaderzaal.

1.2 Toepassingen

Een toepassing is het automatisch volgen van de gebruiker [21]. Deze beschikt over een tracker welke de mogelijkheid geeft om zijn positie waar te nemen; het systeem past aan de hand daarvan de projectierichting aan zodat de gebruiker gevolgd wordt. Het nut hiervan is dat de gebruiker continu geassisteerd kan worden door de projectie.

Ook het omgekeerde is mogelijk: zoals in de use case kan de gebruiker de projectie volgen. Dit kan zijn nut bewijzen in musea en openbare gebouwen maar ook in de bedrijfswereld.

Voor zo een systeem kunnen waarschijnlijk talloze spelletjes ontworpen worden. In het bijzonder wanneer een groot speloppervlak nodig is, en slechts een klein, geconcentreerd gedeelte hiervan gelijktijdig in gebruik is. Een tennisspel hoeft bijvoorbeeld enkel de positie van de bal te tonen, maar deze bal kan wel over een grote oppervlakte bewegen. Ook spelletjes die in plaats van een virtuele wereld de fysieke wereld gebruiken als onderdeel van het spel kunnen een toepassing vormen [13], [8].

Een heel ander domein waar een gerichte projectie zijn nut kan bewijzen, is op publieke plaatsen zoals in winkels waar vandalisme een probleem vormt [13]. Een duur beeldscherm kan vervangen worden door een projector die vanop een veilige afstand een virtueel beeldscherm projecteert. Daar waar een klassieke monitor altijd in het bereik van de gebruiker moet staan, kan een beamer op een veilige lokatie – bijvoorbeeld enkele meters hoog – opgehangen worden. In dit geval is het stuurbare aspect wel minder of zelfs niet van belang.

1.3 Uitdagingen

Een van de vragen die we ons moeten stellen is: hoe kan de hardware voor de projectie vlot bestuurd worden via software? Wanneer omwille van technische beperkingen onder een hoek geprojecteerd moet worden, hoe kan dan het beeld gecorrigeerd worden zodat het resultaat identiek is aan een loodrechte projectie? Als meerdere projectors nodig zijn om een bepaalde oppervlakte te beschijnen, hoe kunnen we dan naden tussen deze beelden voorkomen of minimaliseren?

Een andere uitdaging stelt zich bij interactieve toepassingen. Hoe kan de gebruiker interageren met een geprojecteerd scherm? Een traditionele muis en traditioneel toetsenbord zijn een minder aantrekkelijke optie; deze vereisen immers het meedragen van extra hardware. Het gebruik van gebaren zoals in de use case 1.1 is ook niet vanzelfsprekend.

Hoofdstuk 2

Literatuurstudie en onderbouw

2.1 The everywhere display

IBM heeft tijdens een onderzoek een gelijkaardige opstelling gerealiseerd met de naam: 'The everywhere display project' [16]. In het jaar 2000 had IBM al een eerste prototype. Dat maakt gebruik van een pan- en tiltspiegel uit discoverlichting. De spiegel wordt dan ook bestuurd via het DMX-protocol, erg bekend in de lichtindustrie. De spiegel zelf wordt vlak voor de projector gemonteerd en door deze te richten kan men een willekeurig oppervlak op de muur of vloer beschijnen. Figuur 2.1 toont het tweede prototype.

Het is waarschijnlijk ook IBM die het grootste onderzoek in dit domein heeft uitgevoerd. IBM heeft bovendien ook onderzoek gedaan naar interactie met projecties [9] [13] [8] en heeft testen gedaan met simulaties van reële omgevingen, bestaande uit een winkel [14] [15] en een kantooromgeving met de naam Bluespace [12].

Mitsubishi heeft ook een gelijkaardig onderzoek gedaan met *iLamps* [19] [18]. Hier had men echter de omgekeerde bedoeling; namelijk het beeld steeds op (ongeveer) dezelfde plaats te houden en dat terwijl de projector zich kan verplaatsen. Het nut hier is dat bij dit systeem de projector op eender welke plaats in de ruimte kan staan, zolang het projectieoppervlak vanuit die positie bereikbaar is. Via een kalibratiesysteem wordt dan automatisch de keystone en eventuele rotatie hersteld.

Figuur 2.1: Het tweede prototype van IBM, november 2001 (Bron: [16])

[22] beschrijft een omgeving waarin een reeks van omnidirectionele projectors wordt gebruikt. Deze projectors worden ingezet om objecten zoals tafels en muren als projectiescherm te gebruiken. Uiteindelijk wordt de opstelling zelfs uitgebreid om een draagbaar klembord – voorzien van trackers – te belichten. Er kunnen telkens meerdere projectors ingezet worden voor elk beeld. Deze zullen dan elkaar aanvullen en schaduwen vermijden zolang minstens één projector 'toegang' heeft tot elk oppervlaktefragment. Om dit te realiseren wordt een kalibratiesysteem gebruikt dat beelden van meerdere projectors naadloos op elkaar mapt. Een pan- en tilt-camera wordt gebruikt om deze kalibratie automatisch te voltooien. In een meer geavanceerde opstelling wordt ook telkens het licht gemeten voor verscheidene regio's op het geprojecteerde beeld. Zo kunnen verschillen in lichthoeveeldheid tussen projectors onderling maar ook binnen één projector gecorrigeerd worden.

2.2 Toepassingen

2.2.1 Augmented reality

Meestal wordt augmented reality gerealiseerd door een head-mounted display (HMD) te gebruiken. De positie hiervan wordt getrackt en aan de hand daarvan wordt de echte wereld aangevuld met virtuele objecten via de HMD.

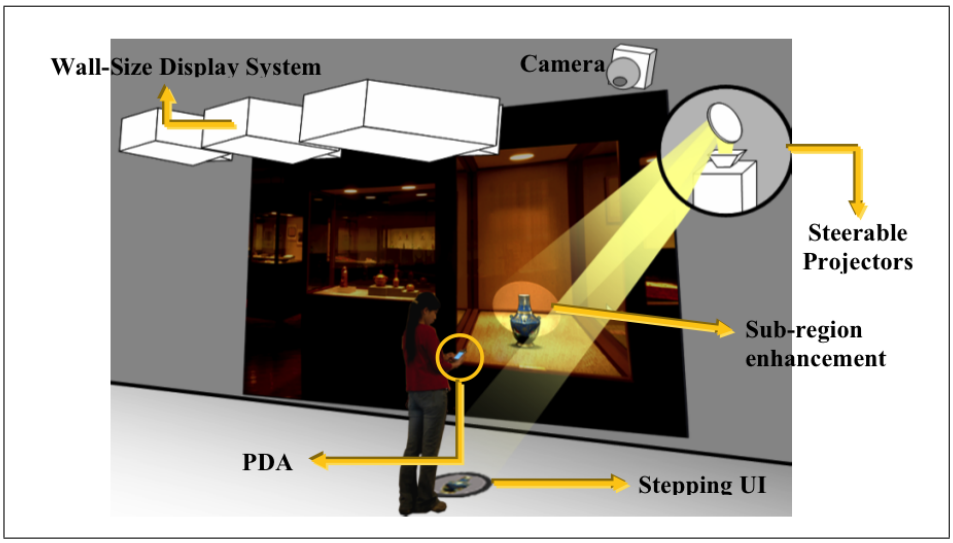
Augmented reality kan op een gelijkaardige manier met een stuurbare projectie gerealiseerd worden. In plaats van de virtuele objecten op het interne scherm van de HMD te projecteren, kunnen ze in de echte wereld rechtstreeks op objecten zoals tafels, muren en kasten geprojecteerd worden.

Een voordeel dat hiermee geïntroduceerd wordt, is dat de gebruiker augmented reality beleeft zonder dat hij hiervoor extra hardware moet dragen. Ook de tracking moet veel minder nauwkeurig zijn. Een HMD moet zeer nauwkeurig getrackt kunnen worden om het virtuele beeld synchroon met de echte wereld te laten bewegen. (Als de gebruiker zijn hoofd naar links draait, moet het beeld in de HMD met dezelfde snelheid naar rechts bewegen om een constante positie in de werkelijke wereld te behouden.) Bij de aanpak met een stuurbare projectie is de enige nodige tracking deze welke de kijkrichting van de gebruiker bepaalt, zodat er op de juiste plaats geprojecteerd kan worden. Het geprojecteerde beeld is dan altijd synchroon met de reële wereld omdat het er werkelijk bovenop geprojecteerd wordt.

2.2.2 Een stuurbare projector als resolutieverbeteraar (subregion enhencement)

Een stuurbare projectie kan gebruikt worden als uitbreiding van een grotere vaststaande projectie. Het centrum, of eender welk gedeelte van de vaste projectie dat belangrijk is, zou in een hogere resolutie kunnen worden weergegeven dan de rest van de projectie zoals in figuur 2.3 [6]. Zo een opstelling wordt ook wel een *foveal display* genoemd, waarbij het hogeresolutiegedeelte de *foveal inset* is [2].

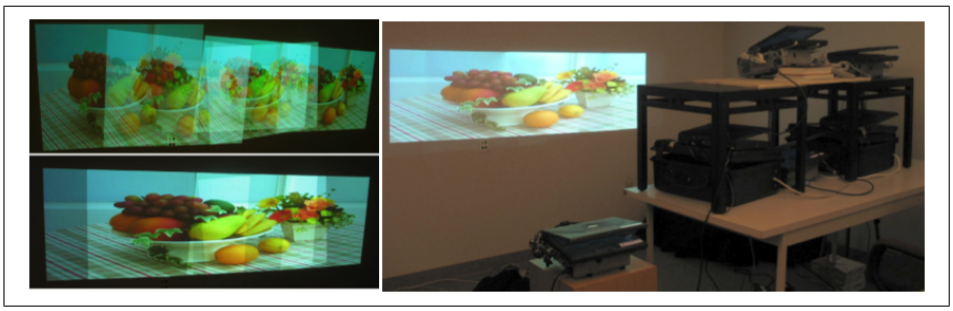
Het resultaat van deze opstelling is gelijkaardig aan hoe wij onze ogen vaak ge-



Figuur 2.2: Subregion enhencement (Bron: [6])

bruiken. We kunnen focussen op een klein detail zonder dat we de context verliezen; het centrum van onze retina ('fovea' genoemd) heeft een hogere resolutie dan het omliggende gedeelte. We kunnen hierdoor een klein gebied heel gedetailleerd waarnemen (in hoge resolutie) binnen een grotere, minder gedetailleerde wereld (de context in lagere resolutie). Voor het projectiesysteem moeten deze twee projectors natuurlijk op elkaar afgestemd worden, wat de nodige kalibratie vereist.

PixelFlex [24] is een oplossing die bestaat uit een rij projectors waarbij de gebruiker kan kiezen hoe deze reeks naar een vlakke muur moet gericht worden. Hij kan kiezen om alle projectors naast elkaar te richten zadat een breed, muurvullend scherm ontstaat. Een andere optie is alle projectors op ongeveer hetzelfde stukje muur te richten zodat er een kleiner scherm met veel grotere pixeldichtheid ontstaat. Andere mogelijkheden zijn het genereren van stereobeelden of niet-rechthoekige projectiebeelden. De kalibratie van PixelFlex gebeurt automatisch door middel van een camera, binnen enkele minuten nadat alle projectors gericht zijn. Ook Mitsubishi heeft een zelfconfigurerend projectiesysteem gebouwd tijdens het iLamps project [18].



Figuur 2.3: Een reeks van zelfconfigurerende projectors die samen een groter, hoog resolutiebeeld vormen. (Bron: [18])

2.2.3 Games

Stuurbare projecties kunnen tal van toepassingen hebben in computergames. De projectie kan op de grond gericht worden waarbij de vloer een soort van interactief speelbord vormt. In Japan was er een erg populair arcadespel waarbij de spelers op bepaalde plaatsen op de grond moesten stappen op het ritme van de muziek. Zo een ED-projectiesysteem kan geherprogrammeerd worden met andere software om andere games mogelijk te maken. De spelmogelijkheden zijn vrijwel eindeloos. Ook traditionele spellen zoals 'Age of Empires' en 'Sim city' kunnen aangepast worden om op een deel van de vloer, de muur, of zelfs het bed van een kind gespeeld te worden [8].

Een meer geavanceerd spel zou de omgeving deel kunnen laten uitmaken van het spel. Meubels of een vuilnisbak zouden met beeldherkenningssoftware herkend kunnen worden en een rol kunnen krijgen in het spel. Het zal echter nog enige tijd duren voordat dit soort systemen op de markt zal komen.

Op figuur 2.4 wordt een spel gespeeld waarbij een kind een kikker moet aantikken. Wanneer de speler met zijn hand in de buurt van de kikker komt, springt de kikker vliegensvlug naar een andere locatie die het kind weer moet vinden.

2.2.4 Het ideale kantoor

Het ideale kantoor is een kantoor dat automatisch de wensen van de gebruiker aanvoelt en zich dan daaraan kan aanpassen. Het is dus zeer sterk context-aware.

Figuur 2.4: Interactief spel waarbij de speler moet proberen de kikker te tikken. (Bron: [8])

BlueSpace [12] is zo een testproject. Een actieve badge die de gebruikers bij zich dragen, helpt het systeem te herkennen wie in het kantoor binnenkomt. Op basis van vooraf vastgelegde voorkeuren zal het kantoor automatisch de temperatuur, kleur van verlichting en andere instelbare elementen aanpassen. Het is aangetoond dat de productiviteit van de werknemer sterk beïnvloed kan worden door deze factoren.

Werknemers hebben zowel nood aan privéruimte als aan een plaats om collaboratief samen te werken. Dit laatste wordt moeilijk op een computerscherm met standaardafmetingen wanneer er meer dan twee personen bij de taak betrokken zijn. In traditionele omgevingen wordt dan een vergaderzaal opgezocht, maar BlueSpace kan zich transformeren tot een ruimte waar collaboratief gewerkt kan worden. Er staat een ED-projector die elk stuk muur in een scherm kan transformeren, wat kan helpen bij de samenwerking tussen personen.

De ED-projector bewijst ook zijn nut wanneer het kantoor als privéruimte gebruikt wordt. Zo zal een binnenkomende e-mail gewoonlijk terechtkomen op het computerscherm waaraan de werknemer zit. Maar wanneer deze persoon ergens rechtstaat, worden deze binnenkomende berichten op de muur voor hem geprojecteerd. (De stoelen bevatten dus drukgevoelige sensors die controleren of deze persoon zit of staat.) De ED-projector zou ook zijn agenda of ToDo-lijst op de muur kunnen projecteren. Indien het scherm niet nodig is om de taak uit te voeren, kan het als virtueel venster naar buiten toe gebruikt worden waarbij het verbonden is met een buitenstaande camera.

De persoon die in het kantoor werkt, kan ook een status instellen (bezet, even weg, vrij, even geduld, lunch, ...) en afhankelijk van deze status zal het systeem zich

anders gedragen. Bijvoorbeeld door binnenkomende berichten niet te tonen zolang deze status 'bezet' aangeeft. Verder kan de active badge helpen om de status van de gebruiker automatisch op 'even weg' te plaatsen wanneer de persoon het kantoor verlaat.

De uiteindelijke bedoeling van BlueSpace is dat elk kantoor deze eigenschappen bezit en dat zo een kantoor geen echte eigenaar heeft. Iemand die hier dan komt werken, kan een willekeurige kantoorruimte binnenwandelen. Hij wordt via zijn active badge herkend, en het kantoor past zich volledig aan aan zijn voorkeuren. Wanneer hij de volgende keer een ander kantoor binnenwandelt, zullen daar diezelfde voorkeuren geladen worden.

2.3 Peephole displays

Het is perfect mogelijk om een ruimte te creëren waarbij elk oppervlak, dus elke vloertegel, elk stuk muur en elk ander object als een scherm kan functioneren. (Bijvoorbeeld ontworpen uit e-paper of een LCD-scherm.) We hebben dan een heel groot oppervlak om gegevens over te brengen. Dit resulteert echter in een onrustige, 'vervuilde' omgeving [5]. Een gebruiker verwacht dat enkel het beeldscherm voor zijn ogen wijzigt, en niet letterlijk alles wat hij nog in zijn ooghoeken ziet.

De peephole-metafoor beperkt het interactieve gebied tot een klein oppervlak in een grotere, continu virtuele laag. Dit heeft verschillende voordelen:

- Schermruimte is niet goedkoop, maar doordat we nu met een kleiner scherm een groter oppervlak kunnen weergeven, wordt er op hardware bespaard;
- Mensen zijn een rustige, statische omgeving gewoon. Hiermee voorkomen we ongewilde afleiding.

Het heeft bovendien ook niet veel nut om informatie te tonen op locaties ver buiten het concentratiecentrum van de gebruiker. Deze kan zich maar op één bepaald gebied concentreren. Dit fenomeen staat bekend als het cocktailpartyeffect [5].

Zo een peephole-metafoor kan geïmplementeerd worden met een head-mounted display. Dit heeft echter als nadeel – net zoals eerder vermeld werd in de sectie over augmented reality – dat er een synchronisatie noodzakelijk is tussen de HMD en de omgeving. Wanneer meerdere gebruikers via een HMD door een peephole display kijken, moet bovendien ook tussen deze twee HMD's gesynchroniseerd worden. Alternatieven zonder deze problemen zijn een positiebewuste PDA, of – passend in de context van deze thesis – een ED-projector.

Een peephole display via een stuurbare projectie vereist dat het (3D-)model van de ruimte definieert wat er te zien is op welke oppervlaktegedeeltes. Het geprojecteerde beeld zal dan telkens gelijk zijn aan het beeld dat een virtuele camera in dit model zou zien als hij dezelfde richting uitkijkt als de projector in de reële wereld. De projectie werkt dus als een magische lens en onthult het overeenkomstige gedeelte uit het virtuele model [4].

Natuurlijk kan dit peephole-principe ook uitgebreid worden door meerdere apparaten – PDA's of ED-projectors – te combineren. De verschillende peepholes kunnen dan filters bevatten die bepaalde informatie verbergen en andere informatie tonen. Afhankelijk van het doel van de applicatie kan dit zijn om clutter te voorkomen of omwille van privacyredenen.

Er bestaan actieve en passieve peepholes [4]. Bij een actieve peephole bestuurt de gebruiker zelf de positie van het scherm via een invoerapparaat. Het goede ruimtelijk geheugen van een mens is hierbij een hulp. Een passieve peephole daarentegen richt zelf zijn projectie; deze begeleidt de gebruiker bij het uitvoeren van zijn taak door de aandacht van de ene naar de andere plaats te leiden. Deze beweging kan bovendien nog verrijkt worden met een geluidsbron die het beeld volgt door te klinken vanuit die richting. Dat is een acoustic peephole [5]. In het echte leven gaan we ook kloppen, wuiven of roepen wanneer we de aandacht van een persoon willen trekken. Dat protocol kan door een display ook toegepast worden.

2.4 Interactie met de projectie

2.4.1 Een oppervlakte als interactief touchscreen

Wanneer er informatie aan een mobiele gebruiker doorgegeven moet worden, dan moet deze gewoonlijk een toestel zoals een PDA of laptop dragen [14].

Het dragen van zo een object veroorzaakt altijd een overlast. De gebruiker moet een bepaald gewicht meenemen en heeft mogelijk maar één hand vrij. Verder hebben mobiele toestellen een beperkte batterijduur en zijn ze gevoelig voor schade.

Door gebruik te maken van een stuurbare projectie zouden deze problemen in theorie verholpen kunnen worden. De volgende twee elementen zijn voldoende om met een computer te interageren:

- een invoermethode en
- feedback.

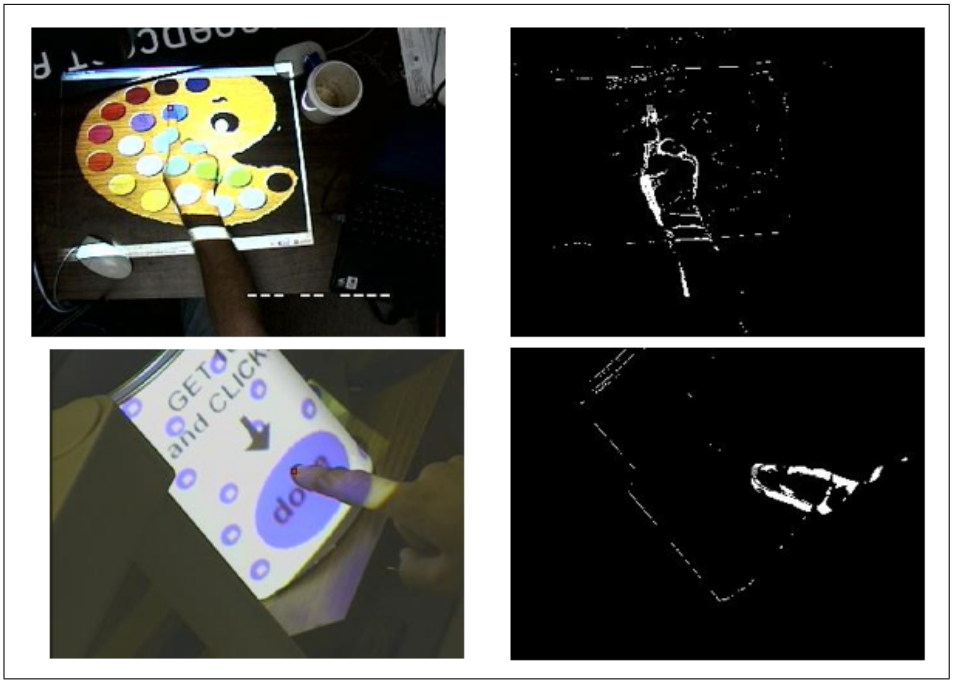
Feedback kan in visuele vorm gegeven worden door de projectie. Deze projectie komt terecht op alledaagse objecten zoals een tafel, muur, kast of eender welk object dat als projectiescherm kan dienen. Bijgevolg is het logisch dat deze objecten zelf geen input van de gebruiker kunnen voelen wat wel het geval is bij een touchscreen. Toch moeten we de touchscreen-metafoor zo goed mogelijk benaderen omdat deze erg natuurlijk aanvoelt.

Om een projectieoppervlak als touchscreen te kunnen gebruiken, moet de positie van de hand waargenomen kunnen worden. Dit kan met een camera gebeuren die naast de projector staat en met dezelfde spiegel naar het scherm kijkt. Het beeld van de camera moet dan dezelfde transformatie ondergaan om het correcte beeld te verkrijgen [13].

Herkenning van de hand van de gebruiker op basis van kleuren is erg moeilijk, want wanneer een hand voor het beeld komt (wat noodzakelijk is bij een touchscreen), dan zal deze de kleuren aannemen van het geprojecteerde beeld op die plaats. Wat wel

waargenomen kan worden, is een verplaatsing van de pixels. Deze wordt zichtbaar door twee opeenvolgende frames met elkaar te vergelijken.

Er bestaan twee aanpakken om een tracking-mechanisme via een videobron te realiseren [15]. Ten eerste kan het vorige frame van de projector afgetrokken worden van het camerabeeld. Dit resulteert in een beeld waarbij enkel de voorgrond, dus de gebruiker, zichtbaar is. Een tweede methode is een op voorhand opgenomen beeld aftrekken van het camerabeeld. Deze tweede methode zou betere resultaten geven als de overblijvende, geprojecteerde beelden via een masking-techniek ook nog verwijderd worden.



Figuur 2.5: Detectie van gebruikersinteractie door verschillen tussen opeenvolgende frames te analyseren. (Bron: [16].)

IBM beschijft in dat onderzoek een volledige winkelsituatie die nagebootst wordt waarbij de klant geassisteerd wordt door een ambient projection. Zo kan de klant het 'scherm' gebruiken om informatie te verkrijgen over een product dat hij op dat moment vastheeft. Ook kan hij producten zoeken waarbij pijltjes geprojecteerd worden in de richting van de locatie; de klant hoeft dus enkel deze pijlen te volgen.

Er is een test uitgevoerd om te onderzoeken hoe gebruikers een touchscreen gebruiken

[13]. Deze gebeurde aan de hand van een taak waar men m&m's op een bepaalde positie moet leggen om zo een afbeelding te creëren. De stuurbare projectie vormt hierbij een hulp door aan te wijzen waar de gebruiker de m&m's kan vinden, hoeveel hij er telkens moet nemen en waar hij ze uiteindelijk moet neerleggen [9].

Een probleem waar aandacht aan geschonken moet worden, is het bewust maken van de klant of gebruiker dat de projectie beïnvloed wordt door zijn acties, dat het systeem bewust is van dat waarmee de klant bezig is. Bij het spel met de m&m's is dat geen probleem omdat daar de projectie duidelijk onderdeel is van het spel. Maar bij de simulatie van een kledingwinkel blijkt het wel een probleem te zijn, omdat de gebruiker daar zijn taak – het zoeken naar kleding – perfect kan uitvoeren zonder hulp van een computer, en omdat hij het niet gewoon is om in een kledingwinkel geassisteerd te worden door een computer. De inhoud van het scherm geeft informatie weer over het kledingstuk dat de gebruiker vastheeft, maar het wordt aanzien als willekeurige reclame [15].

Wanneer op de vloer in plaats van op de muur wordt geprojecteerd, dan is het natuurlijker om met de voeten te interageren – vermits deze continu contact maken met de vloer. [6] bespreekt deze techniek; gebruikers kunnen met hun voeten stappen op de virtuele knoppen die ze willen activeren.

2.4.2 Andere interactietechnieken

Wanneer het niet mogelijk is om apparatuur mee te nemen, kunnen we als mens enkel op de natuurlijke manier interageren; dat is door te spreken of via een vorm van gebarentaal.

In [17] wordt een vision-based user interface gemaakt die bestaat uit een hiërarchie van widgets. Deze hiërarchie kan via een ED-projector geprojecteerd worden en elk widget heeft de mogelijkheid te reageren op bepaalde interacties. Een applicatie kan daar als een laag bovenop geïmplementeerd worden en hoeft geen kennis te hebben van de 'low-level' interactietechnieken.

Als het wel is toegestaan kleine 'gadgets' mee te nemen, dan hebben we verschillende andere opties. Een PDA kan gebruikt worden als afstandsbediening door via

een draadloos netwerk zoals Wi-Fi met het projectiesysteem te communiceren. Een laser pointer kan gebruikt worden om de plaats van de hoge resolutie-overlay (die met een stuurbare projectie wordt gerealiseerd) binnen de projectie te wijzigen [2]. Deze techniek kan natuurlijk ook gebruikt worden voor andere interactiedoeleinden, zoals het activeren van een knop binnen de GUI. Hiervoor is het wel noodzakelijk dat de plaats waar de laser naartoe gericht wordt, waargenomen kan worden door een camera en dat deze plaats nauwkeurig terug herleid kan worden naar de overeenkomstige plaats op het oorspronkelijke beeld. (Via een goede kalibratie of herkenning van het oorspronkelijke beeld in het opgenomen beeld.)

2.4.3 Tracking van de gebruiker

Een voor de hand liggende toepassing van een stuurbare projectie is een display dat de gebruiker volgt. Hiervoor moet het systeem de positie van de gebruiker kennen en het geprojecteerde beeld in die richting sturen.

Een publicatie van IBM [21] beschrijft hoe een user-trackingsysteem gerealiseerd kan worden. Het systeem bevat een model van de omgeving met alle mogelijke oppervlaktes waarop geprojecteerd kan worden. Het werkt als volgt:

1. Track de positie en oriëntatie van de gebruiker.
2. Selecteer een passend oppervlak gebaseerd op de positie van de gebruiker. Het is belangrijk dat de gebruiker niet in de projectiekegel komt te staan.
3. Stel de oriëntatie, grootte en vorm van dat oppervlak vast.
4. Stel alle parameters in om daarop te kunnen projecteren. Dit zijn onder andere spiegeloriëntatie, zoom, focus en afbeeldingstransformatie.
5. Projecteer het beeld.

Het tracken van personen gebeurt door middel van een realtimecamera. Het verschil tussen twee opeenvolgende frames wordt bestudeerd en hier wordt op zoek gegaan naar beeldfragmenten die op een hoofd lijken.

Volgens deze publicatie werkt het systeem goed, maar er zit altijd een kleine vertraging tot vijf seconden in om nauwkeurig tussen twee oppervlaktes om te schakelen. Soms kan het zelfs tot negen seconden duren [13]. Verder is het noodzakelijk om alles nauwkeurig te kalibreren, maar dat is gelukkig maar een eenmalige operatie.

Een andere methode om gebruikers te tracken, is via actieve RFID-tags; hierbij daalt het bereik wel tot 1 à 3 meter voor een enkele RFID-tag [5]. Positiebepaling via RFID zou geoptimaliseerd kunnen worden door bijvoorbeeld de volledige vloer te bekleden met RFID-tags. Als er vervolgens een RFID-lezer in de schoenen van deze gebruiker wordt geplaatst, kan zo erg nauwkeurig zijn positie bepaald worden.

2.5 Beperkingen van projectors

In één van de papers van IBM [16] worden verschillende problemen besproken in verband met het gebruik van projectors in deze situatie.

Helderheid en contrast : Als een projector in een verlichte ruimte wordt gebruikt, dan zal de kleur zwart niet altijd als zwart overkomen. Wanneer een projector wordt gebruikt met een voldoende hoge helderheid (bijvoorbeeld 1200 lumen), dan zal er toch voldoende contrast in het geprojecteerde beeld zitten.

Door de werking van onze ogen zullen we de oppervlaktes die oorspronkelijk wit waren toch waarnemen als zwart, omdat deze bij een voldoende sterke belichting van de projector veel minder licht weerkaatsen dan de belichte delen.

Wanneer het belangrijk is om zeer goede kleuren te bekomen, wordt een videoprojector aangeraden [6]. Deze heeft een juistere kleurrepresentatie dan computerprojectors.

Correctie van schuine projectievervorming : Een niet-loodrechte projectie veroorzaakt altijd een vervorming in het beeld. Dit is een niet-lineaire vervorming die processorintensief is om te berekenen [11]. Gelukkig kunnen grafische kaarten deze berekeningen hardwarematig uitvoeren.

Bij het projecteren van een beeld vindt juist de omgekeerde transformatie plaats van deze bij het waarnemen van het beeld. Wat hier gedaan wordt, is

deze omgekeerde transformatie toepassen op het beeld dat geprojecteerd moet worden. We projecteren dus wat de projector zou zien als deze onder die hoek zou kijken. De twee transformaties zullen elkaar opheffen en een correct beeld zal op de muur verschijnen [20].

Focus : Een projector moet altijd scherp gesteld worden op een bepaald dieptebereik ('depth of focus'). Deze beperking resulteert in onscherpe beelden wanneer op een gedeelte van een muur dat buiten het focusbereik ligt, geprojecteerd moet worden. In de opstelling van IBM wordt gebruik gemaakt van een projector waarbij de focus van de beamer via een seriële verbinding ingesteld kan worden. Huidige projectors hebben een vrij groot dieptebereik, dus mogelijk zal dat nu maar een klein probleem vormen.

Resolutie : De werkelijke resolutie van het beeld zal altijd kleiner zijn dan de resolutie van de projector zelf. De oorzaak hiervoor is de correctietransformatie die altijd een deel van het oppervlak verloren doet gaan. Verder is het aantal pixels ongelijkmatig verdeeld over het geprojecteerde oppervlak. En hoe schuiner geprojecteerd wordt, hoe meer de resolutie verkleind wordt. Een oplossing hiervoor is een projector met extra hoge resolutie gebruiken.

De uiteindelijke resolutie van een projector met 1024x768 pixels zal in de buurt van VGA komen, dus 640x480 pixels [13].

Schaduw door hindernissen : Dit blijft een nadeel bij het gebruik van een projector. Het probleem kan echter beperkt worden door hindernissen tussen de projector en het projectieoppervlak te voorkomen. In het bijzonder als de projectie langs boven gebeurt, is er minder kans op dit probleem, omdat dan de projectiekegel kleiner wordt. (De projectiekegel is het volume in de ruimte dat gebruikt wordt om het licht van de projector over te brengen naar het beeldscherm.)

Wanneer een tafel als projectieoppervlak wordt gebruikt, en er langs boven geprojecteerd wordt, ontstaat er een onvermijdelijke schaduw als de gebruiker over het scherm buigt. (Zoals bij het m&m-spel in [13].)

Gelukkig gaan mensen intuïtief hun eigen positie aanpassen om zo hun schaduw te minimaliseren [13].

Projectors die gebaseerd zijn op lasertechnologie zijn commercieel nog niet goed verkrijgbaar, maar deze hebben wel enkele voordelen. Laserprojectors hebben een onbeperkt focusbereik, zeer goede kleurverzadiging, een hoog contrast en een hoge resolutie [10].

2.6 Kalibratie

Voor een perfect resultaat zijn er twee types van kalibratie nodig, namelijk geometric calibration en photometric calibration.

[7] stelt een kalibratiesysteem voor om beelden naadloos te laten aansluiten. Dit wordt gedemonstreerd aan de hand van een projectieopstelling met 24 projectors. Het algoritme – dat lineaire homografieën gebruikt – zou nauwkeuriger en veel sneller zijn dan de resultaten die in eerdere onderzoeken zijn bereikt.

2.6.1 Geometric calibration

Het eerste type, geometrische kalibratie, is het uitlijnen van de beeldtransformatie volgens het omgevingsmodel bestaande uit projectievlakken. We moeten ervoor zorgen dat het geprojecteerde beeld en de projectiehoek steeds consistent aan elkaar blijven. Een kalibratie is noodzakelijk omdat hardware – in tegenstelling met software – geen onbeperkte nauwkeurigheid heeft.

Als een projector loodrecht voor een vlak geplaatst is, dan hoeft er geen beeldtransformatie uitgevoerd te worden. Maar als dit niet het geval is, moet het beeld gemanipuleerd worden zodat het resultaat equivalent is aan een loodrechte projectie.

Indien de projectie naar een bepaald deel van een oppervlak gericht moet worden, dan kunnen we – indien we over een perfect model van de omgeving beschikken – softwarematig met een nauwkeurigheid van 100% het beeld onder de juiste hoek transformeren, vermits deze transformatie wiskundig te berekenen valt. Het tekenen van onvervormde graphics op het vlak bestaat simpelweg uit het aanbieden van de invers vervormde graphics aan de projector [20]. De relatie tussen het coördinatensysteem op de muur en het lokale systeem in een vlak loodrecht op de projectiestraal

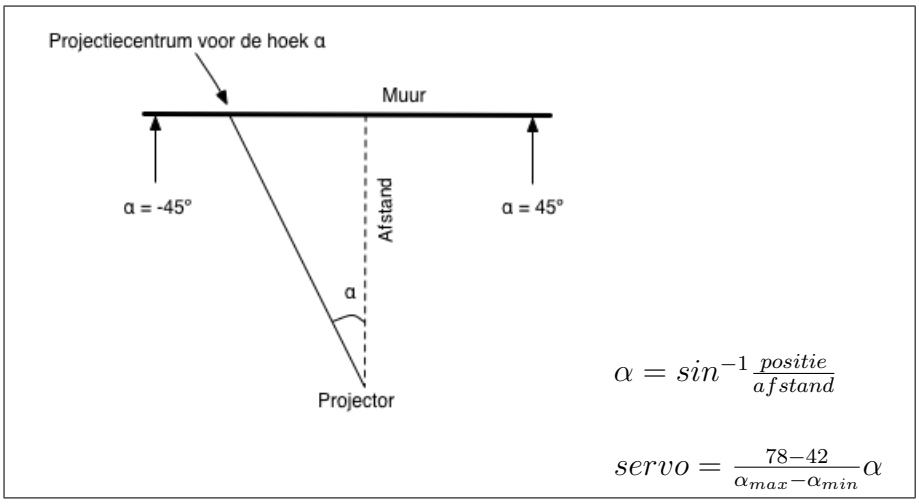
bestaat uit een 3x3 homography matrix. We hebben minstens vier paren van coördinaten nodig om deze matrix te creëren, en deze coördinaten zijn dan een equivalent voor het omgevingsmodel.

Wat bijzonder is aan de projectieopstelling die in [20] besproken wordt, is dat daar de projectiestraal steeds uit een centraal punt vertrekt. Dit in tegenstelling met de opstellingen uit [16], [2] en [6]. Het voordeel dat we verkrijgen bij een fixed-center-projectie is dat de homografiematrix altijd vanuit dat punt berekend kan worden. Bij de opstelling van IBM wordt de projectiestraal via een spiegel gericht; bijgevolg is daar de spiegel bij benadering het centrum. Hun pan- en tiltmechanisme, dat ze gebruiken om de spiegel te draaien heeft echter een rotatiecentrum dat niet op het spiegelvlak maar achter de spiegel gelegen is. Telkens wanneer de pan-of tilthoek aangepast wordt, zal bijgevolg de spiegel ook een ander centrum hebben en wordt dus ook vanuit een ander punt geprojecteerd. Een ander neveneffect is dat de afstand van de projector tot aan het middelpunt van de spiegel geen constante is.

Nadat de beeldtransformatie uitgevoerd is, moet de projectiestraal zelf ook nog gericht worden om synchroon te blijven met het softwaremodel. Hiervoor volstaat een eenvoudige kalibratiemethode. Als we voor elk vlak in het model de projectie handmatig kalibreren in de vier hoeken (in de veronderstelling dat alle vlakken rechthoeken zijn), dan kunnen we via een interpolatietechniek de projectie richten voor tussenliggende punten.

We illustreren dit even aan de hand van een fictief voorbeeld. Stel dat we twee servomotors hebben die respectievelijk de pan- en tilthoek van de projectie controleren. Na een kalibratie hiervan uitgevoerd te hebben, zijn we tot de conclusie gekomen dat een invoerwaarde van 42 de projectie uiterst links (op -45 graden) zal plaatsen en een waarde van 78 de projectie uiterst rechts (op +45 graden). Voor de tilthoek vinden we gelijkaardige meetwaardes die de projectie uiterst naar boven en beneden richten, maar deze laten we voor de eenvoud even achterwege. Merk op dat de invoerwaarde van een servo ongeveer recht evenredig is met de rotatiehoek die deze maakt. Met deze gegevens en een model bestaande uit de dimensies van de kamer kunnen we voor elke x-coördinaat op dat oppervlak de panhoek berekenen die we nodig hebben om onze projectie op die plaats te verkrijgen. Hieruit kunnen we ook de waarde berekenen die de servo moet aannemen, zoals gedemonstreerd wordt in figuur 2.6. Als we dan van daaruit via een interpolatie de waarde berekenen die we

naar de servo moeten sturen, dan zal de projectie op de juiste plaats verschijnen.



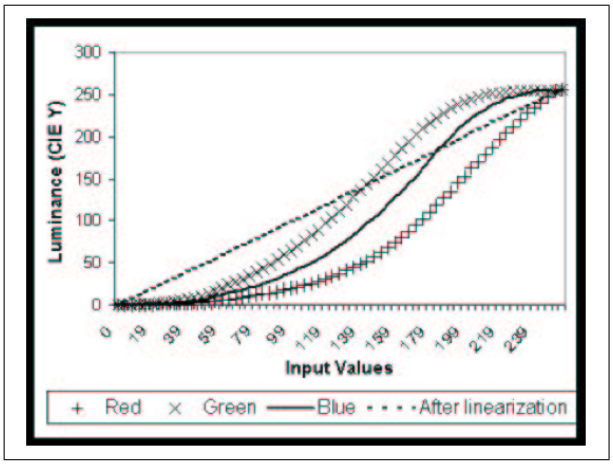
Figuur 2.6: Geometrische kalibratie

Hetzelfde kunnen we doen voor de tilthoek. Hier moeten we echter opmerken dat het aanpassen van de x-coördinaat niet altijd enkel de panhoek beïnvloedt, maar vaak ook in een aanpassing van de tilthoek zal resulteren. Het omgekeerde geldt ook voor de y-coördinaat.

2.6.2 Photometric calibration

Photometric calibration omvat het kalibreren van de kleuren en de helderheid. Bij een erg schuine projectie zullen de verst gelegen gedeeltes minder sterk belicht zijn dan de dichtbij gelegen gedeeltes doordat daar het licht over een groter oppervlak verspreid wordt. Als meerdere projectors in de opstelling aanwezig zijn en deze elkaars beeld moeten aanvullen, komen dit soort fouten duidelijker naar voor. Bovendien heeft niet elke projector een even goede kleurweergave; zo zal de tint puur rood (met RGB-waarde #ff0000) anders zijn bij andere projectors. Het geprojecteerde, rode licht bij een LCD-projector zal ook altijd nog sporen van groene en blauwe componenten bevatten.

De luminantie is niet lineair evenredig met de invoerwaarde. Dit wordt gecorrigeerd in de PixelFlex-applicatie [24]. Figuur 2.7 toont voor een bepaalde projector hoe de luminantie aangepast moet worden om een lineair verband te bekomen.



Figuur 2.7: Luminantie (bij een Proxima 6850-projector) in functie van de invoerwaarde voor de rode, groene en blauwe componenten. (Bron: [24].)

Kleurcorrectie laten we in deze thesis verder buiten beschouwing, net zoals het rekening houden met oppervlaktekenmerken zoals kleur, texturen en reflexie. We gaan ervan uit dat de muren wit zijn, volledig vlak, perfect glad, maar niet spiegelend en dus het licht in alle richtingen verstrooiend.

2.7 Software design

Veel verschillende papers maken gebruik van het VNC- of X11-protocol [24]. Hiermee kunnen traditionele desktopapplicaties het projectiesysteem gebruiken in ongewijzigde vorm. De X11-server wordt gewoonlijk op dezelfde computer uitgevoerd als die waar de applicaties draaien. Wanneer zo een applicatie start, verbindt deze zich met de X11-server om zijn grafische output door te geven. De gemodificeerde X11-server zal dan de juiste beeldtransformatie hierop toepassen en het resultaat doorsturen naar de 'echte' X11-server die met de grafische hardware is verbonden.

2.8 Privacy

Elke technologie die de activiteiten van gebruikers waarneemt en aan de hand daarvan acties onderneemt, creëert een vraag naar privacy. Niet iedereen vindt het even aangenaam als tientallen camera's al zijn bewegingen registreren, in het bijzonder niet als het onduidelijk is waarvoor de informatie wordt gebruikt (zoals bij een nieuwe technologie).

Wanneer een ED-projector (Everywhere Display projector) als monitor gebruikt wordt, zou het kunnen dat hierop persoonlijke informatie zoals e-mails of een kalender getoond wordt. Dit kan erg onaangenaam zijn als er toevallig een persoon binnenkomt. BlueSpace lost dit op door met het active badgesysteem te detecteren als andere personen binnenkomen en alle persoonlijke informatie onmiddellijk te verbergen [12].

Hoofdstuk 3

Vereisten voor het bouwen van een projectiesysteem

Ons doel is het realiseren van een opstelling die gelijkaardig is aan deze van IBM. Deze kan uitgebreid worden met elementen uit de andere papers ([22] en [7]) zoals het combineren van meerdere projectors.

3.1 Enkele aandachtspunten

Gezien de vooruitgang van hardware de laatste jaren zullen enkele van de beperkingen van projectors die nog een probleem vormden tijdens het onderzoek van IBM, nu gedeeltelijk opgeheven zijn. Zo hebben de huidige projectors een hoger contrast en een groter focusbereik. Moderne projectors hebben vaak ook een bredere projectiekegel, wat wel een voordeel is bij een traditionele set-up, maar in ons geval een nadeel. Er moet een klein stuk muur vanaf een grote afstand bereikt worden, en niet een groot stuk muur vanaf een kleine afstand.

Het realiseren van dit systeem bestaat uit verschillende onderdelen:

- Er moet hardware gevonden worden om de projectiestraal te kunnen richten. Er zijn weinig kant-en-klaarapparaten te koop die hiertoe in staat zijn en gelijkertijd voor een aanvaardbare prijs te verkrijgen zijn; dus zal er zelf iets

gebouwd moeten worden. De constructie van IBM kan hierbij een inspiratiebron zijn. In elk geval moeten zowel een pan- als tilthoek aanpast kunnen worden.

- Er is software nodig om dit systeem te besturen. IBM is vertrokken vanuit discohardware, vandaar dat dit bedrijf het DMX-protocol hanteert. Indien ervoor gekozen wordt om zelf een pan/tilt-'hoofd' te bouwen met servomotors, dan zijn de Phidgets [1] USB interface controllers handig om te gebruiken.

- De projector moet ook een beeld ontvangen dat geschikt is om te projecteren. Hiervoor zal ook nog software geprogrammeerd moeten worden om de beeldtransformatie uit te voeren.

- De twee vorige softwaresystemen moeten gekoppeld worden zodat het geprojecteerde beeld synchroon met de projectiehoek blijft. Verder moet nog een applicatie gebouwd worden die betekenis geeft aan onze opstelling, zodat deze nuttig wordt.

[1] Phidgets - http://www.phidgets.com/

Hoofdstuk 4

Architectuur voor een everywhere display projection

De oplossing zal gedeeltelijk uit hard- en gedeeltelijk uit software bestaan. De hardware proberen we zo goed mogelijk te abstraheren in software, zodat aanpassingen in de hardware onafhankelijk van de software doorgevoerd kunnen worden. In dit hoofdstuk zullen alle hard- en softwarecomponenten verder uitgediept worden.

4.1 Software

Tijdens het schrijven van de software moeten keuzes gemaakt worden over wat de beste volgorde van programmeren is. Na elke stap zou alles zo goed mogelijk getest moeten worden.

Oorspronkelijk zou eerst een eenvoudige applicatie gemaakt worden waarbij enkel met het toetsenbord de positie van een statische afbeelding over de muur verplaatst kan worden. Later zou er overgegaan worden naar dynamische applicaties en vervolgens op meerdere muren. Als laatste zou er aandacht besteed worden aan complexere interactietechnieken.

In werkelijkheid hebben we ons niet helemaal gehouden aan die planning. Eerst hebben we geëxperimenteerd met de technologieën die er bestaan om 3D rendering

engines te maken. Hierna kon al snel op een eenvoudig model overgegaan worden dat uit meerdere muren bestond, daarom is die stap meteen gemaakt. Vervolgens is de statische afbeelding uit de vorige stap vervangen door dynamische applicaties. Als één van de laatste stappen is pas de pan- en tiltoutput naar de servomotors gerealiseerd. Aangezien de hardware moeilijker is te debuggen dan software zou het in ons nadeel geweest zijn als we direct alles daarvan afhankelijk hadden gemaakt.

4.1.1 De rendering engine

De rendering engine is de centrale component die de beelden gaat genereren die naar de individuele projectors gestuurd moeten worden. De rendering engine gaat dus de beelden afkomstig van applicaties transformeren zodat deze na projectie equivalent zijn aan een loodrechte projectie.

Om deze beelden te genereren zal deze engine gebruik maken van een omgevingsmodel. Dit is een geometrisch model dat de posities bevat van alle beschikbare projectievlakken en de projectors.

Er zijn verschillende technologieën waarmee dit gerealiseerd kan worden, alle met hun sterke en zwakke punten.

Indien het vereist is dat bestaande applicaties in de ED-omgeving kunnen uitgevoerd worden, dan moet een protocol zoals X11 of VNC gebruikt worden.

X11 : Dit protocol is in de jaren '80 ontstaan om applicaties GUI-mogelijkheden te geven zonder dat deze direct de hardware moeten aanspreken. Hiervan bestaat een open implementatie (X.org [1]) en het protocol kan ook over een TCP-verbinding gebruikt worden. Op de computer waar het beeld getoond moet worden, wordt een X11-server gestart. De GUI-applicatie is dan zelf een client; die verbindt met de server en stuurt zijn GUI-output door.

VNC : (Virtual Network Computing) Bij dit protocol is de rol van client en server omgedraaid ten opzichte van X11. Aan één zijde is er een viewer – de client – die het beeld toont. Aan de andere kant, gewoonlijk ook op een andere computer, is er een server die de beeldoutput van daar door naar de client

doorstuurt. De VNC-server kan gekoppeld zijn aan een X11-sessie op die computer.

De rendering engine kan dus gebouwd worden als een X11-server of als VNC client. Deze engine zal dan het binnenkomend beeld de juiste transformatie laten ondergaan en vervolgens naar de juiste projector sturen. Die transformatie kan uitgevoerd worden via de grafische versnelling van de GPU (grafische kaart), maar de applicatie zelf kan niet eenvoudig van hardwarematige beeldversnelling gebruik maken. Een voorbeeld waar VNC wordt gebruikt is [24].

Een andere optie is het ontwerpen van een eigen eenvoudig 'protocol' waarmee de videoserver aangesproken kan worden. Wij hebben gekozen om deze rendering engine in C# te schrijven, omdat met het .NET framework waar deze taal deel van uitmaakt snel een erg goed resultaat bekomen kan worden. Oorspronkelijk waren we van plan Managed DirectX te gebruiken, maar later ontdekten we dat deze bibliotheek als verouderd wordt beschouwd door het ontstaan van Windows Presentation Foundation (WPF) en XNA in versie 3.0 van het .NET framework. Zowel met WPF als met XNA kunnen grafisch rijke applicaties ontworpen worden. WPF is bedoeld voor dekstop- en browserapplicaties, terwijl XNA eerder bedoeld is voor computergames.

We gebruiken nu WPF en elke projector zal hierin zijn eigen beeld renderen met een *Viewport3D* en een *PerspectiveCamera*. De applicatievensters – welke simpelweg een WPF *UserControl* zijn – kunnen eenvoudig in een Viewport3D geplaatst worden. Een nadeel dat hieruit volgt, is dat alle 'applicaties' in hetzelfde proces uitgevoerd worden. Dat is niet eenvoudig binnen WPF op te lossen aangezien er verwacht wordt dat de volledige GUI in dezelfde thread uitgevoerd wordt.

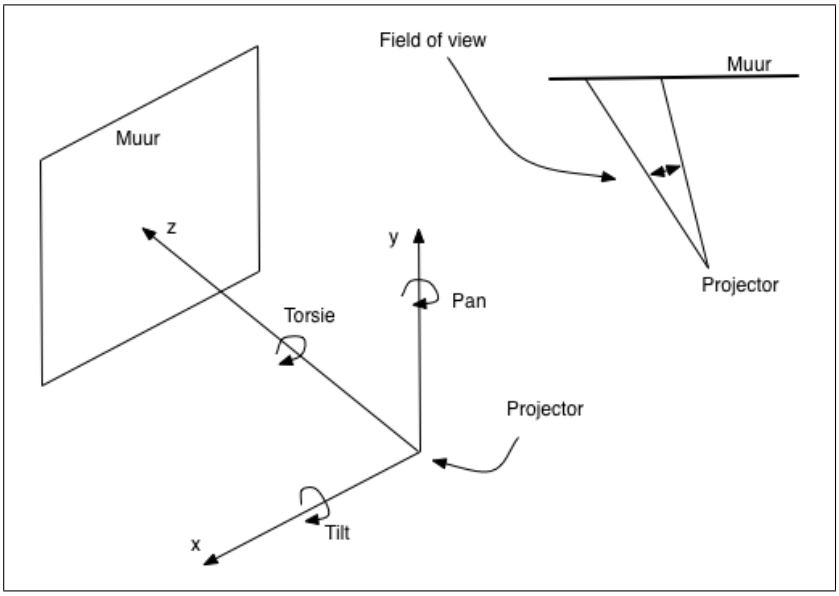
4.1.2 Toepassen van de beeldtransformatie

De rendering engine gaat het beeld moeten draaien volgens de assen die overeenkomen met de assen waarrond later mechanisch gezien ook de projector zal draaien.

Het eerste prototype (Zie sectie 4.2.3) is een fixed centerprojectie zoals [20]; dat maakt het eenvoudiger om beelden te transformeren. (Het tweede prototype uit

sectie 4.2.3 is geen perfecte fixed centerprojectie, maar zou dit zo naukeurig mogelijk moeten benaderen.)

De transformatie bestaat uit drie rotatiecomponenten. Twee componenten compenseren respectievelijk de pan- en de tilthoek. Een derde rotatiecomponent zorgt voor een rotatie rond de projectiestraal. Dit is noodzakelijk voor de gevallen waar we langs boven of onder op de spiegel projecteren. Het beeld zal hier ronddraaien als de panhoek wijzigt. Bij een toename van de panhoek van 180 graden, zou het beeld in dit geval ondersteboven geprojecteerd worden als we die laatste torsierotatie zouden weglaten. Figuur 4.1 toont al deze rotatiecomponenten.



Figuur 4.1: Componenten van een beeldtransformatie

Van deze transformatieapplicatie hebben we eerst een prototype gebouwd (met behulp van managed DirectX). Het model bestond op dit moment enkel uit een statische afbeelding en een projector. De applicatie bevat vijf variabelen, namelijk:

- de rotatie rond de Y-as, die de panhoek moet compenseren;
- de rotatie rond de X-as, die de tilthoek moet compenseren;
- een rotatie rond de Z-as, die de torsie compenseert en vanuit de panhoek berekend kan worden;

- de afstand van de projector tot aan het projectiecentrum op het projectieoppervlak; en
- de verticale hoek die de projectiekegel maakt (field of view).

De gebruikte transformatie is de volgende:

Listing 4.1: Berekenen van de beeldtransformatie

```
device.Transform.World = Microsoft.DirectX.Matrix.Identity *
   pan * tilt * scale * rotation;
```

De variabelen *pan*, *tilt* en *rotation* zijn de drie rotatiematrices die noodzakelijk zijn voor de projectie. Het zicht op de wereld moet bovendien een perspectiefprojectie en geen ortogonale projectie zijn.

Figuur 4.2: Prototype van de beeldtransformatie, geïmplementeerd in C#

4.1.3 Applicaties

Applicaties die gebruik willen maken van de projectie moeten aan een vastgelegde API voldoen zodat de rest van het systeem deze applicaties kan herkennen en moei-

teloos kan integreren. Gewoonlijk zal elke applicatie een aparte bibliotheek (DLL) vormen.

Elke applicatie erft van de class *EverywhereApplication*. In de constructor worden alle belangrijke parameters meegegeven die toegang verschaffen tot het omgevingsmodel. Hiermee kan een applicatie de omgeving ondervragen en in beperkte mate ook wijzigen. Een applicatie kan vensters en inputmodules registreren onder een bepaalde naam.

Vensters : Elk venster is een WPF UserControl. Dit heeft als groot voordeel dat bestaande WPF-applicaties weinig aangepast moeten worden om geïntegreerd te worden in de projectie. Bovendien beschikt elk venster nog over 3D-acceleratie indien dit vereist is. Dat laatste zal hier performanter zijn dan als we een protocol zouden gebruiken waar elk venster eerst volledig softwarematig naar een bitmap wordt gerenderd. Figuur 4.7 toont een applicatie die videobeelden projecteert op een muur.

Input modules : Deze zijn een abstractie voor alle mogelijke input devices. Vermits de renderer zijn werk doet op een centrale computer die deel uitmaakt van een groter – vaak gedistribueerd – systeem, zullen input devices vaak via allerhande netwerkprotocols verbonden worden. Elk device dat in de projectie gebruikt moet worden, zal gewrapt moeten worden door de class EverywhereInput.

Een applicatie kan ook twee vormen van output bevatten:

Beeld-output : De beelden moeten ook op één of andere manier naar de grafische kaart gestuurd worden die aan de projector is gekoppeld. Applicaties kunnen een instantie van de rendering engine, die gekoppeld is aan een projector in de vorm van een *UserControl*, opvragen. Deze kunnen ze dan zelf in een echt venster plaatsen dat op de juiste projector wordt getoond. We hebben dus een applicatie nodig die deze taak op zich neemt.

Pan- en tiltoutput : Voor elke stuurbare projector moeten deze hoeken synchroon lopen met het beeld dat ze ontvangen. Dit zal waarschijnlijk niet altijd via

hetzelfde protocol gebeuren, daarom moet de applicatie hier ook hardware abstraheren.

Deze outputinformatie kan dus door applicaties rechtstreeks uit het model opgevraagd worden.

4.1.4 De verschillende componenten

Het componentendiagram in figuur 4.3 toont de datastroom tussen alle belangrijke componenten – zowel software als hardware – die zich in ons framework bevindt. Dit schema is zeker niet compleet maar toont wel de belangrijkste componenten en hun relaties.

De hardware bestaat uit:

- de servomotors en hun controllerboard;
- een beamer; en
- invoerhardware zoals een keyboard.

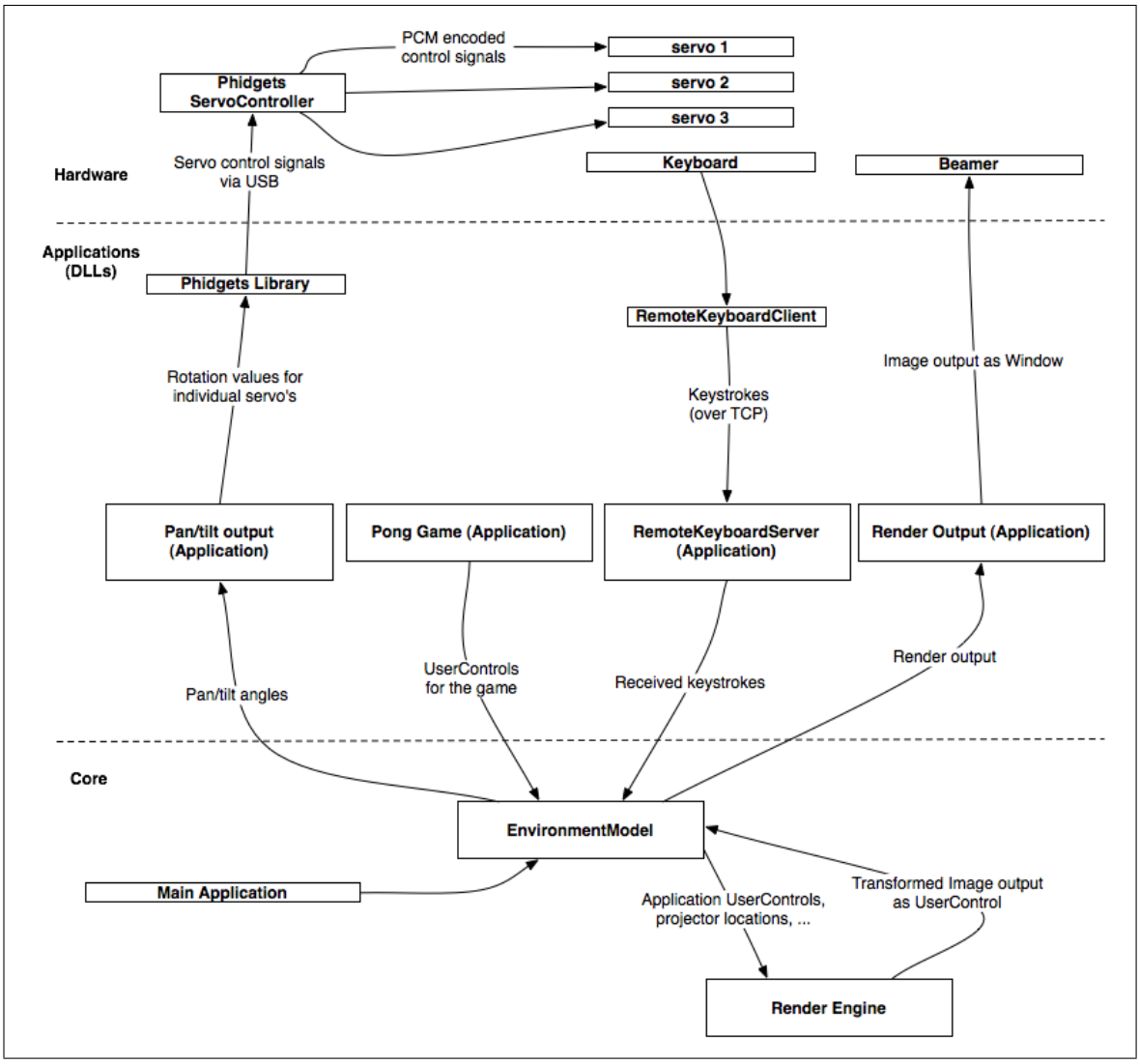
Al deze hardwarecomponenten worden geabstraheerd door applicaties.

In dit schema zijn vier applicaties geplaatst, maar vaak zullen er veel meer actief zijn.

Pan- en tiltoutput : Deze applicatie leest de pan- en tilthoeken die een projector aanneemt uit het model en vertaalt deze naar de draaihoeken welke de servo's moeten aannemen. Het resultaat wordt naar de Phidgets servobibliotheek gestuurd.

Pong-spel : Dit is een voorbeeldtoepassing die enkele *UserControl*s zoals een balletje en scorebord toevoegt aan het model.

Remote keyboard : Dit is een voorbeeld van een invoerapplicatie die de keyboardhardware abstraheert.



Figuur 4.3: Componentendiagram

Render output : Deze applicatie vraagt het resultaat van een instantie van een rendering engine aan. Het ontvangt deze in de vorm van een *UserControl* en plaatst dat control vervolgens in een venster dat op de projector wordt weergegeven.

De 'core' van het framework bevat het omgevingsmodel dat alles aan elkaar koppelt en de Render Engine die de beelduitvoer genereert.

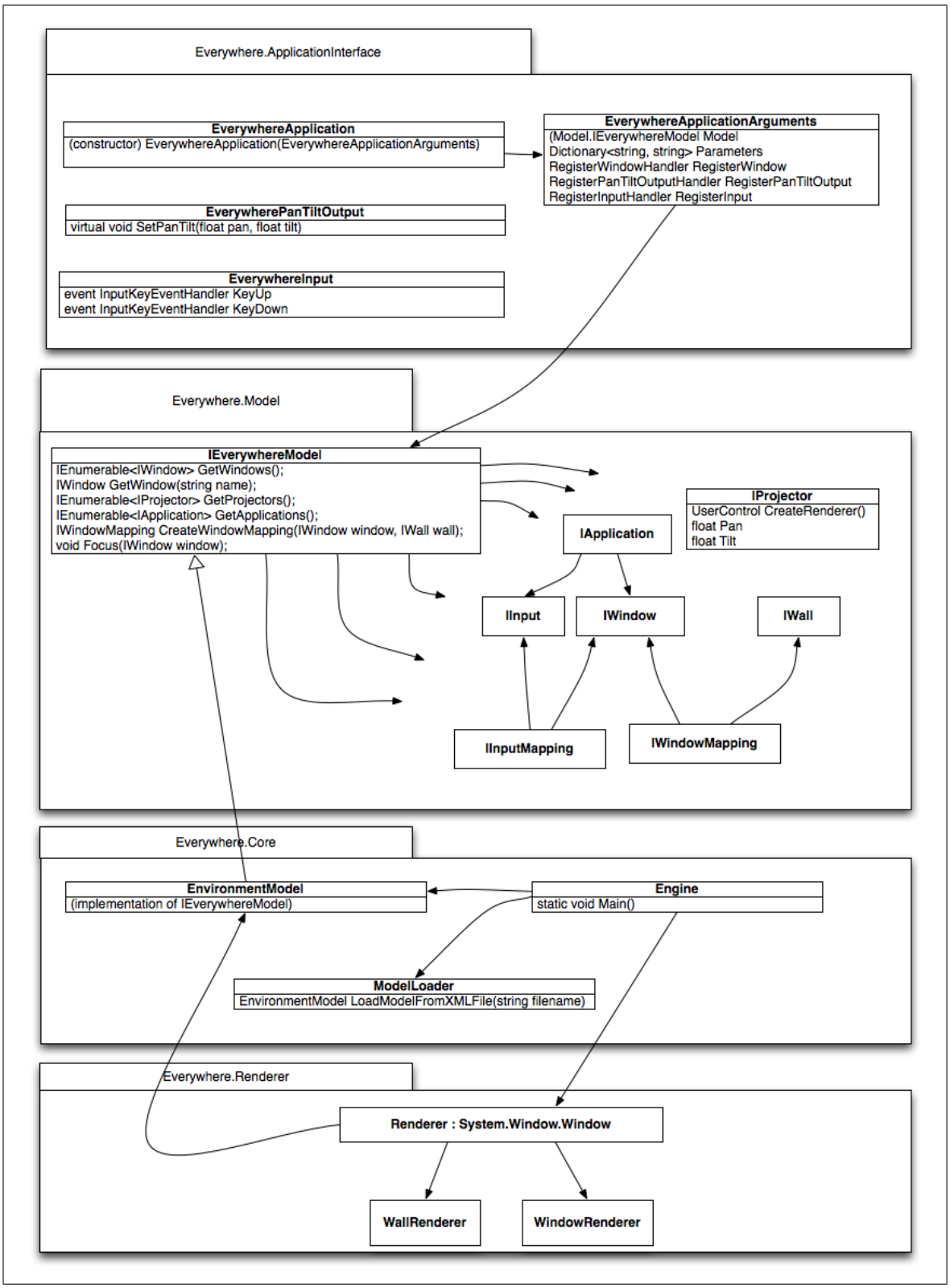
Figuur 4.4 toont de belangrijkste klassen uit het framework. (Ook hier zijn details achterwege gelaten.) De vier namespaces zijn:

Model : Deze namespace bevat een abstracte interface naar alle componenten (input, vensters, projectors, muren...) in de opstelling en hun onderlinge relatie. Een *InputMapping* koppelt de invoer of een invoermodule aan een applicatievenster. Alle classes hierbinnen zijn in werkelijkheid *interfaces* en hebben dus geen implementatie. Hierdoor is het mogelijk om het model volledig los te koppelen van de rendering engine.

ApplicationInterface : Deze namespace bevat enkele abstracte classes waarop applicaties gebouwd zijn. Een applicatie ontvangt een *EverywhereApplicationArguments* in zijn constructor waarmee deze tijdens de loop van het programma dynamisch vensters en inputmodules kan toevoegen aan het model.

Core : Hier start het programma op. De *ModelLoader* zorgt ervoor dat alle applicaties die in de configuratie gedefinieerd staan, direct na het opstarten worden geladen. Deze namespace bevat de 'engine'. Hier is ook voor elke class uit het model een overeenkomstige class die alle methodes implementeert. Als een applicatie denkt te spreken met *Everywhere.Model*, spreekt hij dus in werkelijkheid met *Everywhere.Core*.

Renderer : De rendering engine staat los van de rest, zodat ook dit gedeelte eenvoudig vervangen kan worden. Hier worden de beelden gegenereerd voor elke projector.



Figuur 4.4: UML-schema

4.1.5 Configuratie van de ED-projectie

De ED-projectie kan eenvoudig geconfigureerd worden met behulp van een enkel XML-bestand. We zullen hieronder beknopt de verschillende secties bespreken.

Code listing 4.2 toont een *muur*-sectie. Deze bevat de coördinaten van de linkeronderhoek en twee vectoren die het vlak beschrijven, relatief ten opzichte van de linkeronderhoek. Verder worden de connecties tussen de verschillende muren gedefinieerd. Een venster dat bijvoorbeeld naar rechts beweegt als het op 'frontwall' aanwezig is, zal terecht komen op 'rightwall'.

Listing 4.2: Configuratie van een projectievlak (muur)

```
<wall name="frontwall">
   <left-bottom x="-10" y="10" z="-5" />
   <right-vector x="20" y="0" z="0" />
   <up-vector x="0" y="0" z="10" />
   <transitions right="rightwall" left="leftwall" />
   <background color="#ffffff" />
</wall>
```

Elke zinnige opstelling bevat ook minstens één projector, bijvoorbeeld zoals in code listing 4.3. Het gezichtsveld wordt uitgedrukt in graden en toont de hoek die de projectiekegel maakt. Aangezien het beeld dat deze renderer genereert gewoonlijk gekoppeld is aan een fysieke projector met een beperkt pan- en tiltbereik, kunnen we die grenzen hier ook definiëren. Verder worden de positie en vectoren die de boven- en kijkrichting aangeven, meegegeven. (De pan- en tiltwaarde wordt relatief ten opzichte van deze laatste vectoren berekend.)

Listing 4.3: Configuratie van een projector

```
<projector name="projector1" show-walls="false">
   <field-of-view>25</field-of-view>
   <range minpan="-180" maxpan="180" mintilt="-45" maxtilt="45" />
   <angle-update-interval>0</angle-update-interval>
   <position x="0" y="0" z="0" />
   <up-vector x="0" y="0" z="1" />
   <look-vector x="0" y="1" z="0" />
```

```
</projector>
```

In listing 4.4 wordt een applicatie in de projectie geladen. Er wordt verwezen naar de assembly en de applicatieclass hierbinnen. De parameters die meegegeven worden, zijn afhankelijk van de specifieke applicatie. Tijdens de initialisatie van een applicatie kunnen vensters geregistreerd worden. Listing 4.5 toont vervolgens hoe het venster met de naam 'main' van de applicatie 'klokapp' op een bepaalde plaats op de muur 'frontwall' wordt gemapt. De 'floating'-optie zorgt ervoor dat vensters voor de muur zweven op een bepaalde afstand. Hiermee kunnen we dus eenvoudig de ordening van verschillende vensters forceren (de z-index).

Listing 4.4: Configuratie van een applicatie

```
<application name="klokapp">
   <assembly>...\...\EverywhereExampleApplications.dll</assembly>
   <class>EverywhereExampleApplications.Clock</class>
   <param name="TextColor" value="blue"/>
   <param name="BackColor" value="red" />
</application>
```

Listing 4.5: Configuratie van een window mapping

```
<window-mapping window="klokapp:main" wall="frontwall">
   <position x="2" y="2" />
   <size width="6" height="6" />
   <floating distance="0.1" />
</window-mapping>
```

Listing 4.6 laadt nog een applicatie, maar deze bevat enkel een inputgedeelte. 'RemoteKeyboard' luistert in dit geval op poort 8002 naar inkomende TCP-connecties van de bijbehorende client. Die client (zie figuur 4.8 verderop) kan zich dus op elke computer in het netwerk bevinden en zal de toetsaanslagen van dat toetsenbord volgens de hier gedefinieerde mapping sturen naar het venster 'main' van de applicatie 'klokapp'.

Listing 4.6: Configuratie van een invoerapparaat

```
<application name="remote-keyboard">
```

```
   <assembly>...\...\RemoteKeyboardServer.dll</assembly>
   <class>RemoteKeyboard.RemoteKeyboard</class>
   <param name="port" value="8002" />
</application>
<input-mapping input="remote-keyboard:keyboard"
     receiver="klokapp:main" />
```

Het XML-configuratiebestand bevat alle componenten die gedurende het starten moeten geladen worden, maar applicaties kunnen dit model at runtime volgens hun wensen aanpassen.

Wanneer later het framework uitgebreid moet worden, is het ook de bedoeling om zoveel mogelijk extra functionaliteit toe te voegen in de vorm van applicaties. Hiermee kunnen de kern en renderer van het systeem proper gehouden worden. Bijvoorbeeld: stel dat we bij een specifieke opstelling twee beamers hebben die altijd synchroon moeten bewegen. (Dat kan noodzakelijk zijn omwille van mechanische redenen of juist omdat het voor die toepassing vereist is.) Dat is duidelijk functionaliteit die in de meeste opstellingen niet vereist is, dit kan dus beter als applicatie geïmplementeerd worden.

4.1.6 Voorbeelden van applicaties

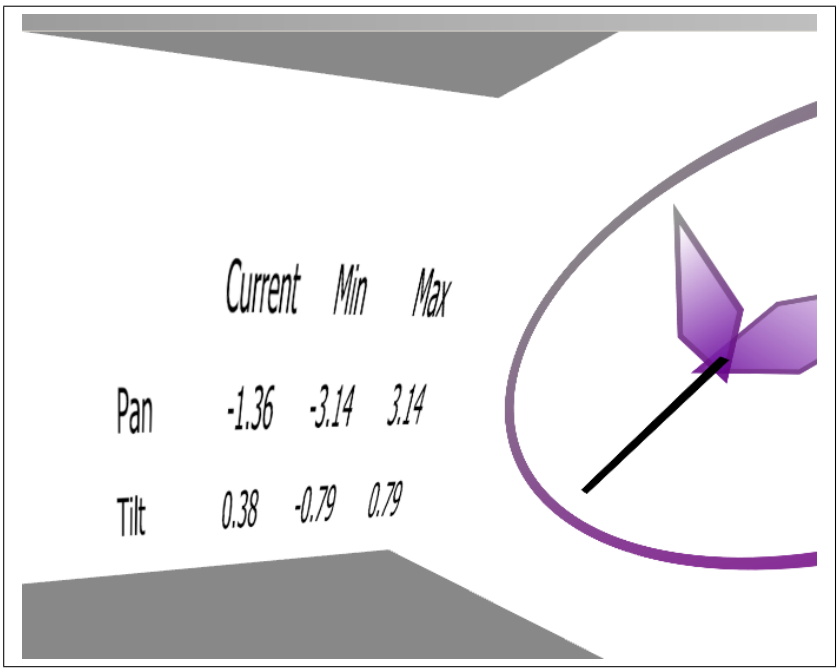
Hier bespreken we enkele voorbeeldapplicaties om de mogelijkheden van ons framework te demonstreren.

Tonen van de pan- en tilthoeken in de ED

Deze applicatie (Figuur 4.5) toont in een venster op een muur in realtime de status van een projector. Applicaties kunnen dus informatie uit het model opvragen.

Pong

Om de inputmogelijkheden te demonstreren, hebben we ook nog het Pong-spel geïmplementeerd. Deze applicatie verwacht als parameter de naam van een muur en zal



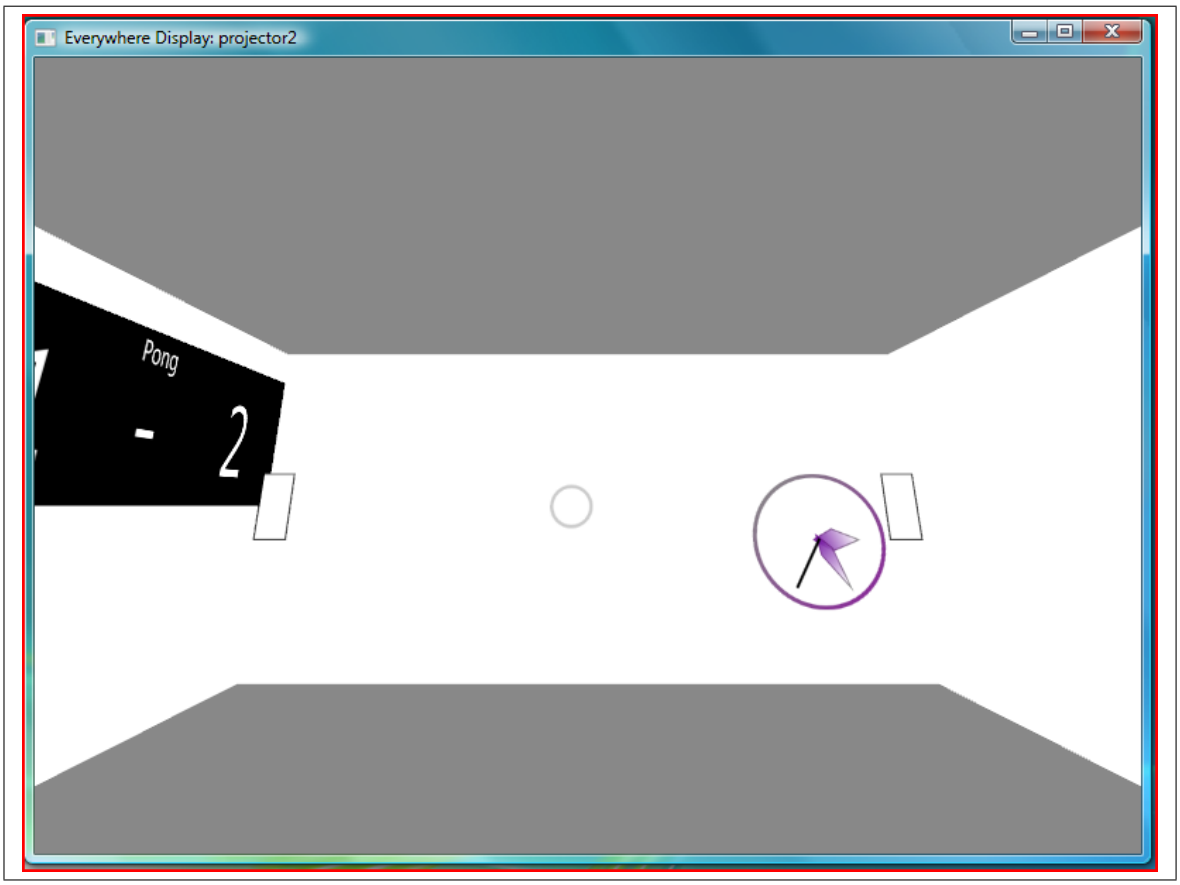
Figuur 4.5: Tonen van pan- en tilthoek van een bepaalde projector op een muur van de ED-projectie.

tijdens het opstarten zelf alle window-mappings die nodig zijn, maken op deze muur. Vanzelfsprekend kan zo ook de positie van het balletje aangepast worden door de applicatie zelf.

Deze applicatie kan zo geconfigureerd worden dat een aparte projector met kleine projectiekegel continu het balletje volgt. Als de hardware snel genoeg is, wordt dus een vorm van foveal-projectie bekomen waarbij het balletje in hogere resolutie (dots per inch) wordt getoond.

Afspelen van video

De performantie is erg goed. Op een recente computer kan high definition (720p) video afgespeeld worden op een muur van de projectie.



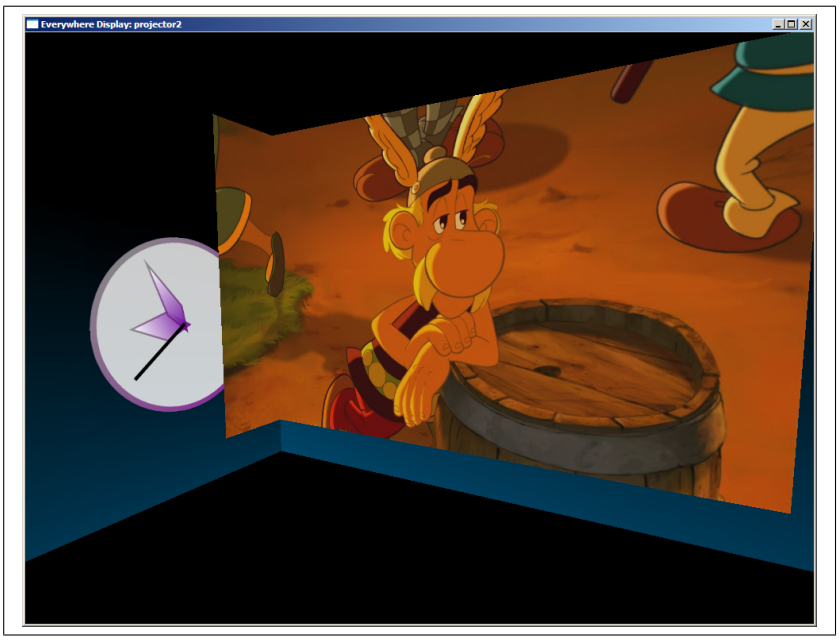
Figuur 4.6: Pong

Taakbeheer

Een taakbeheerapplicatie zou een lijst moeten kunnen opvragen van alle andere applicaties die op dat moment actief zijn, en de mogelijkheid bieden om deze af te sluiten. Ook zouden hiermee nieuwe applicaties at runtime gestart moeten kunnen worden. Tot nu toe is deze applicatie nog niet geïmplementeerd, maar het behoort zeker tot de mogelijkheden.

Remote keyboard

Deze applicatie heeft enkel een inputgedeelte, maar wordt gekoppeld aan de 'remote keyboard client' die wel een GUI heeft. De ED-applicatie bevat een server die luistert



Figuur 4.7: Een videofilm die afspeelt op een muur van de ED

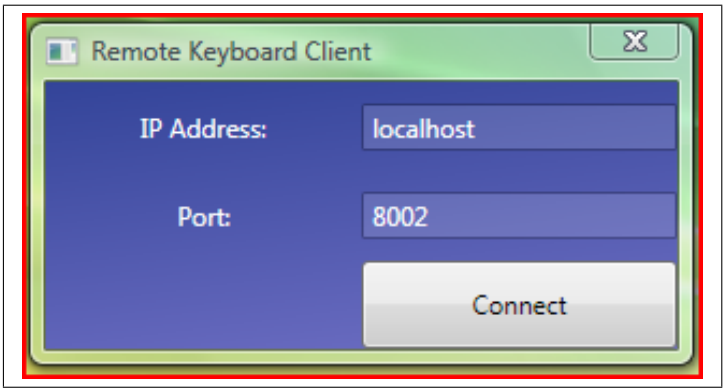
naar een bepaalde TCP-poort. De client die te zien is in figuur 4.8 en vaak op een ander apparaat wordt uitgevoerd, kan hiermee verbinden en de toetsaanslagen die hij ontvangt over het netwerk sturen naar een venster van de ED.

Renderen naar ED Window

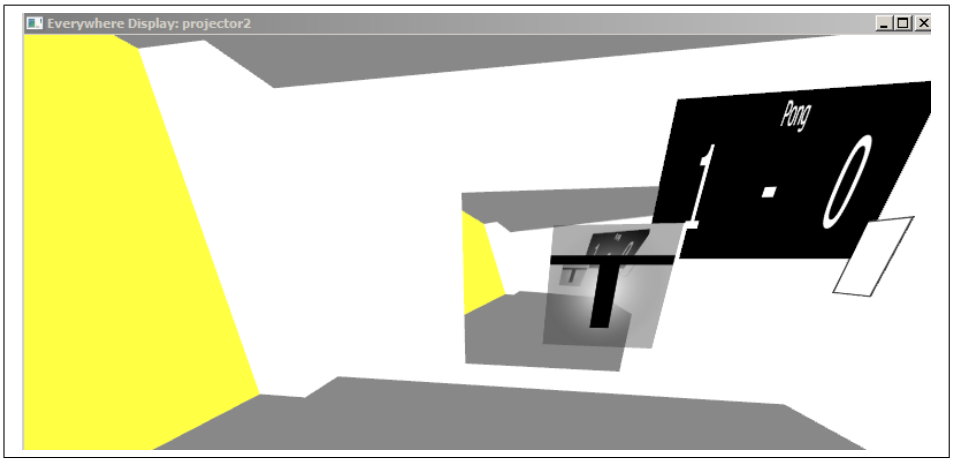
Deze applicatie is totaal nutteloos, maar demonstreert de kracht van het framework (en eigenlijk ook de mogelijkheden van WPF waarop dit allemaal is gebouwd). Hier wordt het beeld van een bepaalde renderer (die het beeld van een projector genereert) terug op een muur geplaatst. Dit veroorzaakt de recursieve beelden van figuur 4.9.

Window manager

Wanneer gebruikers zelf de mogelijkheid hebben om applicaties aan de ED-omgeving toe te voegen en hierbinnen te verplaatsen, dan gaat er een nood ontstaan om deze efficiënt te beheren. We gaan dus een soort van window manager nodig hebben.



Figuur 4.8: Remote keyboard client. De keyboard input van dit venster wordt over het netwerk verstuurd naar een bepaald venster in de projectie volgens een input mapping.



Figuur 4.9: Het renderen van output naar een scherm op een muur

Deze zou dan het verplaatsen van applicaties en het overzichtelijk houden hiervan op zich kunnen nemen.

Momenteel zit deze functionaliteit nog niet in ons framework geïntegreerd, maar misschien dat dit in de toekomst noodzakelijk gaat zijn. De implementatie kan gebruik maken van het decorator pattern dat elk window wrapt. Er mag bovendien maar één instantie van een window manager aanwezig zijn. Het porten van een bestaande window manager heeft in dit geval weinig nut, omdat deze window managers niet geoptimaliseerd zijn voor een everywhere display environment, waar we andere noden hebben dan bij een desktopsysteem.

4.1.7 Ontwerpen van een Hello World-applicatie

Hieronder demonstreren we beknopt de stappen die nodig zijn om een eigen 'Hello World'-applicatie te maken.

Deze applicatie kunnen we in een nieuw DLL-project plaatsen waarbij een referentie naar *ApplicationInterface* wordt toegevoegd. Nu maken we een nieuwe XAML-user-control door de code uit listing 4.7 in *HelloWorld.xaml* te plaatsen.

Listing 4.7: XAML code voor een Hello World-applicatie

```
<UserControl x:Class="HelloWorld.HelloWorldWindow"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Height="300" Width="300">
   <TextBlock TextAlignment="Center"
         VerticalAlignment="Center" FontSize="72">
      Hello world
   </TextBlock>
</UserControl>
```

In de bijbehorende code file *HelloWorld.xaml.cs* zullen we dan de rest van deze 'partial class' plaatsen. (Zie listing 4.8.) Om van deze usercontrol een bruikbare applicatie te maken, hebben we nog een EverywhereApplication nodig. We erven deze laatste class over en in zijn constructor wordt de instantie van dat venster aangemaakt.

Listing 4.8: C# code voor een Hello World-applicatie

```
namespace HelloWorld
{
   public partial class HelloWorldWindow: UserControl
   {
      public ApplicationManagement()
      {
         InitializeComponent();
      }
   }
```

```
   public class HelloWorld: EverywhereApplication
   {
      public HelloWorld(EverywhereApplicationArguments arguments) :
            base(arguments)
      {
         RegisterWindow("helloworld", new HelloWorldWindow());
      }
   }
}
```

4.1.8 Meerdere projectiesystemen combineren

Oorspronkelijk was er het plan eerst een systeem te bouwen dat met een enkele projector werkt en dit later uit te breiden tot meerdere projectors. Door de structuur van het hele framework hebben we echter al vanaf het begin meerdere projectors ondersteund. Er moet nu simpelweg een tweede instantie gemaakt worden van de rendering engine waarbij een referentie naar een andere projector meegegeven wordt.

Zolang deze projectors elkaar niet overlappen, is dit eenvoudig te realiseren. Wanneer we twee of meerdere projectors op hetzelfde vlak willen richten met de bedoeling een groter projectiebeeld te verkrijgen, zullen deze beelden gestitcht moeten worden zodat er geen zichtbare naden ontstaan.

Dit laatste – het combineren van projectiesystemen als alternatief voor een krachtigere, hogeresolutieprojector – is helemaal niet vanzelfsprekend. In het onderzoek dat we eerder hebben gedaan, zijn vaker projectors gecombineerd, maar telkens ging het om statische projectors [18] [7] [6] [24].

Een uitzondering hierop is het ‘foveal projection display’ [2]. Hier wordt een hoge resolutie-overlay (de ‘foveal inset’) bovenop een laag resolutiescherm geprojecteerd. De locatie van de foveal inset kan met een laserpointer verplaatst worden. Hier zien we dus een combinatie van een stuurbare projectie en een vaste projectie. Toch zijn ook hier beperkingen:

- Door de combinatie van de twee resoluties is het voldoende als de foveal inset een nauwkeurigheid heeft van het lage resolutiescherm. Wanneer twee projectiebeelden van dezelfde resolutie samengevoegd moeten worden, dan zullen deze een nauwkeurigheid moeten hebben tot op pixelgrootte.
- Bij het foveal display uit [2] staat één van de twee projecties vast. Wanneer we beide projecties stuurbaar maken, dan kan de afwijking van deze twee beelden ten opzichte van elkaar dubbel zo groot zijn. Bijgevolg moeten we dan ook dubbel zo nauwkeurig kunnen projecteren om twee stuurbare projecties te stitchen.

Meerdere stuurbare projecties combineren in één grotere projectie vereist dus een zeer nauwkeurige kalibratie, die – voor zover bekend – tot nu toe nog nooit is gerealiseerd. Een perfecte kalibratie kunnen we bovendien niet realiseren met de huidige hardware. Een servomotor geeft nooit feedback – deze streeft naar de positie die we opgeven – maar we kunnen niet zeker zijn dat deze ook bereikt wordt. Tenzij we met een camera de projectie waarnemen en met beeldbewerkingstechnieken daaruit de huidige positie kunnen afleiden.

Het combineren van meerdere projectors om onafhankelijke projecties te bekomen, is geen probleem.

4.2 Hardware

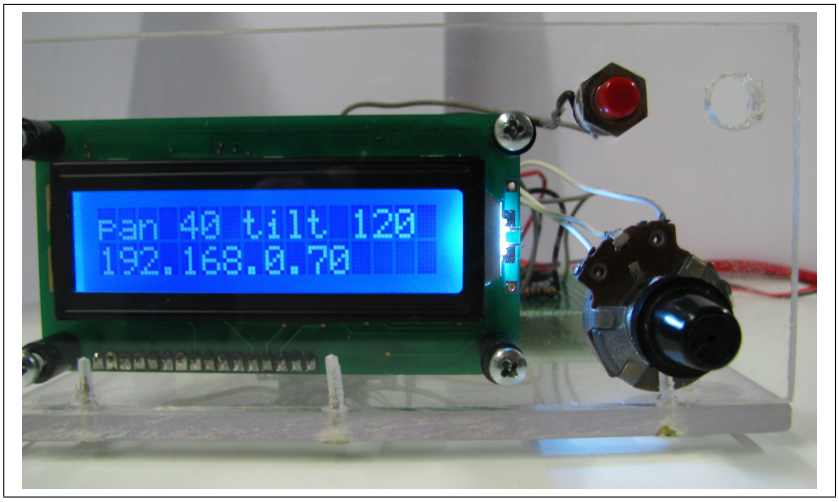
De ED-engine is zo ontworpen dat de hardware volledig geabstraheerd wordt, zowel de pan- en tiltoutput als input worden door een aparte applicatie afgehandeld. Het is dan ook aan deze applicatie om de hardware interface om te zetten naar deze generieke interface van de ED.

4.2.1 LCD output

Om deze hardware-abstractie aan te tonen, hebben we een eenvoudig LCD-schermpje gemaakt dat de pan- en tilthoek weergeeft zoals in figuur 4.10 te zien is. Deze

hardware kan bestaan uit een ethernet chip (enc26j60), atmel AVR microprocessor (Atmega186) en LCD-scherm (HD44780 compatibel).

De applicatie die deze hardware koppelt aan de ED-engine stuurt bij elke wijziging van de pan- of tilthoek een UDP-pakket naar deze microprocessor met de nieuwe waardes. Deze worden vervolgens getoond op het LCD-schermpje. De microprocessor zou natuurlijk net zo goed een servomotor kunnen besturen in plaats van het resultaat weer te geven.



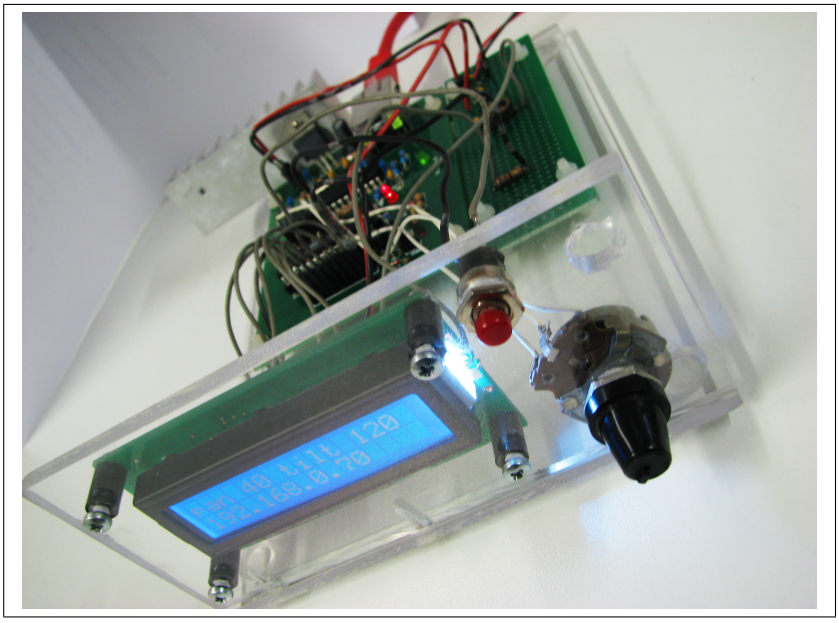
Figuur 4.10: Pan- en tiltoutput op een LCD-scherm

4.2.2 Button input

Later hebben we naast datzelfde LCD-schermpje ook een draaiknop en drukknop geplaatst met de bedoeling ook de invoer te abstraheren. (Zie ook figuur 4.10.) De draaiknop is een potentiometer (met een weerstand van 10 kΩ) die de spanning tussen de grond (0V) en de +3V interpoleert afhankelijk van zijn positie. Een analoog-digitaal-converter (ADC) in de Atmega168 processor zet deze waarde om in een 8 of 10 bits integer. De drukknop is gekoppeld aan een connectie die als I/O-ingang werd geprogrammeerd. (Deze pin hangt met een weerstand – die ze de pull-up weerstand noemen – aan de +3V. Wanneer de knop – die aan de andere zijde aan de grond gekoppeld is – wordt ingedrukt, daalt de spanning op die I/O-pin

onmiddellijk tot 0V met als gevolg dat de interrupt code die hieraan gekoppeld werd, wordt uitgevoerd.)

De ED-applicatie die hiermee communiceert, kan de toestanden van deze knoppen uitlezen en hieraan de gewenste acties koppelen. Wij kozen ervoor de draaiknop te koppelen aan de panhoek zodat de hele projectie meedraait als aan deze knop wordt gedraaid. We hadden ook de draaiknop kunnen gebruiken in een *EverywhereInput* om de 'up' en 'down' events te genereren die het blokje in het Pong-spel besturen.



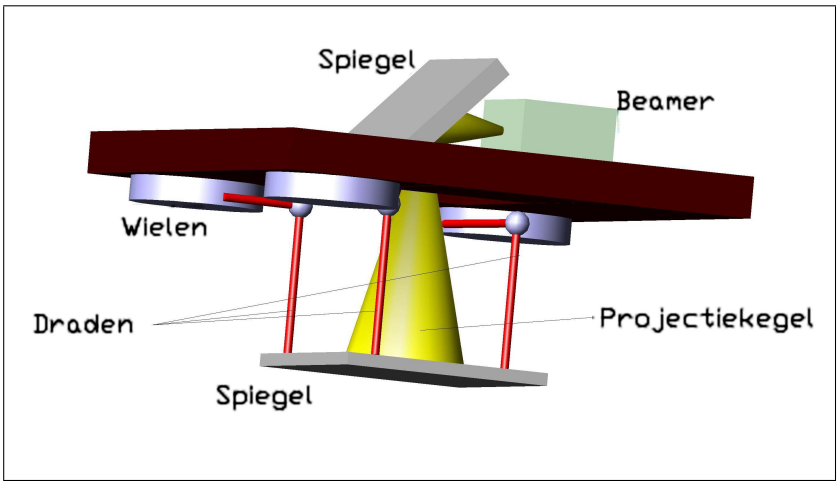
Figuur 4.11: Bovenaanzicht van de output/inputmodule

4.2.3 De mechanische besturing van de projector

Eerste prototype

Het eerste protoype dat we maakten, is geïnspireerd door de werking van marionetten. We hebben dus een andere aansturingsmethode dan de pan-/tiltdiscospiegel van IBM geëvalueerd. De spiegel hangen we op aan drie draden die aan de spiegel zelf in een gelijkzijdige driehoek bevestigd zijn. Door de lengte van de draden te laten variëren, is het mogelijk om de spiegel een willekeurige pan- en tilthoek te laten

aannemen. Het aanpassen van deze touwlengtes zal gebeuren door de touwtjes zelf te laten oprollen op een wiel dat bestuurd wordt door een servomotor. Het concept van deze besturing volgens ons doel zou nog niet eerder uitgetest zijn, daarom waren we erg benieuwd naar de resultaten.



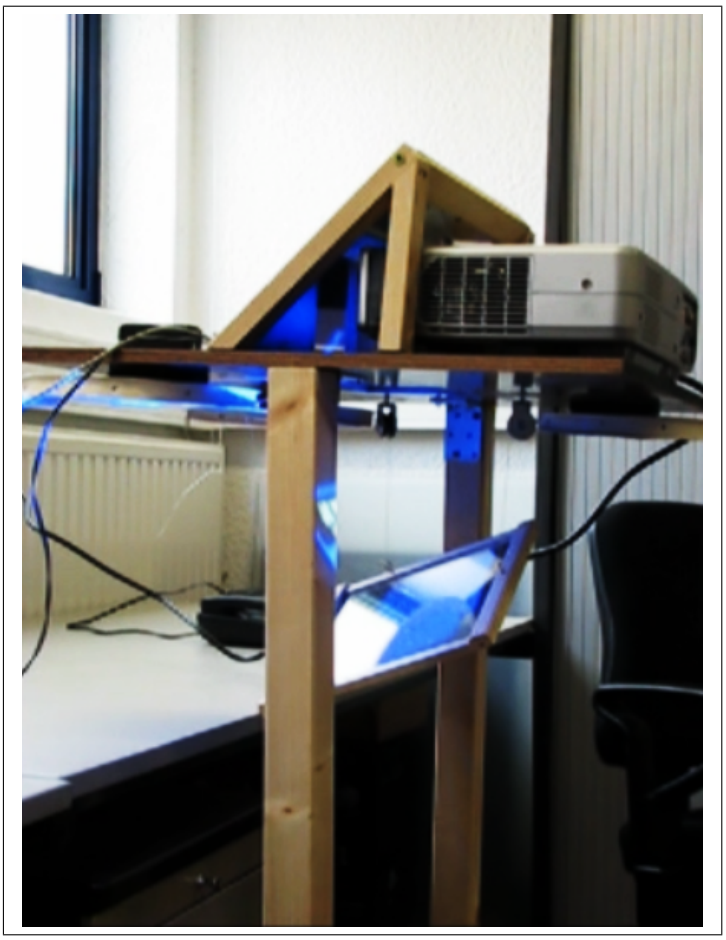
Figuur 4.12: Tekening van het eerste prototype

Het resultaat was echter niet zo veelbelovend. Een groot probleem waarmee we geconfronteerd werden, zijn schommelingen van de spiegel. Deze traden op na elke aanpassing van de pan- of tilthoek. We gingen er ook van uit dat de drie servo's perfect identiek zijn en onder dezelfde invoerwaarde ook dezelfde hoek aannemen. Bij benadering klopt dat ook, maar er zitten soms kleine afwijkingen in.

Om de opstelling te testen, hebben we een kleine C#-applicatie gemaakt die de tilthoek constant houdt en de pan langzaam in het interval [0-2×π] laat variëren. In theorie zou dat in een cilindervormige kamer het projectiebeeld op ongeveer dezelfde hoogte moeten houden. [1] Onze testkamer was – zoals de meeste ruimtes – kubusvormig waardoor op de verst gelegen plaatsen het projectiebeeld altijd een stuk hoger kwam te liggen. (Om technische redenen kon nog niet perfect horizontaal geprojecteerd worden.)

Een ander probleem had betrekking op de gebruikte projector. Deze heeft een erg brede projectiekegel waardoor het beeld gedeeltelijk langs de spiegel belandde bij het selecteren van grotere projectiehoeken. Verder leek de spiegel erg onnauwkeurig

[1] Tenminste, wanneer de projector in het centrum staat



Figuur 4.13: Prototype in actie

onder grotere tilthoeken die noodzakelijk zijn om de muur te bereiken – we willen immers niet op het plafond projecteren.

Eén van de drie gebruikte servomotors had regelmatig last met het vinden van een stabiele toestand. Deze motor schokte regelmatig en draaide soms heel kortstondig in de tegengestelde richting. Het is niet duidelijk of de onsymmetrische belasting hiervan de oorzaak is, want het schokken gebeurde ook een enkele keer zonder dat de motor belast was met het gewicht van de spiegel. Later ontdekten we dat dit kwam doordat onze servomotors 6V in plaats van de gebruikelijke 5V nodig hebben als voedingsspanning.

Figuur 4.15 toont een alternatieve oplossing die we zouden kunnen testen indien de

Figuur 4.14: Onderkant van het prototype. Hier zien we de drie wielen die elk één draad besturen en zo de spiegel kunnen richten.

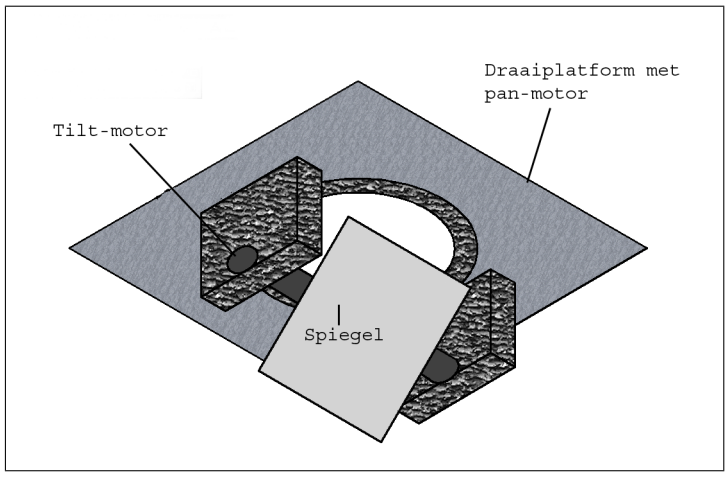
huidige driedradenopstelling totaal niet geschikt blijkt. Bij dit alternatief wordt een draaiplatform gebruikt om de panhoek aan te passen. Op het platform staan twee bevestigingspunten waarop de tiltas wordt bevestigd.

Tweede prototype

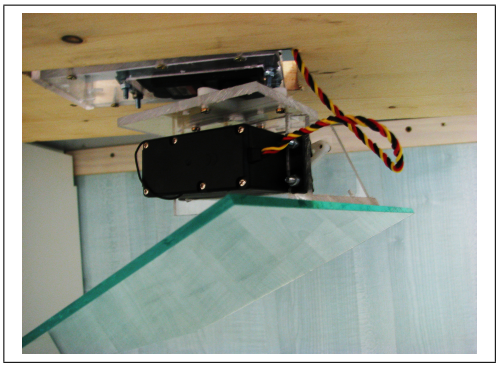
Later hebben we een nieuw prototype gebouwd met twee servomotors die respectievelijk de pan- en tilthoek op zich nemen, gelijkaardig aan de opstelling welke door IBM gebruikt werd. De eerste tests geven ook positieve resultaten. Zie figuren 4.16 en 4.17.

4.2.4 Besturing van de servomotors

In ons eerste prototype worden de servomotors bestuurd met behulp van de Phidgets USB interface. (Zie figuur 4.18.) Hier kunnen we de drie servomotors op aansluiten,



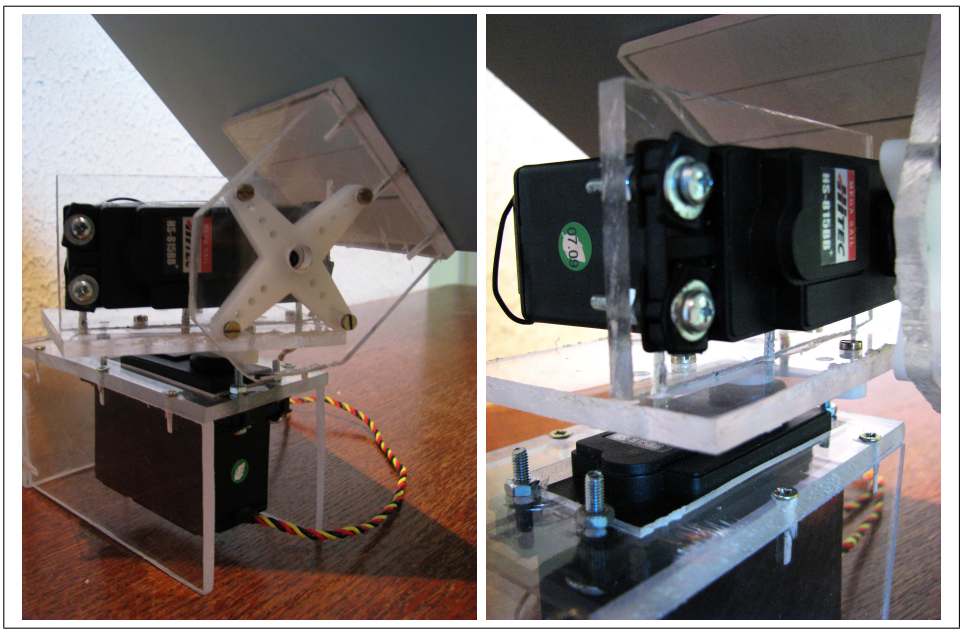
Figuur 4.15: Een alternatief voor ons prototype



Figuur 4.16: Tweede prototype

en deze kunnen eenvoudig via de .NET-bibliotheek van Phidgets aangesproken worden.

De communicatie tussen de servomotors en het Phidgets-kaartje gebeurt via pulse width modulatie (PWM), zoals figuur 4.19 toont. Minstens 50 à 60 keer per seconde wordt er een korte puls gestuurd met een lengte tussen de 1 à 2 milliseconde. De servomotor zal dan streven naar de hoek die overeenkomt met die pulslengte. Deze frequentie van rond de 50Hz is onafhankelijk van de snelheid waarmee de servo beweegt, dit is enkel noodzakelijk om te garanderen dat de servo bekrachtigd blijft. Wanneer er gedurende een langere periode geen signaal naar de servomotor gestuurd wordt, zal deze geen kracht meer uitoefenen en in een vrije toestand terechtkomen. De snelheid waarmee de servo draait, is afhankelijk van de snelheid waarmee de



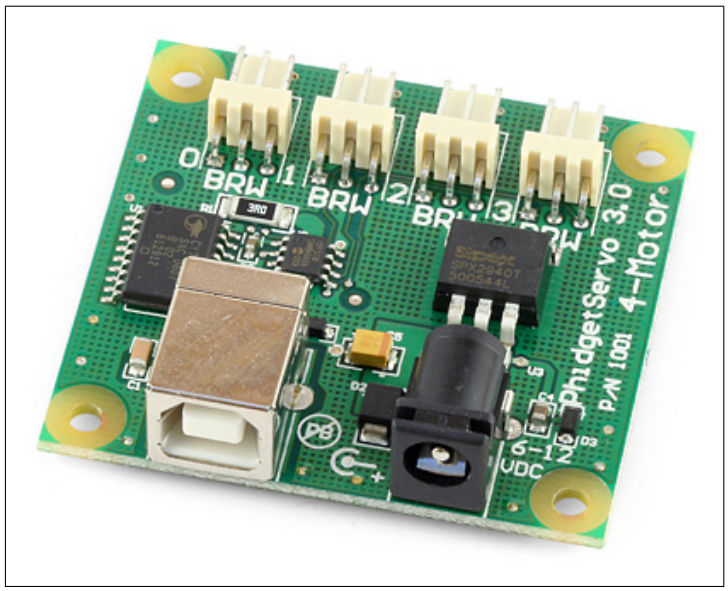
Figuur 4.17: Tweede prototype, close-ups

pulslengte varieert.

De gebruikte servomotors vereisen 6V als voedingsspanning in plaats van de gebruikelijke 5V. Daarom kan enkel het PWM-signaal van de Phidgets hardware gebruikt worden en moet hier een eigen voeding tussengeplaatst worden.

Voor het tweede protype hebben we een Atmega168 microprocessor gebruikt die de servo's via het ethernet bestuurt. (Zie figuur 4.20.) Dit heeft verschillende voordelen:

- er zijn geen speciale drivers meer nodig;
- deze microprocessor kan gedeeltelijk zelfstandig werken zodat een korte netwerkstoring geen probleem vormt;
- het hoofdsysteem wordt minder belast omdat deze microprocessor een deel van de berekeningen op zich neemt;
- we zijn niet meer beperkt tot de lengte van USB-kabels en
- de servo's kunnen van eender welke computer op het netwerk aangesproken worden.

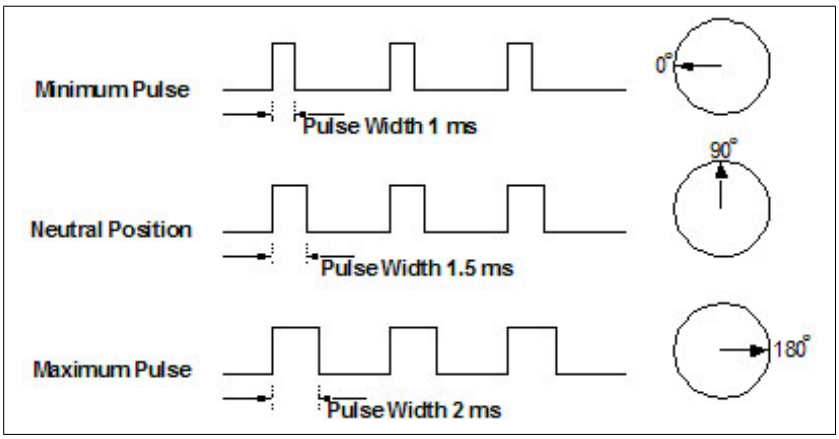


Figuur 4.18: Controller voor de servomotors. Deze heeft een USB-ingang en vier uitgangen voor de servomotors. (Bron: http://www.phidgets.com/)

Het LCD-schermpje dat aan de besturing hangt, toont bovendien het IP address. Deze besturing komt dus de gebruiksvriendelijkheid ten goede. Het nadeel hiervan is dat er meer initieel programmeerwerk bij komt kijken; de microprocessors moeten immers nog in C geprogrammeerd worden.

Nog een ander bruikbaar alternatief is het open-source hardware platform 'Arduino' [3].

Merk op dat servomotors nooit feedback geven. Er kan een PWM-signaal naar de motor gestuurd worden, maar deze zal geen informatie terugzenden. Met andere woorden: er is geen zekerheid over de werkelijke positionering van de spiegel. De beste manier om hiermee om te gaan, is gewoonweg de servo voldoende tijd geven om zich te draaien. Hoeveel tijd dat is, hangt af van zijn snelheid en dat kunnen we enkel experimenteel vaststellen.



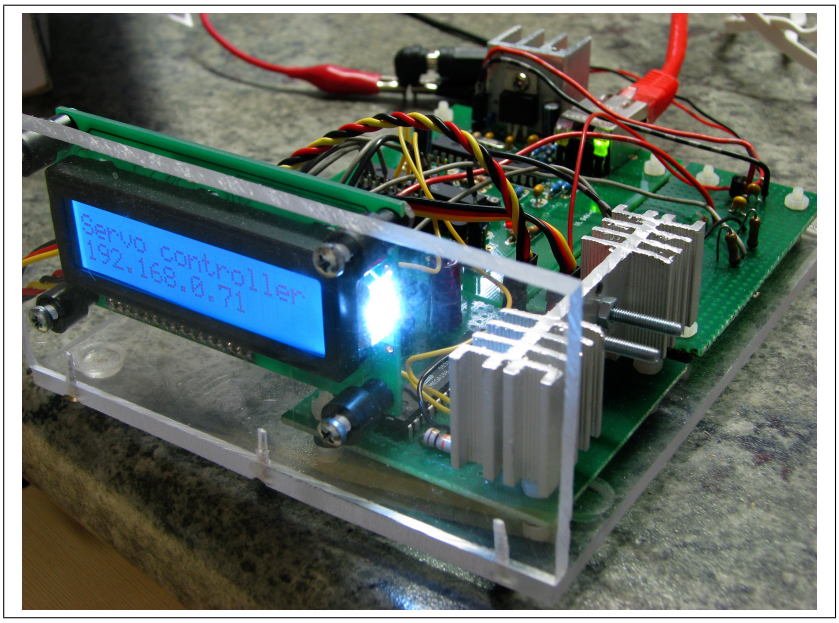
Figuur 4.19: Pulse width modulation – servobesturing (Bron: http://servocity.com)

4.2.5 Kalibratie tussen hard- en software

Zoals al eerder is vermeld, is het belangrijk dat de hardware (de status van de servomotors) synchroon blijft met de software (de beelduitvoer). Anders zullen verkeerde beelden op verkeerde plaatsen geprojecteerd worden.

Een mogelijke methode is de volgende:

- We willen de hardware zo optimaal mogelijk benutten; we gaan dus de uiterste grenzen van elke pan- en tiltmotor aftasten. Dit bereik delen we door 256 of 65536 voor een resolutie van respectievelijk 8 of 16 bits.
- De pan/tilt-set-up moet nu geprogrammeerd worden zodat deze waardes tussen 0 en 256 of 0 en 65536 kan ontvangen. In onze set-up wordt dit gedaan via een UDP-pakket, maar het zou ook via een USB interface kunnen gebeuren.
- In de configuratie van de ED engine moeten we nu de positie en afmetingen van alle projectieoppervlaktes – gewoonlijk de muren – definiëren volgens de werkelijke situatie. Ook de positie van de projector moeten we opgeven. Al deze waardes kunnen we letterlijk meten.
- Uiteindelijk moeten we nog de minimum- en maximumwaardes voor de pan- en tilthoeken opgeven in de configuratie. Dit kan door te experimenteren en de waardes aan te passen als deze uitersten niet overeenkomen.



Figuur 4.20: Het microprocessorcircuit dat eerder al gebruikt werd, is nu uitgebreid om de servo's te besturen over ethernet.

- Als alles werkt, zal de pan/tilt driver in de ED de huidige hoeken continu omrekenen naar een waarde tussen 0 en *max*. Vervolgens zal de besturing van de projector (de servomotors) deze waarde terug gaan interpoleren tussen zijn minimum- en maximumwaarde.

4.3 Mogelijke uitbreidingen

4.3.1 ED over een netwerk

In grote opstellingen in de praktijk zal één enkele computer al snel niet meer voldoende zijn om tientallen projectors te besturen en misschien een veelvoud aan applicaties uit te voeren. We zullen nood hebben aan een groter gedistribueerd systeem.

Dit is niet vanzelfsprekend te realiseren. Als een applicatie op een andere computer wordt uitgevoerd dan waar de beelduitvoer moet terechtkomen, zal al deze beeldoutput via het netwerk getransporteerd moeten worden. De huidige gigabitnetwerken

zijn waarschijnlijk snel genoeg om dit dataverkeer in realtime te kunnen versturen, maar het probleem situeert zich eerder bij het serialiseren en deserialiseren van de GUI-informatie. De technologie die wij gebruiken, genaamd WPF, staat niet toe dat een usercontrol wordt geserialiseerd, en hetzelfde probleem zouden we hebben bij Windows Forms. Dit is een terugkerend probleem dat typisch is voor GUI's: de gebruikersinterface is single-threaded. (Denk aan de Win32 event loop.) De enige bruikbare oplossing is dus het renderen van alle vensters op de computer waarop de bijbehorende applicatie draait. Hierbij verliezen we dan alle optimalisaties die WPF voor ons anders wel kan uitvoeren, zoals het direct renderen van elk venster in de juiste resolutie en het overslaan van verborgen vensters. Bovendien ontvangt de computer waarop de 3D engine vervolgens uitgevoerd wordt, hogeresolutiebeelden voor elke applicatie. Deze moeten allemaal in realtime naar de grafische kaarten gestuurd worden voor de beeldtransformatie. Vanzelfsprekend vereist dat een zeer krachtig systeem.

Wat wel mogelijk is, indien een kleine delay wordt aanvaard, is het implementeren van een VNC client of X11-server bovenop een WPF UserControl. Zo kan een enkele applicatie toch over het netwerk uitgevoerd worden.

Natuurlijk kan de applicatielogica zelf wel gedistribueerd worden.

4.3.2 Security

In ons framework moeten applicaties elkaar volledig vertrouwen. Ze worden immers allemaal in hetzelfde proces uitgevoerd, en de GUI van de verschillende applicaties zitten zelfs in dezelfde thread. Dit veroorzaakt enkele problemen:

- Toepassingen kunnen het geheugen van elkaar lezen. Vertrouwelijke informatie kan eenvoudig gestolen worden.
- Toepassingen kunnen elkaar laten crashen. Ze kunnen zelfs simpelweg de volledige everywhere display laten vastlopen door een oneindige lus in de GUI thread te injecteren.

- In het taakbeheer van het besturingssysteem worden alle applicaties herkend als één enkele ED-applicatie. Wanneer dit afgesloten wordt, zullen alle toepassingen afgesloten worden.

Zolang we zelf de auteur zijn van de code is deze situatie aanvaardbaar. Pas wanneer de gebruiker dynamisch applicaties gaat laden, ontstaat een gevaarlijke situatie die vergelijkbaar is met andere programma's waarbij met een plug-insysteem wordt gewerkt. (Bijvoorbeeld Mozilla Firefox waar alle op XUL-gebaseerde extensies zich in de GUI integreren. [2])

4.3.3 Privacy

Tot nu toe hebben we nog geen rekening gehouden met de privacy van de eindgebruiker. Wanneer gevoelige informatie zoals e-mails, agenda en andere persoonlijke gegevens in het groot wordt geprojecteerd, moet er rekening gehouden worden met andere personen die 'toevallig' meekijken.

Het detecteren van personen zou kunnen gebeuren via RFID of Bluetooth, maar dan verwachten we dat iedereen een RFID-tag, Bluetooth-apparaat of een ander detecteerbaar label bij zich draagt. Vanzelfsprekend is deze methode dus eenvoudig te omzeilen.

4.3.4 Geavanceerde invoer

De invoer wordt geabstraheerd, maar de huidige abstractie ondersteunt enkel toetsaanslagen. Een soort van aanwijsapparaat is zeker een nuttige aanvulling. Hier bestaan wel zeer veel verschillende soorten in. Coördinaten kunnen absoluut of relatief zijn en wanneer een muispijl gebruikt wordt, kunnen deze gebonden zijn aan een enkel venster of zich door het hele systeem bewegen.

Een invoerapparaat dat we kunnen implementeren, is een laser pointer die gebruikt kan worden om een venster doorheen de ruimte te slepen. Met behulp van een

[2]Mozilla Extensions: `https://developer.mozilla.org/en/Extensions`

webcam kan de positie waarop de laser gericht is, waargenomen worden. De plaats waarop de laser schijnt, is altijd veel helderder dan de omliggende gedeeltes; dus die positie in het waargenomen beeld kan eenvoudig bepaald worden. Door het vergelijken van opeenvolgende frames kan vastgesteld worden hoe de laser over de muur wordt verplaatst en dat kan bijvoorbeeld gekoppeld worden aan de beweging van een venster.

Merk op dat we dit als relatief invoerapparaat gebruiken. In theorie kan een laser ook als absoluut invoerapparaat gebruikt worden wanneer we de plaats van de laser terug linken aan een plaats in het ED-model. Dit vereist een goede kalibratie, maar opent wel mogelijkheden zoals het activeren van knoppen in de projectie met een laser.

De XWand die op een onderzoekslabo van Microsoft is ontwikkeld, biedt ook inspiratie [23]. Deze bevat een reeks van sensoren waarmee de 3D-oriëntatie volledig bepaald kan worden. De sensoren bestaan uit gyroscopen en accelerometers en de tracking gebeurt via infrarood LEDs. Als de gebruiker met deze 'wand' naar een object in de omgeving wijst, kan bepaald worden welk object hij bedoelt.

Figuur 4.21: De XWand als invoerapparaat (Bron: [23].)

4.3.5 Projectie op plafond en vloer

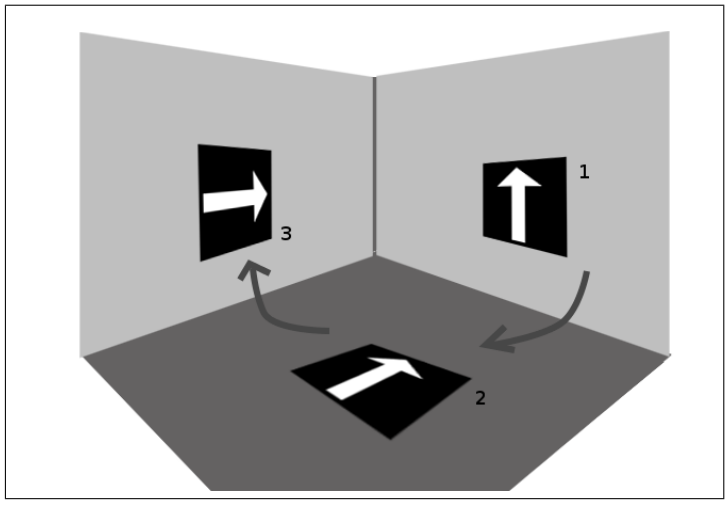
De huidige rendering engine ondersteunt vloer- en plafondprojecties, maar niet alle overgangen tussen deze vlakken. Het probleem situeert zich hier bij de oriëntatie van applicatievensters op horizontale vlakken. Op een muur is heel duidelijk waar

de onder- en bovenkant is, maar op het plafond of op de vloer kunnen we moeilijk spreken van een beeld dat ondersteboven staat.

Stel dat we een klok zouden projecteren op een muur en deze vervolgens omlaag slepen naar de vloer. Hier is geen enkele rotatie voor nodig. De bovenkant van de klok zal wijzen naar de muur waarvan deze afkomstig is. Stel dat we hierna deze klok terug omhoog willen slepen naar een andere muur, één die loodrecht staat op de eerste muur. Zonder rotatie van deze klok zou het beeld 90 graden gedraaid zijn. Figuur 4.22 demonstreert dit.

We nemen aan dat personen eerder de neiging hebben om in het centrum van de kamer te gaan staan om een object dat vlakbij de muur staat te bekijken dan tussen de muur en dat voorwerp. Als we daarmee rekening mogen houden, dan kunnen het centrum van de vloer als 'onder' en de raakpunten met de muren als 'boven' beschouwd worden.

Indien een beeld zich op de grens tussen meerdere vlakken bevindt, is het niet vanzelfsprekend om dit op een natuurlijke manier hierop te projecteren. In het bijzonder als de omgeving uit schuine muren, bolle en holle vormen bestaat, wordt het een zware klus om alles nog natuurlijk weer te geven. Waarschijnlijk is de beste oplossing in deze gevallen dit soort situaties te vermijden en softwarematig de meest geschikte projectieplaatsen te selecteren.



Figuur 4.22: Verplaatsen van de projectie tussen verschillende vlakken

4.4 Toekomstig werk

Buiten de verschillende uitbreidingen die in de vorige sectie besproken werden, zal er ook nog onderzoek moeten gebeuren naar:

- interactiemethodes;
- het inspelen op de positie van de gebruiker via een tracking-mechanisme;
- augmented reality (het integreren van virtuele data met reële voorwerpen);
- collaboratieve applicaties (vaak zullen meerdere personen in dezelfde ruimte aanwezig zijn en van dit systeem gebruik maken.);
- ...

Hoofdstuk 5

Evaluatie en conclusie

Uit ons onderzoek kunnen we concluderen dat een stuurbare projectie zeer veel nuttige toepassingen heeft, maar dat alles goed op elkaar afgestemd moet zijn om tot een werkend geheel te komen. Hoewel zo een projectiesysteem erg flexibel is, heeft het ook zijn beperkingen. Schaduw van personen is moeilijk te voorkomen en het voorzien van interactie is niet altijd vanzelfsprekend.

Wij hebben een framework gebouwd waarmee WPF-applicaties met enkel een minimale aanpassing in een ED-projectiesysteem geplaatst kunnen worden. Tot hiervoor bestond er geen eenvoudige methode om dit te realiseren. Door de modulaire opbouw kan bovendien zowel de invoermethode als de besturing van de projector geabstraheerd worden.

Het is wel duidelijk dat dit nog geen afgerond geheel is. Er zal nog veel werk zijn om hiermee een nuttige applicatie te maken die voor gebruikers werkelijk een meerwaaarde betekent.

Bibliografie

[1] X.org foundation. http://www.x.org/.

[2] B. Ahlborn, O. Kreylos, S. Shafii, B. Hamann, and O. Staadt. Design and implementation of a foveal projection display. http://www.cipic.ucdavis.edu/~hamann/AhlbornKreylosShafiiHamannStaadtIJIG2007PaperAsSubmited12112006.pdf.

[3] Arduino. Open-source electronics prototyping platform. http://arduino.cc/.

[4] A. Butz and A. Krüger. A generalized peephole metaphor for augmented reality and instrumented environments. 2003.

[5] A. Butz and A. Krüger. Applying the peephole metaphor in a mixed-reality room. 2006.

[6] L. Chan, W. Ye, S. Liao, Y. Tsai, J. Hsu, and Y. Hung. A flexible display by integrating a wall-size display and steerable projectors. *Lecture notes in computer science*, 4159:21, 2006.

[7] H. Chen, R. Sukthankar, G. Wallace, and T.-J. Cham. Calibrating scalable multi-projector displays using camera homograpyh trees. 2001.

[8] I. Claudio Pinhanez. Creating ubiquitous interactive games using everywhere displays projectors. 2002.

[9] I. R. Division. Ubiquitous interactive graphics. 2002. http://www.research.ibm.com/ed/publications/rc22495.pdf.

[10] F. D. P. H. P. H. U. K. U. K. H. V. Günter Hollemann, Bernd Braun. Rgb lasers for laser projection displays. 2000. http://www.laserfx.com/Backstage.LaserFX.com/Systems/rgb-dpss.pdf.

[11] IBM. Augmenting reality with projected interactive displays. 2001.

[12] IBM. Bluespace: Creating a personalized and context-aware workspace. 2001. http://www.research.ibm.com/ed/publications/rc22281.pdf.

[13] IBM. Transforming surfaces into touch-screens. 2001. http://www.research.ibm.com/ed/publications/rc22273.pdf.

[14] IBM. Augmenting a retail environment using steerable interactive displays. 2003. http://www.research.ibm.com/ed/.

[15] IBM. Embedding interactions in a retail store environment. 2003. http://www.research.ibm.com/ed/.

[16] C. P. IBM. The everywhere display project. 2003. http://www.research.ibm.com/ed/.

[17] R. K. IBM, A. Levas, and C. Pinhanez. Dynamically reconfigurable vision-based user interfaces. 04 2003. http://www.research.ibm.com/ed/publications/icvs03.pdf.

[18] R. Mitsubishi, Raskar, J. van Baar, P. Beardsley, T. Willwacher, S. Rao, and C. Forlines. ilamps: Geometrically aware and self-configuring projectors.

[19] R. Mitsubishi Electric Research Laboratories, Raskar and P. Beardsley. A self correcting projector. 2002.

[20] I. Mitsugami, N. Ukita, and M. Kidode. Fixed-Center Pan-Tilt Projector and Its Calibration Method. 2005. http://ai-www.naist.jp/people/ukita/MyPapers/MVA2005.pdf.

[21] G. Pingali, C. Pinhanez, T. Levas, R. Kjeldsen, and M. Podlaseck. User-following displays. 2002. http://www.research.ibm.com/ed/publications/icme02.pdf.

[22] R. Sukthankar. Toward ambient projection for intelligent environments. 2005.

[23] A. Wilson. Xwand. http://research.microsoft.com/en-us/um/people/awilson/wand/default.htm.

[24] R. Yang, D. Gotz, J. Hensley, H. Towles, and M. Brown. PixelFlex: a reconfigurable multi-projector display system. *Proceedings of the conference on Visualization'01*, pages 167–174, 2001. http://vis.uky.edu/~gravity/Research/Yang-research/publications/PAPERS/yang01-Vis.pdf.

Hoofdstuk 6

Bijlagen

6.1 Application interface

De *ApplicationInterface* namespace bevat de abstracte classes waar elke ED-applicatie van afgeleid moet zijn.

Listing 6.1: Application interface

```
namespace Everywhere.ApplicationInterface
{
    public delegate Model.IWindow RegisterWindowHandler(string name, UserControl window);
    public delegate Model.IInput RegisterInputHandler(string name, EverywhereInput input);

    public delegate void InputKeyEventHandler(Key key);

    /// <summary>
    /// Base class for every ED application
    /// </summary>
    public class EverywhereApplication
    {
        protected readonly Model.IEverywhereModel Model;
        protected readonly Dictionary<string, string> Parameters;
        protected readonly RegisterWindowHandler RegisterWindow;
        protected readonly RegisterPanTiltOutputHandler RegisterPanTiltOutput;
        protected readonly RegisterInputHandler RegisterInput;

        // Every application should have the same constructor as this one
        public EverywhereApplication(EverywhereApplicationArguments arguments)
        {
            ...
        }
```

```
    }

    /// <summary>
    /// Arguments which are passed during the initialisation of an EverywhereApplication
    /// </summary>
    public class EverywhereApplicationArguments
    {
        public Model.IEverywhereModel Model { get { return _model; } }
        public Dictionary<string, string> Parameters { get { return _parameters; } }
        public RegisterWindowHandler RegisterWindow { get { return _registerWindow; } }
        public RegisterInputHandler RegisterInput { get { return _registerInput; } }

        ...
    }

    /// <summary>
    /// Base class for an input
    /// </summary>
    public class EverywhereInput
    {
        public event InputKeyEventHandler KeyUp;
        public event InputKeyEventHandler KeyDown;

        public void SendKeyUp(Key key)
        {
            if (KeyUp != null) KeyUp(key);
        }

        public void SendKeyDown(Key key)
        {
            if (KeyDown != null) KeyDown(key);
        }
    }
}
```

6.2 Omgevingsmodel

De namespace *Model* bevat de interface die applicaties kunnen gebruiken om informatie over het model op te vragen. Deze is gebouwd volgens de noden van de huidige applicaties en is zeker nog niet compleet. Toekomstige applicaties zullen misschien nog andere informatie nodig hebben uit het model, dus deze interface kan nog groeien.

Listing 6.2: Het omgevingsmodel

```
namespace Everywhere.Model
{
   public delegate void WallEventHandler(IWall wall);
   public delegate void ProjectorEventHandler(IProjector projector);
   public delegate void ApplicationEventHandler(IApplication application);
   public delegate void WindowEventHandler(IWindow window);
   public delegate void WindowMappingEventHandler(IWindowMapping mapping);
   public delegate void InputMappingEventHandler(IInputMapping mapping);
   public delegate void InputEventHandler(IInput input);
   public delegate void PanTiltOutputEventHandler(IPanTiltOutput output);

   public interface IEverywhereModel
   {
      // Events
      event WallEventHandler WallAdded;
      event ProjectorEventHandler ProjectorAdded;
      event ApplicationEventHandler ApplicationInserted;
      event WindowEventHandler WindowLoaded;
      event WindowEventHandler WindowRemoved;
      event WindowMappingEventHandler WindowMappingAdded;
      event WindowMappingEventHandler WindowMappingRemoved;
      event InputMappingEventHandler InputMappingAdded;
      event InputEventHandler InputAdded;
      event PanTiltOutputEventHandler PanTiltOutputAdded;

      /// <summary>
      ///   Retrieve all windows in the model
      /// </summary>
      IEnumerable<IWindow> GetWindows();

      /// <summary>
      ///   Retrieve window from the model matching with this
      ///   application window.
      /// </summary>
      IWindow GetWindow(UserControl applicationWindow);
      IWindow GetWindow(string name);

      /// <summary>
      /// List all the window mappings
```

```
    /// </summary>
    IEnumerable<IWindowMapping> GetWindowMappings();
    IEnumerable<IWindowMapping> GetWindowMappings(IWindow window);
    IEnumerable<IWindowMapping> GetWindowMappings(UserControl window);

    /// <summary>
    /// List all the inputs
    /// </summary>
    /// <returns></returns>
    IEnumerable<IInput> GetInputs();
    IInput GetInput(string name);

    /// <summary>
    /// List all the walls
    /// </summary>
    IEnumerable<IWall> GetWalls();
    IWall GetWall(string name);

    /// <summary>
    ///   Steer projectors to get this window into focus
    ///   The engine can decide which mappings are used to
    ///   show this window. If possible, all mappings will be used.
    /// </summary>
    void Focus(IWindowMapping window);
    void Focus(IWindowMapping window, IProjector projector);
    void Focus(IWindow window);
    void Focus(IWindow window, IProjector projector);

    /// <summary>
    ///   List all the projectors
    /// </summary>
    IEnumerable<IProjector> GetProjectors();
    IProjector GetProjector(string name);

    /// <summary>
    ///   List all the currently active applications.
    /// </summary>
    IEnumerable<IApplication> GetApplications();

    /// <summary>
    /// Creates a new mapping between a window and a wall
    /// </summary>
    IWindowMapping CreateWindowMapping(IWindow window, IWall wall);
}

public interface IWall
{
    string Name { get; }
    Size2d Size { get; }
}

public interface IProjector
```

```
{
    string Name { get; }
    PanTiltRange Range { get; }
    float Pan { get; set; }
    float Tilt { get; set; }

    UserControl CreateRenderer();

    event ProjectorEventHandler ViewAngleChanged;
}

public interface IApplication
{
    string Name { get; }
    ApplicationInterface.EverywhereApplication EverywhereApplication { get; }
}

public interface IWindow
{
    string Name { get; }
    UserControl EverywhereWindow { get; }
}

public interface IWindowMapping
{
    void MoveWindowHorizontal(float amounth);
    void MoveWindowVertical(float amounth);

    // Dimensions
    Point2d Position {get; set; }
    Size2d Size {get; set; }

    float FloatingDistance { get; set; }

    // Wall and window
    IWall Wall { get; }
    IWindow Window { get; }

    // Events
    event WindowMappingEventHandler PositionChanged;
}

public interface IInputMapping
{
    IInput Input { get; }
    IWindow Receiver { get; }
}

public interface IInput
{
    string Name { get; }
}
```

```
    public interface IPanTiltOutput
    {
        string Name { get; }
        IProjector Projector { get; }
    }
}
```

6.3 Lengte van de draden

Hier volgt de methode die we hebben gebruikt om de draadlengte van ons eerste prototype te berekenen.

Afhankelijk van de pan- en tilthoek die men wil verkrijgen, moet de draadlengte van de drie draden anders ingesteld worden. De berekening is vrij eenvoudig.

Laten we afspreken dat draad 1 de hoek van 0 graden vormt, draden 2 en 3 vormen dan respectievelijk hoeken van 120 en 240 graden rond de oorsprong. (Zie afbeelding 6.1)

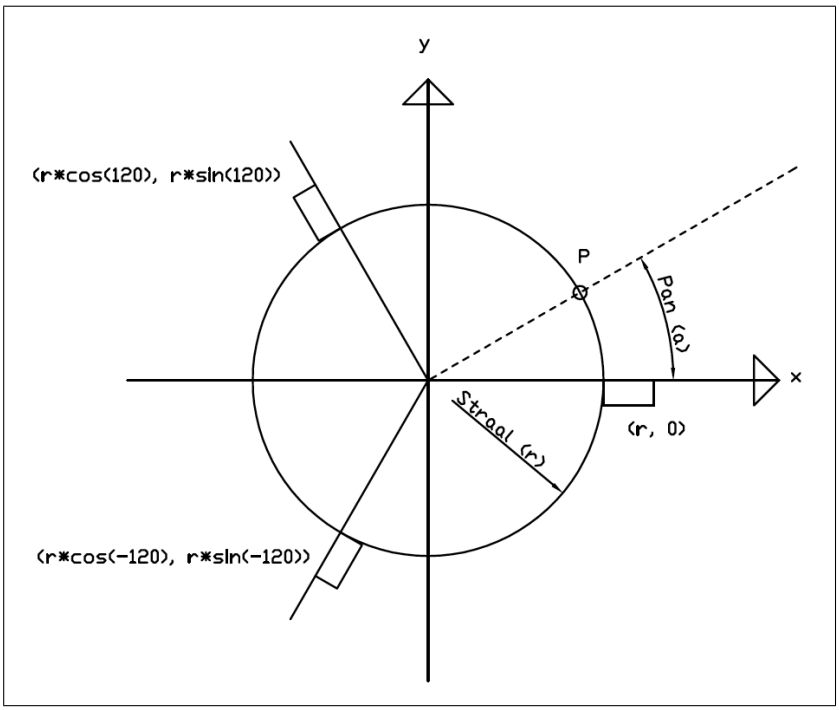
De coördinaten van een draadaanhechtingspunt (x, y, z) kunnen we dan invullen in volgende vergelijkingen.

$$\begin{cases} x = rcos(\gamma) \\ y = rsin(\gamma) \\ ax + by + cz = 0 \end{cases}$$

Verder kunnen we de spiegel voorstellen als een vlak door de oorsprong. Stel dat de spiegel (a, b, c) als normaalvergelijking heeft, dan:

$$spiegel \leftrightarrow ax + by + cz = 0 \tag{6.1}$$

Als we vertrekken in de begintoestand waarbij zowel de tilt- als de panhoek gelijk zijn aan 0, dan is de normaalvector:



Figuur 6.1: De drie draden

$$(a, b, c) = (0, 0, 1) \tag{6.2}$$

Na het toepassen van een tilthoek β:

$$(a, b, c) = (sin\beta, 0, cos\beta) \tag{6.3}$$

Na het toepassen van een panhoek α:

$$(a, b, c) = (cos\alpha sin\beta, sin\alpha sin\beta, cos\beta) \tag{6.4}$$

Deze a, b, c, x en y vullen we nu in de eerdere vergelijking van het vlak:

$$rcos\alpha sin\beta cos\gamma + rsin\alpha sin\beta sin\gamma + zcos\beta = 0 \tag{6.5}$$

$$z = -(rcos\alpha sin\beta cos\gamma + rsin\alpha sin\beta sin\gamma)/cos\beta \tag{6.6}$$

Deze variabele z bepaalt dus de lengte van het touw. Als we het touw laten oprollen rond het wiel van een servomotor, dan is de draaiingshoek recht evenredig met deze variabele z, waarbij de factor gelijk is aan de diameter van het wiel.