Fakulteten för ekonomi, kommunikation och IT
Avdelningen för Datavetenskap

Mathias Andersson
Henrik Bäck

En anpassningsbar applikationsstruktur för flerpunktspekskärmar

An adaptable application structure for multi-touch screens

Examensarbete, 30 högskolepoäng
Civilingenjörsprogrammet IT

Datum:	2009-01-15
Handledare:	Katarina Asplund
Examinator:	Donald F. Ross
Löpnummer:	E2009:01

Karlstads universitet 651 88 Karlstad
Tfn 054-700 10 00 Fax 054-700 14 60
Information@kau.se www.kau.se

Denna uppsats är skriven som en del av det arbete som krävs för att erhålla en civilingenjörsexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

_____
Mathias Andersson

_____
Henrik Bäck

Godkänd, 2009-01-15

_____
Handledare: Katarina Asplund

_____
Examinator: Donald F. Ross

Sammanfattning

Klassisk mus-interaktion har begränsningen att bara en punkt kan aktiveras på skärmen i taget. Interaktionen med de flesta applikationer blir, på grund av detta, sekventiell vilket kan vara en begränsning eftersom människan är van att utforska problem simultant med båda händerna. Flerpunktspekskärmar[1] är en relativt ny teknik som möjliggör nya interaktionsmöjligheter där flera pekpunkter kan vara aktiva simultant.

Det här examensarbetet fokuserar på problemen kring att skapa en applikation med användargränssnitt för flerpunktspekskärmar. Applikationen som utvecklats har en grundstruktur vilken är möjlig att vidareutveckla och på så sätt anpassas för att passa nya behov, till skillnad från de implementationer som granskats vid förstudien vilka fokuserar på att lösa specifika problem. Under examensarbetet har också en flerpunktspekskärm byggts av komponenter som kunnat köpas i detaljhandeln.

[1]Engelska: Multi-Touch Screens

Abstract

Classical mouse interaction is limited in that only a single screen point at a time can be activated. Most application interaction is, due to this, performed in a sequential manner. This may be a limiting factor as humans by nature explore problems with both hands simultaneously. Multi-touch screens are a relatively new type of technology that enables new ways of interaction where multiple touch points can exist simultaneously.

This thesis focuses on the problem with developing an application with a user interface designed for multi-touch screens. The application that has been developed defines a base structure from which future applications can be developed, unlike implementations examined in the feasibility study which mainly focus on solving specific problems. A multi-touch screen is also built using standard off the shelf components.

Tack

Ett stort tack till Magnus Andersson för tillverkningen av det chassi som all hårdvara monterats i.

Ett stort tack till Katarina Asplund vid Karlstads universitet för all den tid hon har lagt ned på detta arbete.

Tack till Peter Rönnlund vid Karlstads universitet som bjöd på fika i en trött stund.

Innehåll

1 Inledning **1**

1.1 Problem, mål och motivation 1

1.2 Primära resultat 2

1.3 Jämförelse med existerande produkter 3

1.4 Denna uppsats disposition 3

2 Bakgrund **5**

2.1 Historia 5

2.1.1 En-punktsinmatning 6

2.1.2 Flerpunktsinmatning 9

2.2 Pekskämar och flerpunktspekskärmar idag 12

2.2.1 Möjliga tillämpningar 13

2.3 Sammanfattning 14

3 Potentiella lösningar **15**

3.1 Krav på systemet 15

3.1.1 Hårdvara 15

3.1.2 Mjukvara 15

3.2 Hårdvara 16

3.2.1 Pekpunktsregistrering 16

3.2.2 Bildåtergivning 21
3.3 Mjukvara 23
3.3.1 Spårare 23
3.3.2 Applikation 25

4 Hårdvaruprototyp **29**
4.1 Pekpunktsregistrering 29
4.1.1 Kamera 29
4.1.2 Beräkningar 32
4.1.3 Bildinhämtning 36
4.2 Pekpunkter 36
4.2.1 Ljuskälla 36
4.2.2 Problem 38
4.2.3 Infraröd ljuspenna 38
4.3 Bildåtergivning 39
4.3.1 Val av lösning 39
4.4 Chassi 42
4.4.1 Mått 42
4.4.2 Intern uppbyggnad 43
4.4.3 Anslutningar 45
4.4.4 Start och avstängning 46
4.5 Sammanfattning 46

5 Mjukvaruprototyp **47**
5.1 Spårare 48
5.1.1 Funktionalitet 48
5.2 Kommunikation – spårare och applikation 51
5.2.1 Om TUIO 51

5.3 Applikation 53
5.3.1 Uvecklingsmiljö 54
5.3.2 Implementation 55
5.4 Sammanfattning 72

6 Utvärdering av prototyp **75**
6.1 Utvärdering av krav 75
6.1.1 Hårdvara 75
6.1.2 Mjukvara 77
6.2 Iakttagelser och begränsningar 80
6.2.1 Fördröjningar i interaktionen 80
6.2.2 Grafikproblem 81
6.3 Sammanfattning 81

7 Slutsatser **83**
7.1 Detta projekt 83
7.1.1 Pekpunktsinmatning 83
7.1.2 Optimering av mjukvara 84
7.2 Framtida vidareutveckling 84
7.2.1 Användargränssnittsanpassning 84
7.3 Lärdomar 86

Referenser **87**

A Figurreferenser **91**

B Ritningar för chassi **93**

C Programkod, två objekt placeras **101**

Figurer

1.1 Schematisk översikt över systemets uppbyggnad 2

2.1 Historisk tidslinje för alternativa inmatningssytem 6
2.2 THE RAND TABLET 7
2.3 PLATO 1978 8
2.4 reactTable* 10
2.5 iPhone 3G 11
2.6 Microsoft Surface 12

3.1 Utspridd upplysning 18
3.2 Frustrerad total intern reflektion 19
3.3 Lysdiodspenna 20

4.1 Hårdvaran 30
4.2 XBOX Live Vision 31
4.3 Kamerans synfält 33
4.4 Kamerans synfält utökas genom en spegel 35
4.5 Utspridd belysning uppmonterad 37
4.6 Pekpennas utseende och uppbyggnad 39
4.7 Plattskärmens bildpanel 40
4.8 Lager för att bland annat ge vit färg 41
4.9 Plattskärm monterad i chassi 42

4.10 Hårdvarans uppbyggnad internt 43
4.11 Kameran är monterad ovan plattskärmens kretskort 44
4.12 T.v intag till chassit och t.h. intern koppling 45

5.1 Översikt över mjukvaruprototyp 47
5.2 Bildsegmentering med tre sammanhängande områden 50
5.3 TUIO, Meddelandetyper 52
5.4 TUIO, Profiler 53
5.5 Mjukvaruprototypens delar 55
5.6 Koordinatsystem och ankarpunkt för lager 61
5.7 Djupledsordning för lager 62
5.8 Lager läggs i vyn via lagerhanteraren 62
5.9 Rotation kring ankarpunkt 67
5.10 Nytt koordinatsystem vid ankarpunkten 68
5.11 Skalning av lager 69

6.1 En ögonblicksbild av gränssnittet 76
6.2 Rotation och skalning 78
6.3 Skärmavbild av pianot 79

7.1 Ett dåligt användargränssnitt för pekning 85

Tabeller

3.1 För och nackdelar för plattskärm och digitalprojektor 22
3.2 Olika spårare som implementerar TUIO 24
3.3 Meddelande-typer, internt i applikation 27

4.1 Kameramodell 30
4.2 Mätvärden för kamerans synfält 33
4.3 Komponenter för konstruktion av infraröda ljuspennor 39
4.4 Plattskärmsmodell 41
4.5 Glasspecifikation 44
4.6 Spegelspecifikation 45

5.1 OSC Argumenttyper 52
5.2 TUIO, Meddelandeargument 54

6.1 Prislista 77

Programkod

5.1 Meddelanden från TUIO-lyssnaren 57
5.2 Hantering av meddelanden 58
5.3 Hantering av ”set”-meddelande 59
5.4 Hantering av ”alive”-meddelande 60
5.5 Protokollet Touching 63
5.6 Beräkna rotationsvinkeln 69
5.7 Applicera en riktning på vinkeln 70
5.8 Beräkning av ny skalfaktor 71

Kapitel 1
Inledning

Den här uppsatsen kommer att visa hur man kan lösa problemet med att mata in flera simultana pekpunkter i ett system via en och samma in-dataenhet. Uppsatsen kommer att visa på ett sätt att konstruera en in-dataenhet men framför allt visa på hur en mjukvara för ändamålet kan konstrueras.

1.1 Problem, mål och motivation

Klassisk mus-interaktion har begränsingen att bara en punkt kan aktiveras på skärmen i taget. Interaktionen med de flesta applikationer blir, på grund av detta, sekventiell vilket kan vara en begränsning eftersom människan är van att utforska problem simultant med båda händerna. I vissa situationer är det önskvärt att kunna ha flera simultana punkter att arbeta med då detta exempelvis kan ge snabbare interaktion [32].

Om ett system kan detektera flera simultana pekpunkter öppnas nya möjligheter för vilka operationer som kan utföras på systemet [32], exempel på detta kan vara att man kan vill kunna reglera flera reglage samtidigt. Dessutom öppnar det upp möjlighet för flera personer att samtidigt, beroende eller oberoende av varandra, använda systemet, vilket kan vara användbart vid samarbete på en och samma indata-enhet.

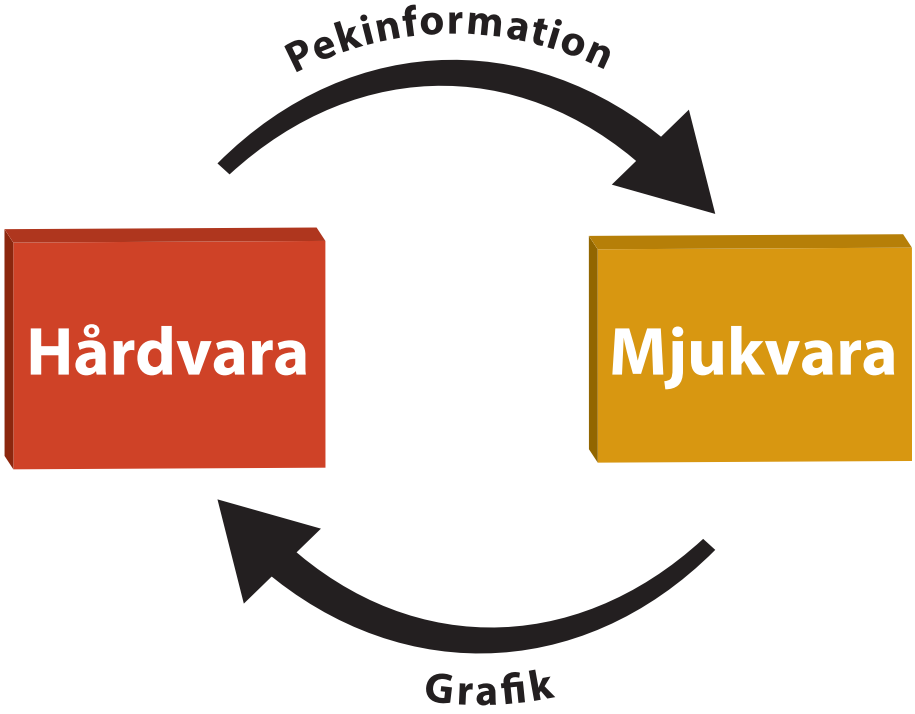
Målet med detta examensarbete är att skapa en mjukvara för flerpunktspekskärmar som skall kunna användas som grundstomme för vidareutveckling och på

så sätt kunna täcka fler behov. Mjukvaran måste kunna ta hand om flera samtidiga pekpunkter och utföra olika operationer beroende på pekpunkters rörelse individuellt eller i förhållande till varandra. En hårdvaruenhet, en flerpunktspekskärm, kommer också att byggas för att kunna tillämpa funktionaliteten i applikationen.

1.2 Primära resultat

Två prototyper har tagits fram, en hårdvaruprototyp och en mjukvaruprototyp. Figur 1.1 visar en schematisk bild över hur delarna i systemet samverkar.

Hårdvaruprototypen som tagits fram är byggd utifrån komponenter som går att köpa i detaljhandeln och konstruktionen har kostat under 3000 SEK. Hårdvaran levererar information till mjukvaran om var pekpunkter befinner sig på skärmytan.



Figur 1.1: Schematisk översikt över systemets uppbyggnad

Mjukvaran som byggts har inbyggt stöd för tre operationer på grafiska objekt. Dessa är rotation, skalning och förflyttning. Det är också möjligt att introducera nya sorters objekt som har stöd för andra operationer utan att skriva om mjukvarans stomme. Mjukvaran har i grunden stöd för att visa text, bilder och video vilka också kan manipuleras med de inbyggda operationerna. Andra typer av media kan också introduceras i och med att det går att införa nya typer av objekt.

1.3 Jämförelse med existerande produkter

Några implementationer för flerpunktspekskärmar har granskats under förstudien och exempel på granskade implementationer är mjukvara medföljande olika ramverk såsom Touché. Dessa implementationer fokuserar på att lösa en specifik uppgift, exempelvis att rotera bilder.

Till skillnad från de granskade implementationerna är implementationen utförd i detta arbete byggd för att kunna anpassas efter det behov som finns för en specifik implementation. På detta sätt blir applikationen en flexibel grundstomme och kan användas inom flera olika områden.

1.4 Denna uppsats disposition

Den här uppsatsen består av sju kapitel. I kapitel 2 introduceras begreppet flerpunktspekskärmar och en kort historik kring datainmatning ges. Kapitel 3 ger en överblick av vilka delar som måste utredas samt vilka olika lösningar som är potentiellt möjliga. Några av de potentiella lösningar som finns i kapitel 3 väljs ut och deras implementation förklaras i kapitel 4 och 5.

Prototypkonstruktionen har delats upp i två kapitel för att göra beskrivningen mer överskådlig. Kapitel 4 handlar enbart om den hårdvara som konstruerats och hur denna fungerar. Samma sak gäller för mjukvaran, som beskrivs i kapitel 5.

I kapitel 6 presenteras resultat från utvärderingen av både hårdvaruprototypen och mjukvaruprototypen. Utvärderingen baseras på de krav som finns presenterade i kapitel 3. Slutsatsen för uppsatsen återfinns i kapitel 7.

Kapitel 2
Bakgrund

Pekskärmar har fördelen att de erbjuder direkt manipulation då en användare arbetar med sina händer direkt på skärmen. En ovan användare har då lättare att koordinera händerna och synintrycken än vid användande av tangentbord och mus [38][39]. Till skillnad från traditionella pekskärmar har flerpunktspekskärmar möjlighet att ta emot flera simultana pekpunkter vilket i sin tur innebär att nya sorters operationer kan utföras [32]. Flerpunktsinmatning är ingen ny teknik men det är först de senaste åren som den har börjat användas i olika applikationer [39]. Vissa mobiltelefoner [11] och datorer utrustas idag med flerpunktspekskärmar av olika slag med varierande användningsområden.

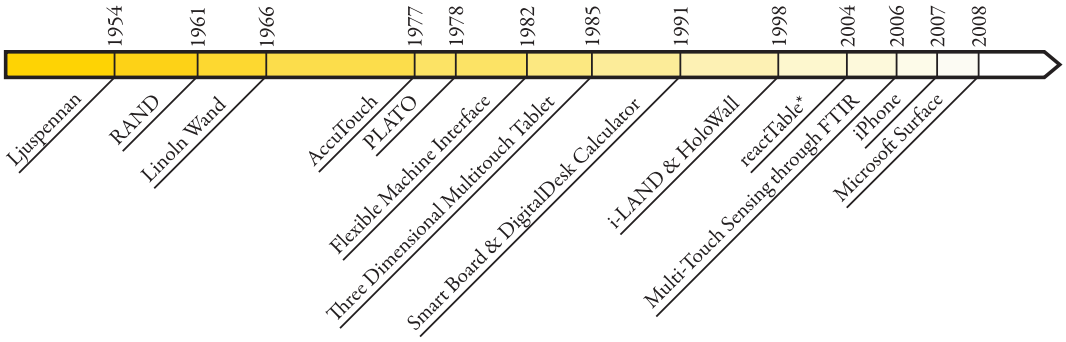
I det här kapitlet ges i avsnitt 2.1 en kort historik över olika inmatningsmetoder, relaterade till pekskärmar och flerpunktspekskärmar. I avsnitt 2.2 beskrivs hur pekskärmar och flerpunktspekskärmar används och en sammanfattning för det här kapitlet finns i avsnitt 2.3.

2.1 Historia

I detta avsnitt presenteras några historiska hållpunkter inom områden som kan relateras till pekskärmar och flerpunktspekskärmar. Figur 2.1 visar en tidsaxel där de prototyper och produkter som tas upp är utplacerade.

Först i detta avsnitt kommer inmatningssystem relaterade till pekskärmar att

presenteras och efter detta presenteras inmatningsystem relaterade till flerpunktspekskärmar.



Figur 2.1: Historisk tidslinje för alternativa inmatningssytem

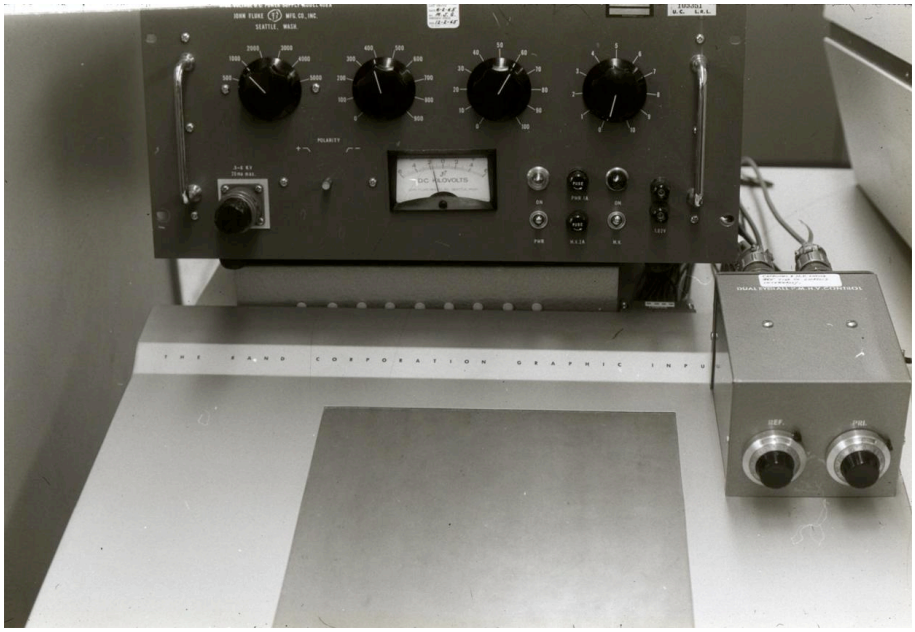
2.1.1 En-punktsinmatning

1954 introducerades ljuspennan som inmatningsmetod [30]. Med ljuspennan arbetade användaren direkt på skärmen med en för ändamålet utvecklad penna. Pennan hade en inbyggd ljussensor som kände av när CRT-skärmens katodstråle träffade densamma. Katodstrålens position är känd och utifrån denna information kan man härleda pennans position i två dimensioner.

THE RAND TABLET (se figur 2.2) lanserades 1961 [5] och hade en 10 gånger 10 tum stor yta vilken kunde läsa diskreta punkter i planet. Ytan bestod av en elektriskt laddad platta. På ytan användes en speciell penna (sond) för att göra inmatningar. Pennan hade en tryckkänslig spets för att simulera en verklig penna och på så sätt mata in data i tre dimensioner. Då inmatning och utmatning var fysiskt separerade kan THE RAND TABLET främst liknas vid dagens ritbord[1] [6].

1966, fem år efter RAND, utvecklades en in-dataenhet kallad THE LINCOLN WAND [25], där användaren kunde peka direkt på skärmen. Fortfarande

[1]Engelska: Tablet



Figur 2.2: THE RAND TABLET

användes en speciell penna för att göra kopplingen mellan pekpunkten på skärmen och datorn. Pennan var utrustad med en ultraljudsmottagare och dess position gick att utläsa i tre dimensioner, till skillnad från ljuspennans två dimensioner. Tekniken fungerade, i kort, så att skärmen hade fyra stycken inbyggda ultraljudssändare som periodiskt sände ut en ultraljudssignal. När användarens penna tog emot ljuden från sändarna kunde avståndet beräknas till varje sändare. På så sätt kunde positionen för pennan bestämmas.

Elva år senare, 1977, tillverkades en transparent yta, kallad AccuTouch^{TM}[21], som kunde avläsa en punkt baserat på fingrets placering på skärmen. Systemet bestod av två lager av elektriskt ledande material. När ett finger tryckte på ytan slöts en elektrisk krets mellan de två lagren. Fingrets position beräknades genom att mäta potentialen, horisontellt och vertikalt, i punkten [8].

Året efter AccuTouch, 1978, användes en pekskärm i en utbildningsprodukt vid namn PLATO [9] (se figur 2.3). PLATO hade en plasmaskärm om 512×512

bildpunkter[2] som dessutom var utrustad med möjligheten att ta emot data genom att placera ett finger på skärmen. En användare kunde på så sätt göra val i olika applikationer utan att använda tangentbordet. Pekskärmen hade en upplösning på 16×16 punkter vilket innebar att det var möjligt att känna av 256 olika områden på skärmen. Runt skärmen placerades infraröda ljuskällor och mottagare. Dessa skapade ett rutnät av ljusstrålar över skärmen. När ett finger sattes på skärmen bröts två ljusstrålar och detta gav fingrets position som vidarebefordrades till applikationen [2].

Figur 2.3: PLATO 1978

Från 1980-talet och framåt började pekskärmar leta sig in i allt fler kommersiella produkter [36]. Till exempel lanserade Hewlett-Packard kommersiella ar-

[2]Engelska: Pixels

betsstationer för personligt bruk vilka hade en inbyggd pekskärm, tekniken som användes var samma som den i PLATO.

Under nittiotalet, närmare bestämt 1991, presenterades Smart BOARD™. Smart BOARD, är en interaktiv whiteboard som är menad att användas i utbildningssamanhang.

2.1.2 Flerpunktsinmatning

Redan 1982 togs en prototyp fram för att kunna känna av flera simultana inmatningspunkter från en eller flera användare. Prototypen fick namnet The Flexible Machine Interface [36] och fungerade genom bildinläsning. En yta läses av med en kamera och fingrarna uppträder som svarta punkter som kameran kan avläsa.

Tre år senare, 1985, presenterades en yta med möjlighet att ta emot flera samtidiga punkter och som dessutom talar om graden av kontakt [26]. Ytan, som var indelad i 64 gånger 32 punkter, tillät att användaren arbetade direkt med fingrarna på ytan. Avläsningen skedde genom att ytan periodiskt skannades av med hjälp av en algoritm där punkter med lägre elektrisk laddning upptäcks. Ytan rapporterade plankoordinater samt hur hårt användaren tryckte på punkten. Detta gav tredimensionell inmatning.

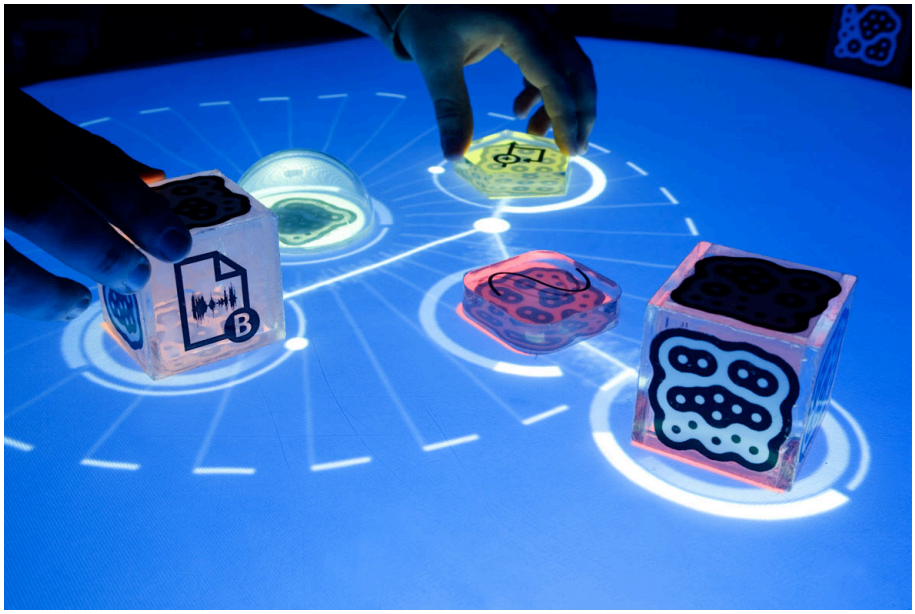
The DigitalDesk Calculator [43] presenterades 1991. Enheten projicerade ett gränssnitt på toppen av ett bord. Bredvid projektorn fanns även en kamera som registrerade användarens rörelser över bordet och dessutom objekt på bordet. Systemet kunde ta emot flera samtidiga inmatningar och känna igen dokument placerade på bordet.

Sju år senare, 1998, presenterades projektet i-LAND [41] och som en del av projektet togs två enheter fram, DynaWall och InteracTable. DynaWall var en stor interaktiv whiteboard som hade möjligheten att ta emot information från flera samtidiga användare och InteractTable var ett bord runt vilket upp till sex

personer kunde stå. Bordet kunde ta emot in-data från flera samtidiga användare.

Samma år presenterades även HoloWall [28]. HoloWall var en vägg designad för att kunna ta emot fler samtidiga inmatningar men även känna igen föremål som hölls upp framför ytan. HoloWall fungerade genom att en yta projicerades bakifrån och från samma håll belyses även ytan med ett infrarött ljus. Vid projektorn fanns en kamera som filmade ytan och det infraröda ljuset som belös ytan. Kameran kunde registrera det infraröda ljuset som reflekteras och på så sätt räkna ut pekpunkter.

2004 presenterades en fungerande prototyp av reactTable*[24] (se figur 2.4). reactTable* var ett interaktivt musikinstrument som hade stöd för att kunna känna igen flera fysiska objekt placerade på ytan samtidigt. Objekt identifierades optiskt med hjälp av en kamera och märkningar på varje objekt. Utifrån objekten, dess riktning och deras inbördes förhållande genererades musik [22].



Figur 2.4: reactTable*

De tre åren 2006, 2007 och 2008 innebar stora språng på marknaden för flerpunktspekskärmar. 2006 presenterade Jefferson Y. Han en flerpekpunktsyta som

kunde konstrueras både enkelt och icke kostnadskrävande [10]. Han presenterade sina idéer på konferensen TED (Technology, Entertainment, Design)[42] vilket fick världen att öppna ögonen för vad som kan åstadkommas.

Apple® iPhone™ [11] (se figur 2.5) lanserades i början av 2007 och är en telefon med en inbyggd flepunktspekskärm. Flerpunktspekskärmen i iPhone används exempelvis för att navigera webbsidor [12], titta på bilder, spela spel med mera. Apple tillhandahåller även en utvecklingsplattform för de som vill utveckla applikationer till telefonen och med denna går det utveckla applikationer som kan utnyttja flerpunktspekskärmen [13].

Figur 2.5: iPhone 3G

2008 levererades det första exemplaret av Microsoft® Surface™ [4] (se figur 2.6). Microsoft Surface fungerar genom att en bild projekteras bakifrån på en yta.

Ytan belyses även med infrarött ljus från projektorns håll. Kameror på undersidan registrerar ytan och kan registrera pekpunkter från flera samtidiga användare och dessutom känna igen objekt. Microsoft tillhandahåller ett utvecklingsverktyg för Surface vilket endast är tillgänglig för utvalda samarbetspartners [3].

Figur 2.6: Microsoft Surface

2.2 Pekskämar och flerpunktspekskärmar idag

Marknaden för flerpunktspekskärmar börjar idag ta fart. Microsoft® Surface™ finns idag i ett antal AT&T-butiker [4] och Apple säljer iPhone framgångsrikt [14].

Utöver dessa produkter är interaktiva whiteboards populärt i utbildningssammanhang. Här är det dock främst traditionella pekskärmar som används och inte flerpunktspekskärmar.

När det gäller flerpunktspekskärmar levererar ett företag vid namn N-trig ytor för detta. N-trig levererar ytor till bland annat Dell och Intel [31] som implementerar dessa i sina datorer.

Tekniken som kommer att användas för att ta fram en hårdvaruenhet i detta arbete liknar den som återfinns i Microsoft Surface.

2.2.1 Möjliga tillämpningar

Flerpunktspekskärmar har flera potentiella användningsområden. De skulle exempelvis lämpa sig bra som interaktiva whiteboards då de har möjlighet att ta emot inmatningar från flera samtidiga användare. Detta gör att flera personer samtidigt kan använda tavlan. En interaktiv whiteboard ger dessutom möjlighet att kunna lagra den information som matats in och på så sätt göra informationen på tavlan bestående.

Från whiteboard är steget inte långt till anslagstavlor. Det skulle kunna gå att tillverka stora anslagstavlor där flera personer samtidigt kan läsa och sätta upp digitala anslag. Sådana anslagstavlor skulle kunna användas på exempelvis företag, skolor eller i köpcentrum. Det skulle också vara tänkbart att koppla samman anslagstavlor och på så sätt ge användarna en möjlighet att snabbt publicera sitt anslag på flera fysiska platser.

I presentationssammanhang finns också tänkbara användningsområden. I tvsändningar görs analyser av sporthändelser och valsiffror samt väderkartor presenteras. Exempelvis skulle en väderkarta inte behöva vara animerad i förväg för att visa närmare vyer av områden. Presentatören skulle i stället kunna navigera i kartan med bara händerna.

Tekniken lämpar sig också inom design. Det är mer naturligt att arbeta med två händer[32] och därmed går det också både enklare och fortare att utföra uppgifter så som att skapa kurvor. Exempel på en teknik som passar in här är “Ta-

peDrawing", där man använder båda händerna för att forma kurvor med hjälp av tejp. Man slipper då problemet med att konvertera de analoga ritningarna till digitalt senare.

Sist men inte minst finns stor potential för flerpunktspekskärmar inom musik- och medie-industrin. Istället för att ha exempelvis en mixer med fysiska kontroller skulle denna kunna vara i en dator. Flerpunktspekskärmen kan då användas för att justera olika reglage på mixerbordet och dessutom kan flera reglage regleras på en och samma gång. Fördelen här är att mixerbordet kan anpassas i minsta detalj efter vad som för tillfället behövs och reglage som inte används kan döljas.

2.3 Sammanfattning

Flerpunktspekskärmar ger ett antal fördelar i jämförelse med pekskärmar. Bland annat låter de användaren använda flera händer och fingrar när denne utforskar ett problem och dessutom låter de flera personer samtidigt interagera med ett system. Flerpunktspekskärmar passar inte i alla situationer men kan dock med fördel användas i flera specifika tillämpningar.

Pekskärmar är ingen nyhet på marknaden och så inte heller flerpunktspekskärmar. Det har funnits varianter av pekskärmar sedan 1954 och redan från början har de använts för att mata in data till olika applikationer. Det har också i cirka 20 år funnits pekskärmar som kan ta emot flertalet in-datapunkter trots att det endast är på senare år de har börjat hitta in i olika kommersiella produkter.

Kapitel 3
Potentiella lösningar

I det här kapitlet diskuteras olika lösningsförslag för både hård- och mjukvara till prototypen. I avsnitt 3.1 presenteras de krav som finns på prototyperna och därefter i avsnitt 3.2 presenteras lösningsförslag till hårdvaran. Mjukvaran och hur denna kan bli utbyggbar presenteras i avsnitt 3.3.

3.1 Krav på systemet

3.1.1 Hårdvara

Hårdvarans uppgift är att visa det grafiska gränssnittet genom att visa en bild från en videokälla. Hårdvaran måste också kunna användas för att detektera pekpunkter som befinner sig på dess pekyta och konstruktionen får inte kosta över 4000 SEK i tillverkning. Nedan presenteras en översikt av kraven:

1. Visa gränssnitt från videokälla
2. Användas för att detektera pekpunkter
3. Max 4000 SEK i tillverkingskostnad (inkl. moms)

3.1.2 Mjukvara

Mjukvaran skall kunna visa minst två objekt samtidigt. Ett objekt avser i det här fallet en bitmappsbild. Med hjälp av pekpunkter skall operationer kunna utföras

på objekten. I det här fallet är operationerna rotation, proportionell skalning och förflyttning. Mjukvaran skall också vara designad på ett sådant sätt att det möjliggör introduktion av nya objekt och operationer. De krav som ställs på mjukvaruprototypen är:

1. Minst två objekt skall kunna visas samtidigt
2. Rotationsoperation
3. Proportionell skalning
4. Förflyttning
5. Möjliggöra introduktion av nya lager

3.2 Hårdvara

3.2.1 Pekpunktsregistrering

För att flerpunktspekskärmen skall kunna användas måste samtliga pekpunkter som befinner sig på skärmen registreras av systemet (se krav 2 i avsnitt 3.1.1). Det finns ett flertal olika sätt att genomföra denna registrering på och i denna uppsats utreds kapacitiv, optisk och sensorbaserad registrering vilka beskrivs i tur och ordning.

3.2.1.1 Kapacitiv pekpunktsregistrering

Genom historien har flera olika produkter använt sig av kapacitiv avläsning av pekpunkter [26] [21]. Metoden fungerar genom att mäta förändringar i kapacitans på en yta och på så sätt generera pekpunkter. Ett inköp eller utvecklande av en sådan metod ligger dock ej inom varken budget eller tidsrymd för detta arbete. Anledningarna till detta är att tekniken är såpass avancerad att det kräver myc-

ket tid och ekonomiska resurser att utveckla en sådan teknik och att ett inköp av en färdig produkt är utanför budget. Därför kommer detta alternativ inte att behandlas vidare.

3.2.1.2 Optisk pekpunktsregistrering

Ett annat sätt att läsa av aktiva pekpunkter på skärmen är genom optisk avläsning.

Detta fungerar genom att en kamera monteras så att den filmar ytan som skall avläsas. Avläsning av ytan kan lämpligen ske på motsatt sida användaren för att användarens händer och armar då inte är i vägen för avläsningen. Det enda som då syns för kameran är de punkter som är nära skärmens yta eller i kontakt med ytan.

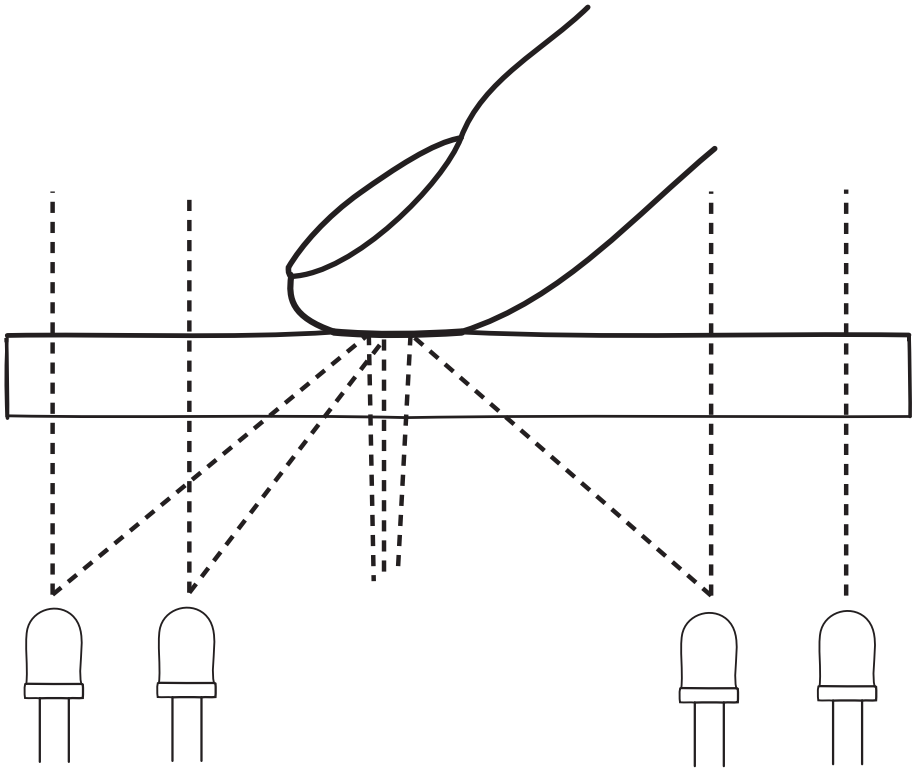
För att kameran tydligt skall kunna se pekpunkter från motsatt sida användaren måste dessa framhävas, exempelvis genom belysning. För att inte störa skärmens bild bör ett, för ögat, osynligt ljus användas för detta ändamål [28] [35]. Ett exempel på osynligt ljus är infrarött ljus, vilket återfinns i exempelvis fjärrkontroller till hemelektronik.

Hur upplysningen av pekpunkterna sker kan varieras och flera metoder presenteras nedan.

Utspridd upplysning

Ett sätt att lysa upp punkterna är med en teknik som kallas utspridd upplysning[1]. Ytan som skall registreras belyses underifrån med infrarött ljus [28]. När ett finger placeras på ytan kommer detta att reflektera ljuset nedåt så att detta kan registreras av kameran (se figur 3.1).

[1]Engelska: Diffused Illumination



Figur 3.1: Utspridd upplysning

Frustrerad total intern reflektion

Ett annat sätt att belysa pekpunkter är via så kallas frustrerad total intern reflektion[2] (FTIR) [10].

Tekniken går ut på att man sänder in infrarött ljus i en plexiglasskiva[3]. Ljuset sänds in med en vinkel lika med eller större än den så kallade *kritiska vinkeln* för total reflektion. När ljuset försöker bryta sig ut från plexiglasskivan kommer det att träffa på luft som har ett lägre optiskt brytningsindex än plexiglas. Detta gör att allt ljus kommer att reflekteras och stanna kvar inuti plexiglasskivan.

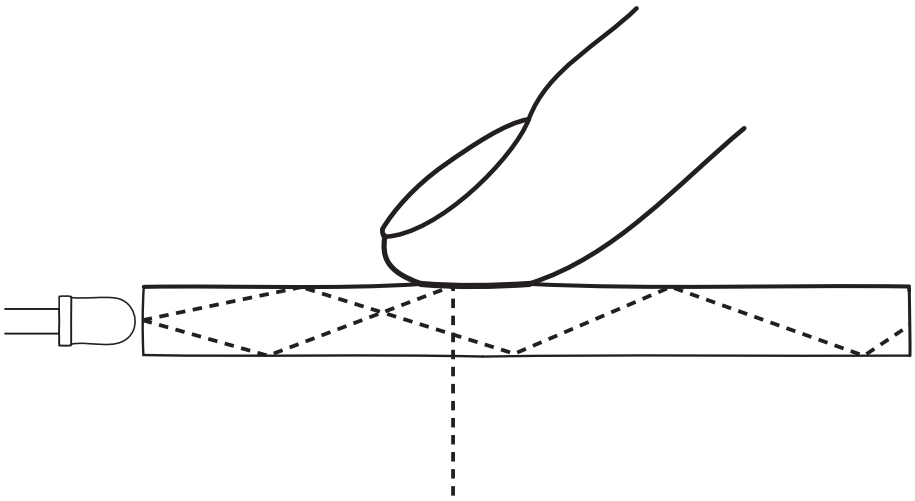
Den *kritiska vinkeln*, θ_c, då total reflektion uppstår kan härledas med hjälp av Snells lag [33] (se ekvation 3.1). Total reflektion uppstår endast när refraktionsvin-

[2]Engelska: Frustrated Total Internal Reflection
[3]Polymetylmetakrylat

keln är större än 90 grader. I ekvationen är n_1 brytningsindex för plexiglasskivan och n_2 brytningsindex för luft.

$$n_2 sin\theta_c = n_1 sin90^\circ \Leftrightarrow \theta_c = arcsin\left(\frac{n_1}{n_2}\right) \tag{3.1}$$

När ett finger sätts mot plexiglasskivans yta medför det en förändring av den kritiska vinkeln vid det aktuella området. Detta innebär att ljuset kommer att lämna plexiglaset och reflekteras mot fingret (se figur 3.2)

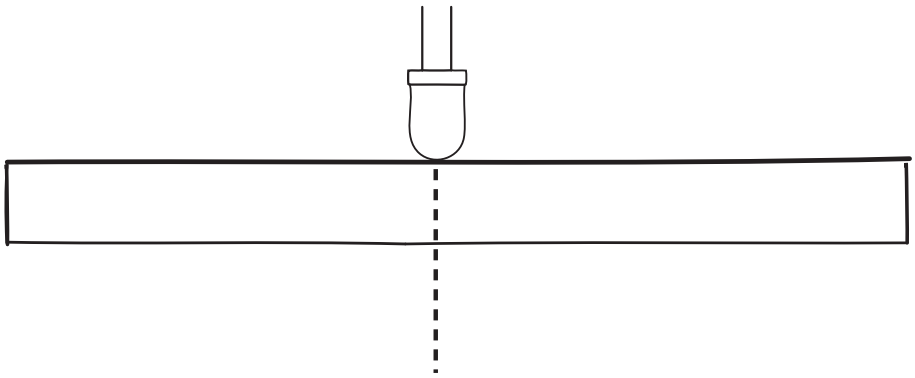


Figur 3.2: Frustrerad total intern reflektion

Penna

Eftersom pekpunkter registeras med hjälp av en kamera är det möjligt att belysa kameran direkt med en infraröd ljuspunkt, istället för indirekt via reflektion. Detta är en säkrare metod eftersom ljuset träffar kameran med högre intensitet. Metoden går också att kombinera med de övriga metoderna för optisk avläsning utan att göra modifikationer i hårdvaran.

Genom att bygga in en infraröd lysdiod i en penna kan pennan användas för att peka på skärmen. Ljuset från pennans infraröda lysdiod kommer att uppfattas som en pekpunkt av kameran (se figur 3.3).



Figur 3.3: Lysdiodspenna

3.2.1.3 Sensorbaserad pekpunktsregistrering

Den sensorbaserade metoden går ut på att plattskärmen har inbyggda sensorer för att känna av ljus. När ett finger placeras på skärmen registreras detta av TFT-fotoceller inbyggda skärmens bildpunkter [7].

Denna metod kräver antingen att en hårdvara med tekniken köps in eller kan konstrueras. Detta alternativ kommer, på grund av dess komplexitet, inte att vidare behandlas i denna uppsats.

3.2.1.4 Val av lösning

Hårdvarulösningen har valts utifrån två aspekter. Den första aspekten är priset och den andra är tiden för att utföra jobbet. Båda dessa parametrar är ytterst

begränsade i detta projekt då det finansieras privat samt för att arbetet max kan uppta cirka 5 veckor då även mjukvara skall tas fram.

Den lösning som valts baseras på utspridd upplysning och är en optisk pekpunktsregistering med hjälp av en kamera.

3.2.2 Bildåtergivning

För att flerpunktspekskärmen skall fungera måste ut-data, i form av grafik, presenteras på den yta som registrerar pekpunkter (se krav nummer 1 i avsnitt 3.1.1). Det finns flera sätt att lösa detta på och det går att kombinera olika sorters bildåtergivningsmetoder med optisk pekpunktsregistrering.

För det här arbetet finns två tänkbara alternativ och dessa är digitalprojektor[4] och LCD-skärm[5], vilka diskuteras nedan.

3.2.2.1 Digitalprojektor

Ett möjligt bildåtergivningssätt är att använda sig av en digitalprojektor. En digitalprojektor monteras så att den projicerar en bild på en duk. Digtalprojektorn kan monteras i sitt ursprungliga skick men kan ibland behöva kompletteras med exempelvis speglar då avståndet för projicering kan bli för kort [10]. Vissa digitalprojektorer kan också behöva kompletteras med ett filter för att ta bort oönskat infrarött ljus från projektorns lampa [28] vilket annars skulle störa optisk pekpunktsregistrering (se avsnitt 3.2.1.2).

3.2.2.2 LCD-skärm (Plattskärm)

Ett alternativ till digitalprojektorer är LCD-skärmar, som i mindre skala är billigare. Ett problem med LCD-skärmar, vidare benämnda plattskärmar, är att de kan

[4]Engelska: Video projector
[5]Med eller utan TFT

behöva modifieras för att användas tillsammans med olika typer av pekpunktsregistrering.

Då optisk pekpunktsregistrering (se avsnitt 3.2.1.2) används måste skärmen kunna släppa genom ljus. Plattskärmen måste monteras isär så att bildpanelens bakstycke kan avlägsnas samt dess kretskort flyttas för att göra det möjligt att sända ljus rakt genom bildpanelen [29] (se avsnitt 3.2.1.2). Skärmen kan innehålla optiska filter vilka kan behöva avlägsnas för att kunna registrera pekpunkter korrekt.

3.2.2.3 Val av lösning

För att kunna välja mellan dessa två alternativ måste dess för och nackdelar vägas mot varandra. Detta görs i tabell 3.1.

Plattskärm	*Digitalprojektor*
+ Billig + Ofta hög upplösning	+ Bra bildkvalité + Ytan på vilken bilden projiceras kan vara av ett material som har god genomströmning av infrarött ljus
– Kräver modifiering – Plattskärmens konstruktion kan förhindra att optisk pekpunktsregistering används	– Dyr – Värmeutvecklande, extra ventilation kan krävas – Kan kräva långt projiceringsavstånd

Tabell 3.1: För och nackdelar för plattskärm och digitalprojektor

För bildåtergivning har en plattskärm valts eftersom det är det billigaste alternativet. I valet mellan videoprojektor och plattskärm är ekonomi i detta fall den styrande faktorn.

3.3 Mjukvara

Mjukvaruprototypen är en stor del av den pekpunktsbaserade inmatningen. Mjukvarudelen består av en spårare[6] samt en applikationsdel vilken innehåller ett grafiskt användargränssnitt och logik som tolkar informationen från spåraren. Dessa delar beskrivs nedan.

3.3.1 Spårare

När man använder optisk pekpunktsregistrering är den data man erhåller en sekvens av stillbilder föreställande pekytan. På denna yta uppträder de pekpunkter som senare måste detekteras. För tillämpningen är de erhållna bilderna oanvändbara i sitt ursprungliga format. Användbar information utläses ur bilden genom att behandla och tolka bilden, vilken är spårarens uppgift. En spårare kan ha flera olika funktioner men ska åtminstone lämna i från sig information om koordinater för pekpunkter.

3.3.1.1 Kommunikation – spårare och applikation

Applikationen måste få information om pekpunkter från spåraren. Hur spåraren tillhandahåller denna information varierar beroende på implementation.

API

Ett sätt att låta applikationen kommunicera med spåraren är via ett API. APIets utformning varierar mellan olika spårare och måste därför anpassas efter vilken spårare som används. Detta innebär att applikationen måste skrivas om för varje spårare som skall användas.

[6] Engelska: Tracker

TUIO

Ett annat kommunikationssätt är att använda sig av ett protokoll. TUIO [23] är ett protokoll framtaget specifikt i syfte att förmedla information om optiskt identifierade objekt. TUIO fungerar över valfritt transportprotokoll, men UDP används med fördel i ett IP-nätverk eftersom det är snabbare än TCP.

3.3.1.2 Olika spårare

Det finns flertal olika spårare som skulle kunna användas. Spårarna implementerar olika sätt att lämna i från sig data men gemensamt för de som studerats är att de implementerar protokollet TUIO (se avsnitt 3.3.1.1). I tabell 3.2 finns exempel på ett antal spårare som implementerar detta protokoll och fördelen med detta är att det enkelt går att byta mellan olika spårare.

Spårare
BBTouch
Touché
TouchLib

Tabell 3.2: Olika spårare som implementerar TUIO

3.3.1.3 Val av lösning

Kommunikation mellan spårare och applikation via TUIO har valts på grund av dess fördelar mot en API-baserad lösning. TUIO möjliggör ett byte av spårare utan att applikationen behöver skrivas om.

För detta arbete har spåraren *Touché* valts eftersom den har öppen källkod och en enkel konfiguration samt att den fungerar på Apple MacOS® X som även applikationen körs under (se avsnitt 5.3.1). Dock skulle vilken spårare som helst, som implementerar protokollet TUIO, kunna användas.

3.3.2 Applikation

Applikationens uppgift är att ta emot pekpunktsinformation och bearbeta denna information. Applikationen ritar upp gränssnittet på skärmen och innehåller logik för att reagera på pekpunktsinformation.

3.3.2.1 Uppritning av gränssnitt

En yta

En möjlig lösning är att varje objekt ritas direkt till skärmen. Objektet innehåller då den information som skall ritas till skärmen. När en operation utförs på ett objekt måste skärmen ritas om för att reflektera den ändring som operationen innebär. Denna lösning ger en mycket statisk funktionalitet då grundstrukturen måste implementera operationer specifikt anpassade efter varje enskilt objekt och den grafiska förändring som operationen innebär.

Flera lager

En annan möjlig lösning är att använda ett lager för *varje* objekt som skall visas på skärmen. Varje lager kan då exempelvis innehålla information om vad som skall ritas, dess position samt metoder för att tala om för lagret om omgivande händelser. Lagret har på detta sätt möjlighet att själv ändra sin grafiska representation vid operationer och skärmen ritas sedan om med information tillhandahållen av lagren. På detta sätt skapas ett lager för varje grafiskt objekt som skall ritas, vilket är bra ur ett expansions- och separeringsperspektiv.

3.3.2.2 Datahanteringslogik

Datahantering av dedicerad funktion

Ett sätt att hantera pekpunktsdata är att skapa en dedicerad funktion som innehåller alla de operationer som skall kunna utföras på objekt. När data om pekpunkter tas emot räknar denna funktion ut vilken operation som skall utföras och även vilket objekt som operationen skall utföras på.

Om gränssnittet ritas upp i en yta måste det först bestämmas vilka objekt som skall påverkas och för att göra detta krävs information om objektens utseende och position samt vilken pekpunkt som är aktuell. Utifrån denna information kan man sedan beräkna vilken operation som skall utföras och på vilka objekt denna skall utföras. När detta är klart uppdateras skärmbilden för att avspegla dessa operationer.

Om istället lager används låter man lagren själva avgöra om de innehåller pekpunkten då dessa vet sin position och sitt utseende. Efter detta beräknas vilken operation som skall utföras och därefter skickas en begäran om utförandet av operationen till lagret. Lagret utför operationen och skärmen uppdateras genom att begära information om lagrets nuvarande utseende.

Datahantering i lager

Om lager används kan ett annat sätt vara att applikationen tar emot pekpunkter från spåraren och översätter dessa till tre stycken olika meddelande-typer (se tabell 3.3). Dessa tre meddelande-typer kan sedan användas för att informera lagren om vilka pekningar som har utförts och lagren kan utifrån dessa själv avgöra vilka operationer som skall utföras. Detta ger en mer generell datahanteirngslogik där all specifik funktionalitet flyttats till lagren.

Ett *Aktivera pekpunkt*-meddelande kommer att orsaka en kollisionsdetektering mellan pekpunktens koordinat samt de olika lager som finns i systemet.

Typ	*Argument*
Aktivera pekpunkt	Pekpunktens identitet, koordinat
Flytta pekpunkt	Pekpunktens identitet, koordinat
Avaktivera pekpunkt	Pekpunktens identitet

Tabell 3.3: Meddelande-typer, internt i applikation

Om en kollision mellan pekpunkten och lagret uppstår kan meddelandet skickas vidare till detta lager. Dessutom lagras information om att denna kollision har uppstått.

När det gäller meddelandena *Flytta pekpunkt* samt *Avaktivera pekpunkt* så vet systemet redan vilket lager som är associerat med den aktuella pekpunktsidentiteten. Meddelanden skickas då vidare direkt till detta lager.

Det finns en stor fördel med att låta varje lager ta hand om inkomna meddelanden. Om lagret självt i stället hanterar meddelandena kan nya lagertyper introduceras utan att omgivande struktur måste modifieras.

3.3.2.3 Val av metod

Den metod som valts baseras på att objekt ritas i separata lager (se avsnitt 3.3.2.1), eftersom objekten på detta sätt blir separata entiteter vilket ger en mer modulär lösning.

Den logik som sköter datahanteringen ska ligga i varje lager, eftersom lagret på detta sätt kan introducera nya typer av operationer utan att omgivande programkod måste skrivas om, se motivationsexempel nedan.

Motivationsexempel

Låt ett lager ta emot meddelandet *Rotera X grader*. Låt vidare information om två pekpunkter skickas till en dedicerad funktion i systemet. Nu skall en beräkning utföras för att räkna ut hur många grader som lagret skall roteras och slutligen

skall meddelandet *Rotera X grader* skickas till lagret.

Introducera sedan ett nytt lager som inte skall kunna roteras men spela upp ett ljud. Ett sådan lager skulle kunna ta emot meddelandena *Spela* och *Stoppa*. Nu måste en del av applikationen skrivas om för att kunna veta vilka lager som svarar på vilka meddelanden samt lägga till kunskap för att veta om ett *Spela*- eller *Stoppa*-meddelande skall skickas.

Om all denna funktionalitet istället finns i lagret behöver applikationen enbart ta reda på vilket lager som skall ha ett visst meddelande och därefter vidarebefordra detsamma. Varje lager kan själv bestämma vad som skall hända när meddelandet tas emot.

Kapitel 4
Hårdvaruprototyp

En prototyp (se figur 4.1) har tagits fram för detta projekt enligt de val som gjorts i kapitel 3. Hårdvaruprototypen kommer att användas för att testa mjukvaruprototypen. I det här kapitlet kommer utförliga beskrivningar att ges över hur hårdvaruprototypen är konstruerad.

I avsnitt 4.1 presenteras konstruktionen för pekpunktsregistrering och i avsnitt 4.2 metoden för att skapa pekpunkter som kan detekteras. I avsnitt 4.3 ges information om bildåtergivningen och i avsnitt 4.4 beskriver chassits konstruktion. Sist i detta kapitel, i avsnitt 4.5, ges en sammanfattning.

4.1 Pekpunktsregistrering

Pekpunktsregistreringen sker optiskt med hjälp av en kamera (se avsnitt 3.2.1.2).

I avsnitt 4.1.1 presenteras kameran och modifikationer för denna, i avsnitt 4.1.2 beskrivs de beräkningar som gjorts för att bestämma kamerans placering och i avsnitt 4.1.3 beskrivs hur bilder hämtas in till mjukvaran.

4.1.1 Kamera

För att läsa in pekpunkter optiskt används en kamera. En webbkamera har inhandlats för ändamålet och modellen kan ses i tabell 4.1. Med fördel kan en kamera av bättre kvalité användas för att få snabbare överföring av bilder samt bätt-

Figur 4.1: Hårdvaran

re kvalité på bilderna. Detta skulle ge bättre resultat eftersom ursprungsbilderna skulle bli av en högre kvalité. Kameran (se figur 4.2) i detta projekt är konstruerad för 640×480 bildpunkters upplösning i 30 bilder per sekund.

Tillverkare	*Modell*	*Inköpsställe*
Microsoft	Xbox LIVE Vision	GameStop Sweden AB

Tabell 4.1: Kameramodell

Figur 4.2: XBOX Live Vision

4.1.1.1 Modifiering för infrarött ljus

För pekning används infrarött ljus (se avsnitt 4.2.1). Detta gör att kameran med fördel kan modifieras för att enbart registrera infrarött ljus, vilket i sin tur innebär att inget annat ljus kommer att störa kameran.

Filter

Ett filter, som enbart släpper igenom infrarött ljus, har monterats in i kameran. Bäst resultat fås genom att använda ett filter som enbart släpper genom ljus av den våglängd som används i ljuskällan (se avsnitt 4.2.1). Ett sådant filter är dock relativt dyrt och kan kosta över 100 USD [27].

I stället används fullt exponerat och framkallat negativ [34]. Fullt exponerat och framkallat negativ har den egenskapen att det blockerar synligt ljus men släpper fortfarande genom IR-ljus. Kameran monteras isär och filtret placeras framför

kamerans bildsensor. Efter testning framkom att *tre lager* av sådant negativ behövdes för att blockera så mycket synligt ljus som möjligt. Fler lager än tre innebar ingen märkbar skillnad för varken synligt eller infrarött ljus.

Bilder som tas med kameran kommer enbart att registrera infrarött ljus eller ljus som ligger mycket nära det infraröda ljusspektrumet. Allt annat ljus bör släckas ut av filtret.

4.1.2 Beräkningar

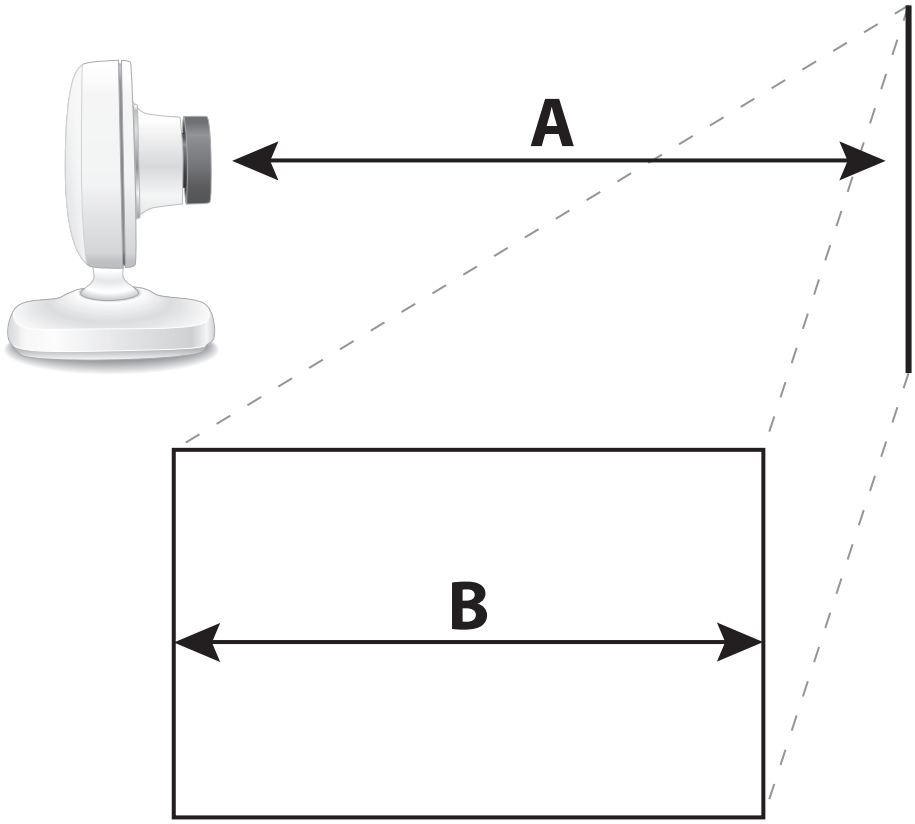
Kameran måste monteras inuti chassit om den skall kunna registrera pekpunkter underifrån, enligt 3.2.1.2. Ett lämpligt avstånd mellan kameran och bildskärmens yta måste bestämmas och för att göra detta måste först kamerans synfält beräknas. Avståndet mellan kameran och bildskärmen har använts för att beräkna chassits höjd.

4.1.2.1 Kamerans synfält

Det *exakta* avståndet mellan kameran och ytan som registreras är inte intressant, utan det viktiga är att man får ett avstånd så att åtminstone hela ytan registreras. I prototypen är det bra att ha lite utrymme för justeringar i efterhand samt möjligheten att korrigera för fel. Eftersom bildskärmen (se avsnitt 4.3) är i bredbildsformat[1] och kameran registrerar en nästan kvadratisk yta så är bredden på skärmen det intressanta.

Låt A vara avståndet från kamerans lins till ytan som skall registrera. Låt vidare B vara bredden på det område som kan registreras av kameran vid avståndet A. För att kunna ge chassits höjd är det nu intressant att veta förhållandet mellan A och B (se figur 4.3).

[1] Engelska: Wide screen



Figur 4.3: Kamerans synfält

Mätvärden

Tabell 4.2 visar olika mätpunkter samt förhållandet mellan A och B (se figur 4.3).

A (mm)	*B (mm)*	*A/B*
100	75	4/3
90	65	18/13
80	60	4/3
70	53	70/53
60	45	4/3
50	40	4/3
40	30	4/3
30	22	15/11

Tabell 4.2: Mätvärden för kamerans synfält

Som synes i tabell 4.2 kretsar alla värden kring $\frac{4}{3}$. Formel 4.1 erhålls från dessa värden.

$$A = \frac{4}{3}B \tag{4.1}$$

4.1.2.2 Chassits höjd

Det är intressant att veta hur långt från ytan som kameran måste sitta för att kunna registrera hela ytan.

Utifrån formel 4.1 kan vi härleda kamerans avstånd, A, eftersom B är känd från skärmens specifikationer.

$$\begin{cases} \mathrm{A} = \frac{4}{3}\mathrm{B} \\ \mathrm{B} = 485 \text{ mm} \end{cases} \Rightarrow \mathrm{A} = \frac{485 \cdot 4}{3} = \frac{1940}{3} \approx 647 \text{ mm}.$$

Detta ger att chassits höjd måste som allra minst vara 647 mm. Utöver detta tillkommer kamerans egen höjd samt den höjd som en monteringsanordning upptar.

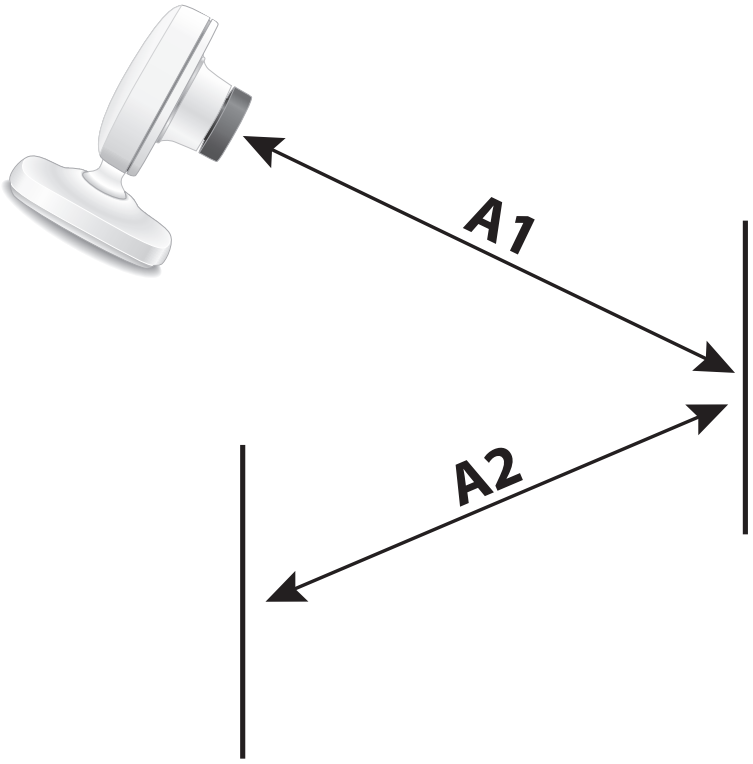
Minska chassits höjd

För att reducera chassits höjd används en spegel. Spegeln reflekterar inkommande ljus från pekpunkter och förlänger på så sätt avståndet mellan ytan och kameran.

Teoretiskt kan höjden på chassit halveras med hjälp av en plan spegel. Dock är detta inte praktiskt möjligt eftersom:

1. inget får skymma ytan och
2. kameran inte kan monteras exakt i toppen på chassit.

Kameran kan dock monteras på sidan av chassit och vinklas för att titta neråt. Om spegeln också vinklas kan en vy över toppen på lådan åstadkommas enligt



Figur 4.4: Kamerans synfält utökas genom en spegel

figur 4.4. Kameran kan antingen placeras på kort eller långsidan. Placeras den på kortsidan finns det fritt utrymme att justera den i höjdled. Placeras den däremot på långsidan så begränsas placeringen av ett kretskort tillhörande bildskärmen.

Att montera kameran på kortsidan kräver att en monteringsanordning byggs eftersom kameran måste monteras på högkant och kan då inte stå på sin fot (se figur över kamera 4.2).

Monteringskonstruktionen för kretskortet skapar en “naturlig hylla” där kameran kan monteras. Överkanten på denna “hylla” är 70 mm från toppen då den måste placeras intill skärmen. Kameran placerades på denna “hylla” för att inte behöva tillverka ovannämnda monteringsanordning för kameran.

647 mm är det minsta avståndet mellan kameran och ytan som skall registreras. Hälften av detta avstånd är $\frac{647\,\mathrm{mm}}{2} = 324$ mm. Minsta möjliga höjd är därmed 324 mm. Höjden på chassit beslutades till 400 mm för att ge möjlighet att montera kameran ca 70 mm nedanför toppen av chassit samt ge utrymme för justeringar i efterhand.

4.1.3 Bildinhämtning

Kameran anslöts till en dator med hjälp av en USB-kabel. En mjukvara kan då, via en kompatibel drivrutin, använda sig av kameran för att inhämta bilder.

Spåraren hämtar in bilder och bearbetar sedan dessa för att utläsa relevant information. Mer om denna procedur presenteras i avsnitt 5.1.

4.2 Pekpunkter

Då optisk pekpunktsdetektering används måste någon av de metoder som beskrivs i avsnitt 3.2.1.2 används för att skapa pekpunkter. Metoden baseras på utspridd upplysning som använder sig av optisk pekpunktsregistrering med hjälp av en kamera.

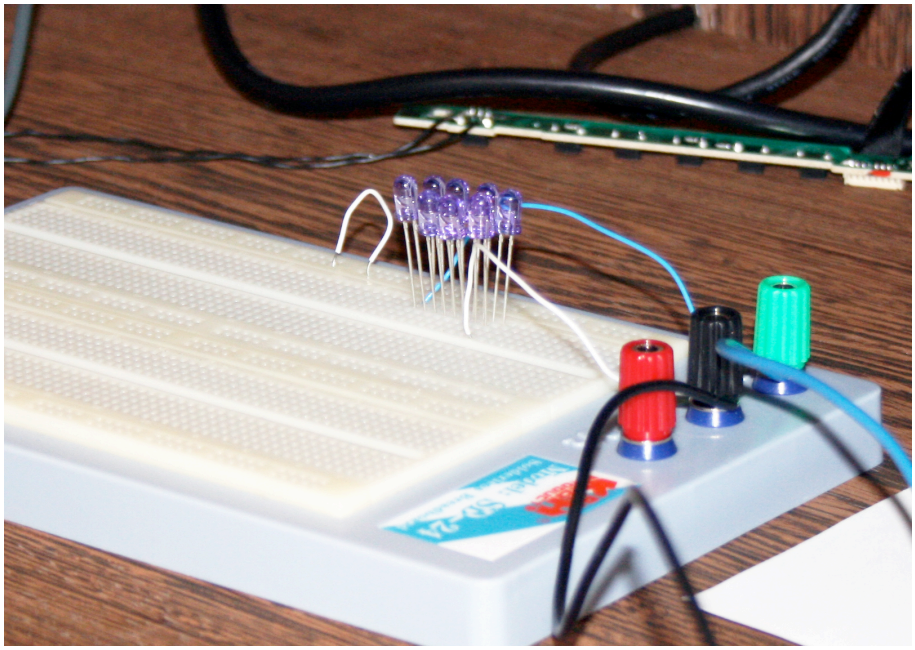
I avsnitt 4.2.1 beskrivs ljuskällan som används och i avsnitt 4.2.2 beskrivs de problem som uppstod med metoden. Sist, i avsnitt 4.2.3, beskrivs den alternativa metoden med infraröda ljuspennor.

4.2.1 Ljuskälla

Enligt avsnitt 3.2.1 är infrarött ljus en lämplig ljuskälla att använda för pekpunktsregistering. För denna prototyp har infraröda lysdioder som emmiterar ljus med en våglängd på 880 nm [34].

4.2.1.1 Utspridd upplysning

Bland beskrivna metoder i avsnitt 3.2.1.2 är tekniken med *utspridd upplysning* den teknik som är teoretiskt sett enklast att genomföra. Detta eftersom ett antal lysdioder enbart behöver monteras upp i ett jämnt mönster.



Figur 4.5: Utspridd belysning uppmonterad

Ett antal infraröda lysdioder monterades upp på en laborationsplatta som sedan placerades inuti chassit (se figur 4.5). Då undersidan av plattskärmens panel är blank uppstår reflektioner från ljuskällan och sådana reflektioner är svåröverrösta av reflektioner från pekpunkter (exempelvis fingrar). För att reducera reflektionen och ge en jämnare spridd belysning placerades ett tunt skisspapper under den blanka ytan. Reflektionerna reducerades dock marginellt.

4.2.2 Problem

4.2.2.1 Utspridd upplysning

Ett problem med den utspridda upplysningen var att det inte gick att med enbart fingrar överrösta de reflektioner som kvarstod efter att skisspappret hade monterats. Det gör det omöjligt att använda denna teknik i den aktuella konstruktionen. För att komma vidare måste en annan teknik användas.

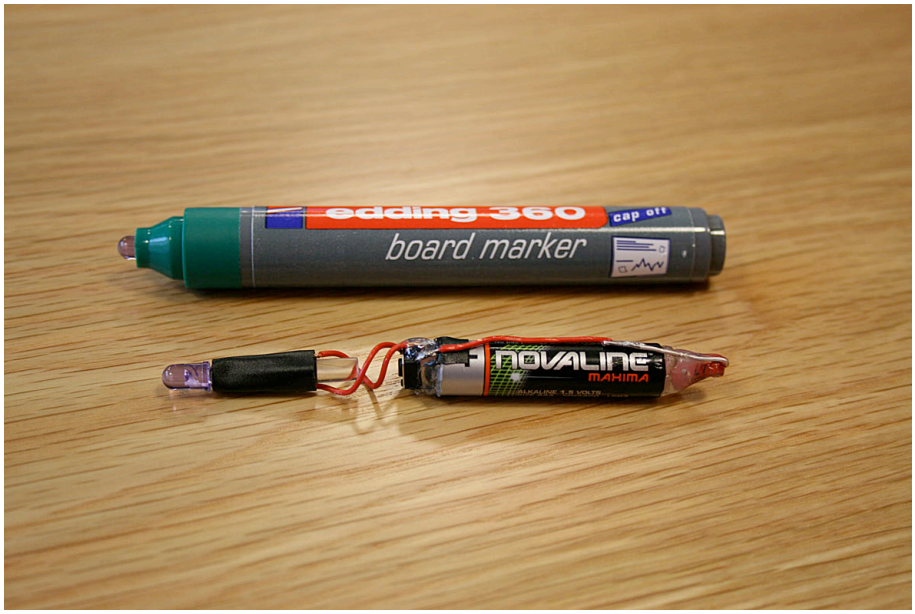
4.2.3 Infraröd ljuspenna

Som lösning på de problem som uppstått med pekpunktsinmatningen (se avsnitt 4.2.2) har två infraröda ljuspennor tillverkats. Pennorna har inbyggda infraröda lysdioder och emitterar själva infrarött ljus i stället för att reflektera, detta gör ljuspunkterna mycket starkare. En fördel med att använda dessa pennor är att de fungerar med alla metoder som beskrivits i avsnitt 3.2.1.2.

4.2.3.1 Uppbyggnad

Pennorna använder en whiteboard-penna som grundstomme. Inuti pennan finns ett AAA-batteri, en infraröd lysdiod, en brytare, ett motstånd, en plexiglasstav samt en metallbricka. Samtliga komponenter finns i tabell 4.3.

Lysdioden har placerats i pennans spets (se figur 4.6). När lysdioden trycks in sluts brytaren, med hjälp av plexiglasstaven, och dioden tänds. Detta medför att pennan kan användas för pekning genom att den enbart sätts ned mot ytan. När pennans spets inte är tryckt mot någon yta är lysdioden släckt vilket medför att pennan då inte registreras som en pekpunkt.



Figur 4.6: Pekpennas utseende och uppbyggnad

4.3 Bildåtergivning

4.3.1 Val av lösning

För bildåtergivning har en plattskärm (LCD) använts (se avsnitt 3.2.2.3).

Komponent	*Tillverkare och modell*	*Inköpsställe*
OH-penna	Edding 363	
Infraröd lysdiod	Osram SFH485-2	ELFA AB
Tangentbordsswitch		Kjell & Company AB
Batteri 1,5 Volt AAA	Duracell	Clas Ohlson
Motstånd, 48 Ohm		ELFA
Plexiglasstav		
Metallbricka		

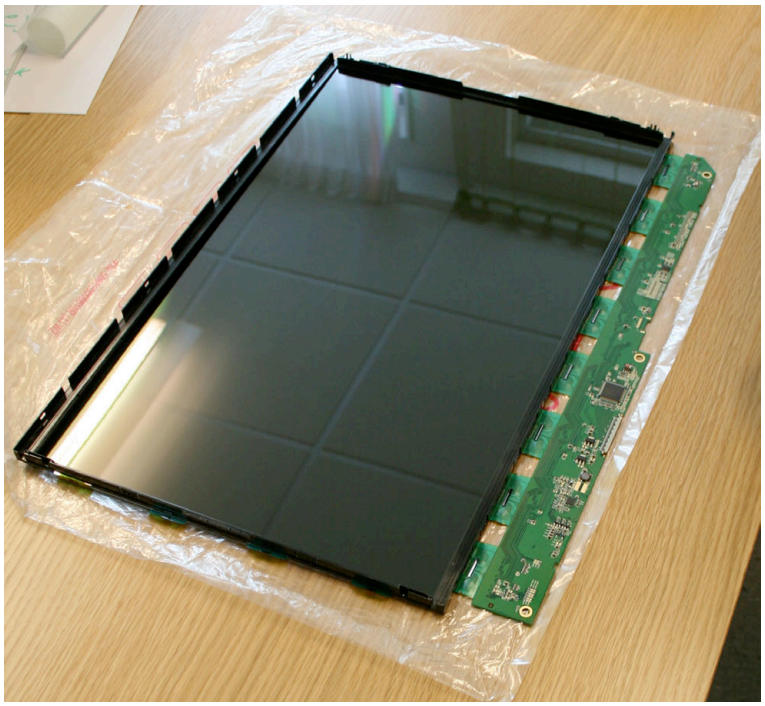
Tabell 4.3: Komponenter för konstruktion av infraröda ljuspennor

4.3.1.1 Plattskärm

Plattskärmen har ett antal icke önskvärda egenskaper, som dessutom kan variera från modell till modell. Eftersom skärmen måste kunna släppa genom infrarött ljus måste bakstycket avlägsnas och så även de kretskort som återfinns där.

Plattskärmen är uppbyggd av en bildpanel ytterst, ett antal filtrerande lager samt en bakgrundsbelysning. En del av de filtrerande lagern har sådana egenskaper att de påverkar ljusets genomflöde på ett sådant sätt att det är omöjligt att detektera pekpunkter. Dessa lager måste tas bort.

I figur 4.7 syns plattskärmens bildpanel utan filtrerande lager men det vita lagret (se figur 4.8) lämnas kvar. Utan detta får bilden på skärmen avsevärt sänkt kvalité. Detta lager släpper dessutom genom infrarött ljus och utgör därför inget större problem.



Figur 4.7: Plattskärmens bildpanel

Figur 4.8: Lager för att bland annat ge vit färg

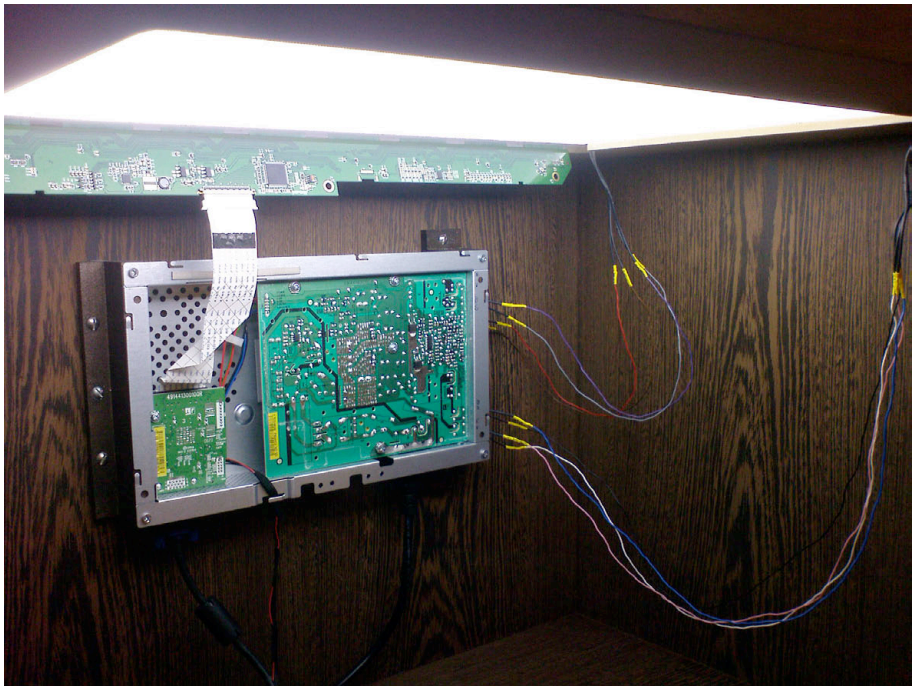
Plattskärmen har monterats i chassit (se avsnitt 4.4) tillsammans med bakgrundsbelysningen (se figur 4.9). Plattskärmens kretskort har monterats på chassits vägg, eftersom de inte får vara i vägen för den optiska bildinläsningen (se avsnitt 4.1.1).

Märke, modell och storlek

För ändamålet valdes den billigaste plattskärmen i storlek 22 tum som var tillgänglig. Information kring vilken skärm som används finns i tabell 4.4. Storleken 22 tum valdes utifrån att det är en vanligt förekommande skärmstorlek vilket gör det möjligt att få ett bra pris samt att det är en relativt stor skärm. Priset söktes med hjälp av webbsidan `http://www.prisjakt.se`.

Tillverkare	*Modell*	*Inköpsställe*
LG	W2234S	Nya Internetworking Data i Gbg AB (Inet)

Tabell 4.4: Plattskärmsmodell



Figur 4.9: Plattskärm monterad i chassi

4.4 Chassi

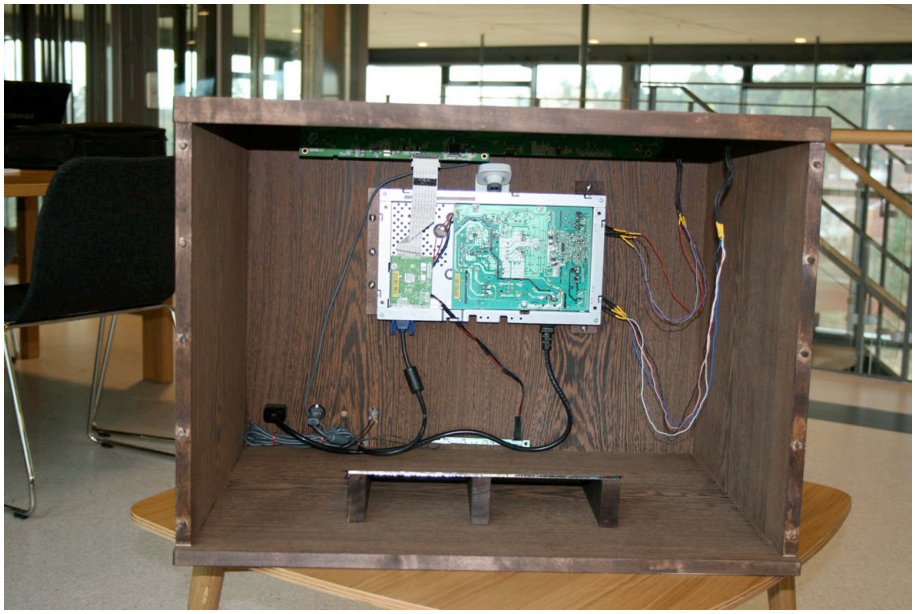
Ett chassi har konstruerats i trä. Chassit rymmer all hårdvara utom den dator som tar emot pekinformation och visar gränssnittet. Själva chassit har konstruerats av utomstående och ritningar över chassit finns i bilaga B.

4.4.1 Mått

Chassits inre höjd är 400 mm, vilket beslutades utifrån de beräkningar som gjordes i avsnitt 4.1.2.2. Bredden och djupet på chassit ges av plattskärmens storlek. Djupet på chassit måste dessutom utökas med bredden på kretskorten och kameran som skall monteras i bakkant. Chassits bredd är 645 mm och djupet är 475 mm.

4.4.2 Intern uppbyggnad

Inuti chassit sitter kontrollerkort för plattskärmen, en kamera, en spegel samt kablar för anslutning. En översiktsbild av chassits interna uppbyggnad finns i figur 4.10



Figur 4.10: Hårdvarans uppbyggnad internt

4.4.2.1 Plattskärm

Som tidigare nämnts används en plattskärm (se avsnitt 4.3) för bildåtergivning. Plattskärmen är monterad i chassits topp med sin bildpanel uppåt. Plattskärmen har ett antal kretskort som inte får vara monterade så att de blockerar kamerans synfält. Dessa kretskort har därför monterats mot chassits bakre vägg (se figur 4.9).

4.4.2.2 Glasskiva

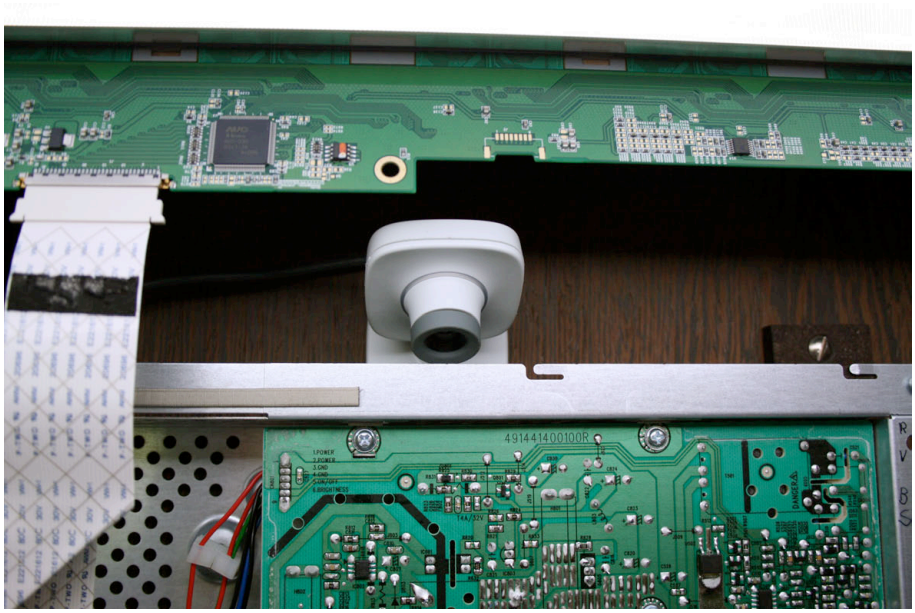
På toppen av chassit har en glasskiva monterats. Syftet med denna skiva är att förhindra att fingrar eller pennor används direkt på plattskärmens bildpanel och på så sätt skadar denna. Specifikationer för glasskivan finns i tabell 4.5.

Typ	*Inköpsställe*
6 millimeter tjock glasskiva	Karlstads Nya Glasmästeri AB

Tabell 4.5: Glasspecifikation

4.4.2.3 Kamera

Kameran har monterats på "hyllan" ovan plattskärmen kretskort (se figur 4.11). Detta ger den ett lämpligt avstånd till spegeln och därmed ytan som skall registreras.



Figur 4.11: Kameran är monterad ovan plattskärmens kretskort

4.4.2.4 Spegel

En plan spegel har monterats i botten av chassit. Dess uppgift är att förlänga det optiska avståndet mellan kameran och ytan som skall registreras. Spegelns specifikationer finns i tabell 4.6.

Typ	*Inköpsställe*
3 millimeter tjock plan spegel	Karlstads Nya Glasmästeri AB

Tabell 4.6: Spegelspecifikation

4.4.3 Anslutningar

Chassit har anslutningar för elektricitet, USB samt en D-Sub-15. Electricitetskontakten används för att ge skärmen ström (se figur 4.12). Chassit har även ett intag för USB där en USB-kabel (Typ A till Typ B) kan anslutas mellan datorn och chassit. Internt är kameran ansluten till denna kontakt via en USB Typ A (se figur 4.12).

Figur 4.12: T.v intag till chassit och t.h. intern koppling

För att minimera risken för störningar i signalen till plattskärmen finns inget intag för videosignal monterad på chassit. I stället har videosignalskabeln dragits genom chassit. Detta ger möjligheten att utan skarvning ansluta skärmen direkt till den dator som skall visa användargränssnittet.

4.4.4 Start och avstängning

För att kunna starta plattskärmen måste det gå att nå den inbyggda start- och avstängningsknappen. För att göra detta möjligt har en knapp monterats vid anslutningarna på chassits baksida, knappen syns i figur 4.12. Denna knapp har internt kopplats över den befintliga start- och avstägningsknappen på plattskärmen.

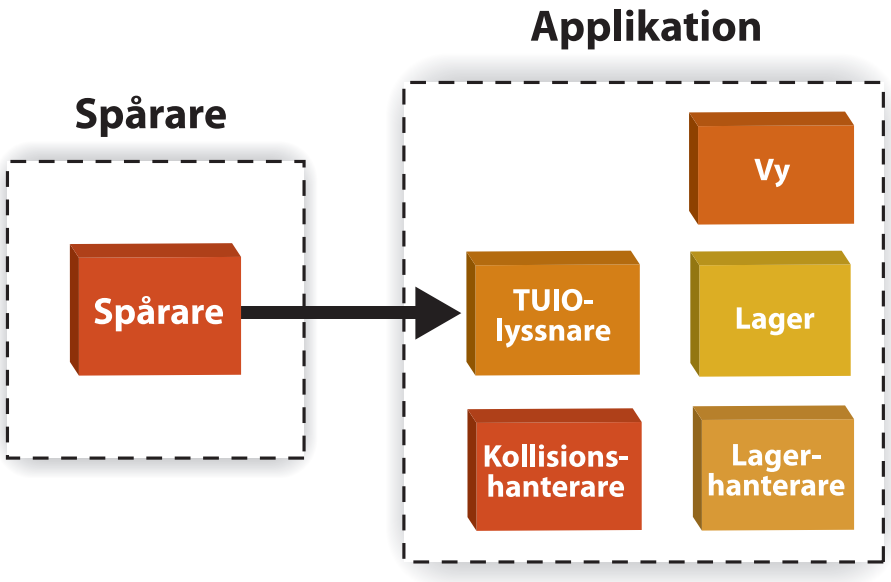
4.5 Sammanfattning

En hårdvaruprototyp har konstruerats av en plattskärm, en kamera samt infraröda lysdioder. Initialt var planen att använda utspridd upplysning (se avsnitt 3.2.1.2) men detta misslyckades och istället skapades ett antal pennor med infraröda lysdioder som kan användas för pekning.

Bildåtergivningen sker med hjälp av en plattskärm vilken har monterats, tillsammans med kameror och speglar, inuti ett chassi. På toppen av chassit finns en glasskiva som skall förhindra användaren från att skada plattskärmens bildpanel.

Kapitel 5
Mjukvaruprototyp

I detta kapitel beskrivs mjukvaruprototypens konstruktion i detalj. I figur 5.1 syns en översikt över mjukvaruprototypen och som kan ses består mjukvaruprototypen av två separata delar, *Spårare* och *Applikation*. Till vänster syns spåraren (se avsnitt 5.1) som skickar information till den applikation som har implementerats (se avsnitt 5.3), till höger i figuren.



Figur 5.1: Översikt över mjukvaruprototyp

I avsnitt 5.1 presenteras hur spåraren fungerar och i avsnitt 5.2 beskrivs kommunikationen mellan spåraren och applikationen. Avsnitt 5.3 behandlar applika-

tionsdelen av prototypen och hur denna är konstruerad. I avsnitt 5.4 sammanfattas detta kapitel.

5.1 Spårare

Hårdvaruprototypen som presenterats i denna uppsats använder optisk pekpunktsregistrering (se avsnitt 4.1). Den information som erhålls är en sekvens av stillbilder. För att tolka dessa bilder och utifrån dessa generera pekpunkter behövs en så kallad spårare.

I denna prototyp används spåraren Touché (se avsnitt 3.3.1.3) men avsnitten som följer är generell för de flesta spårare.

5.1.1 Funktionalitet

Bilden innehåller en mängd information men enbart en delmängd av denna information är intressant för tillämpningen och exempelvis är färginformation inte intressant. Genom att applicera lämpliga filter på bilden gallras irrelevant information bort. Den intressanta informationen har olika utseende beroende på vilka förhållanden som råder när bilden tas. På grund av detta behövs olika filter vid olika förhållanden.

Efter filtrering skall spåraren tolka och analysera den kvarvarande informationen. Spåraren måste identifiera pekpunkter och följa dessa mellan olika bilder i sekvensen. Till sist lämnar spåraren ifrån sig information om de aktiva pekpunkterna och deras position.

5.1.1.1 Filtrering

Filter är matematiska funktioner som appliceras på bilden och ger den ett nytt utseende. Nedan beskrivs några exempel på filter som kan användas och utav dessa

användes bakgrundssubtraktion, oskärpa samt kontrast i mjukvaruprototypen. Invertering används inte i prototypen på grund av att pekpunkter uppträder som ljusa punkter mot mörk bakgrund vilket är det önskvärda beteendet.

Bakgrundssubtraktion — Detta filter används för att ta bort huvuddelen av den i bilden icke intressanta informationen. Först sparas en referensbild som inte innehåller några pekpunkter. Från varje bild som skall filtreras subtraheras referensbilden och kvar blir bara de delar av bilden som har ändrats sedan referensbilden togs. Det vill säga, i stort sett blir enbart pekpunkterna kvar [35].

Oskärpa — Filtret används för att reducera bruset i bilden.

Invertering — Detta filter används för att byta ut alla färger mot dess invers. Enkelt sett kan man säg att en pixel med byte-färgvärde `0` får värdet 255, 1 får 254 och så vidare. På så sätt blir vitt svart och svart blir vitt och kan användas då pekpunkter exempelvis uppträder som skuggor istället för ljuspunkter.

Kontrast — Kontast är ett filter som används för att öka skillnaden mellan de ljusa och mörka områdena i bilden. Detta filter används för att göra svaga ljuspunkter i en mörk bild tydligare så att spåraren senare kan upptäcka dem.

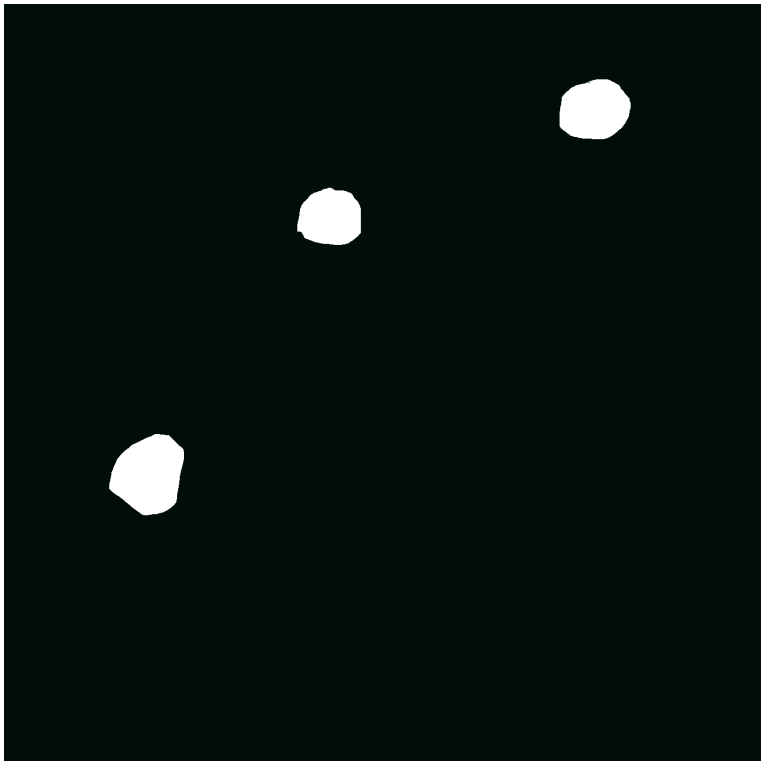
5.1.1.2 Bildsegmentering

Med bildsegmentering skapas en bild där varje bildpunkt representeras av en bit. Det här gör att varje bildpunkt antingen kan anta värdet färgad eller ofärgad (svart eller vit) [37]. Den här tekniken används för att skapa en bild där ljusa områden

blir vita och resten blir svart. Ur en sådan här bild är det enklare att finna sammanhängande områden än i en bild med fler färgbitar per bildpunkt.

5.1.1.3 Tolkning och analys

Efter lyckad filtrering och segmentering har spårare en bild med noll eller fler sammanhängande pekområden (se figur 5.2).



Figur 5.2: Bildsegmentering med tre sammanhängande områden

För varje sammanhängande pekområde måste spåraren avgöra om området är tillräckligt omfattande för att klassas som en pekpunkt. Dessutom måste spåraren avgöra om det är en tidigare punkt som flyttats eller en ny punkt.

5.2 Kommunikation – spårare och applikation

Den här prototypen använder sig av protokollet TUIO för att hämta in information om pekpunkter. För mer information varför TUIO används se avsnitt 3.3.1.1.

5.2.1 Om TUIO

TUIO [23] utvecklades som en del av reactTable* [24] (se avsnitt 2.1.2) och bygger på OSC-protokollet. Detta gör det möjligt att använda TUIO på alla system som har stöd för OSC, exempelvis Flash [23].

Innan ytterligare information om TUIO ges kommer en kort presentation om OSC-protokollet och hur det är uppbyggt.

5.2.1.1 OSC

OSC [45], OpenSound Control, är ett protokoll för kommunikation mellan datorer och multimediaenheter som exempelvis synthar. OSC är optimerat för nätverkskommunikation och är konstruerat för klient/server-kommunikation. OSC är dock oberoende av vilket transportprotokoll som används.

OSCs datablock kallas för paket och innehåller ett eller flera meddelanden. Ett meddelande består av en adress, en typsträng samt noll eller fler argument. De datatyper som stöds enligt OSC är ASCII-strängar, 32-bitars flyttal, 32-bitars heltal (integer) samt 64-bitars tidstämplar och binär data [45] [44].

Adressen i meddelandet ger destinationen [44], men de är inte förbestämda i protokollet vilket gör att man kan skapa nya sorters adresser. Adressen är uppbyggd av noder som är separerade med ett “/”, exempelvis skulle en svensk postadress kunna se ut enligt “/sverige/karlstad/universitetsgatan/2”. Sista noden i adressen är mottagaren av meddelandet.

Typsträngen är en ASCII-sträng som inleds med ett ",". Varje tecken i sekvensen motsvarar exakt datatypen för sekvensen av argument i meddelandet. I tabell 5.1 återfinns en lista över alla datatyper som stöds hos argumenten.

OSC Typ	*Motsvarande typ*
i	Heltal, 32 bitar
f	Flyttal, 32 bitar
s	OSC-sträng
b	OSC-blobdata

Tabell 5.1: OSC Argumenttyper

Antalet argument i meddelandet ges av antalet tecken i typsträngen och som tidigare nämnt ges även datatypen för argumentet av typsträngen.

5.2.1.2 TUIO över OSC

TUIO definierar en uppsättning meddelanden i OSC (se figur 5.3) med uppgift att förmedla information om objekt och pekpunkter. Meddelandena är definierade på så sätt att första argumentet i meddelandet anger vilken typ meddelandet har.

I huvudsak finns två meddelandetyper. En meddelandetyp, *alive*, talar om vilka objekt och pekpunkter som befinner sig på ytan. En annan meddelandetyp, *set*, ger information om objekt och pekpunkter och där anges exempelvis information om objektens position finns. Utöver dessa definierar TUIO ett meddelande som anger sekvensnummer, *fseq*, samt ett meddelande med namnet på servern, *source*. [23]

```
/tuio/[profileName] source name
/tuio/[profileName] set [sessionID parameterList]
/tuio/[profileName] alive [list of active sessionIDs]
/tuio/[profileName] fseq [int32]
```

Figur 5.3: TUIO, Meddelandetyper

TUIO-meddelanden är uppdelade i profiler (se figur 5.4). Profilnamnet finns lagrad i OSC-adressen och är där ändnoden. Profilens namn föregås av "/tuio/" i adressen [23].

2D Interactive Surface

```
/tuio/2Dobj set s i x y a X Y A m r
/tuio/2Dcur set s x y X Y m
```

2.5D Interactive Surface

```
/tuio/25Dobj set s i x y z a X Y A m r
/tuio/25Dcur set s x y z X Y Z m
```

3D Interactive Surfaces

```
/tuio/3Dobj set s i x y z a X Y Z A B C m r
/tuio/3Dcur set s x y z X Y Z m
```

raw profile

```
/tuio/raw_[profileName]
/tuio/raw_dtouch set i x y a
```

custom profile

```
/tuio/_[formatString]
/tuio/_sixyP set s i x y 0.5
```

Figur 5.4: TUIO, Profiler

Varje meddelandetyp har ett antal argument som varierar beorende på vilken profil som används. Argumenten som används för de olika meddelandena syns i figur 5.4. Semantiken för dessa argument syns i tabell 5.2 där även datatyper för de olika argumenten finns [23]. Observera att dessa argumenttyper inte skall blandas ihop med OSC-datatyper!

5.3 Applikation

Den applikation som har implementerats för att demonstrera några av de möjligheter som finns med flerpunktspekskärm har kallats *Lime - A Slight Touch of Lime*.

Argumenttyp	*Semantisk betydelse*	*OSC-datatyp*
s	Sessionsidentitet, temporärt ID	i
i	Klassidentitet	i
x, y, z	Position, mellan 0..1,	f
a, b, c	Vinkel, mellan 0..2PI	f
X, Y ,Z	Förflyttningsvektor	f
A, B, C	Rotationsvektor	f
m	Acceleration	f
r	Roteringsacceleration	f
P	Fri parameter	i,f,s,b

Tabell 5.2: TUIO, Meddelandeargument

I avsnitt 5.3.1 beskrivs den utvecklingsmiljö som används och i avsnitt 5.3.2 förklaras detaljer kring implementationen.

5.3.1 Uvecklingsmiljö

Applikationen har skrivits i Objective-C® för Apple MacOS X 10.5. Objective-C är utökning av C som ger objektorientering och vissa dynamiska egenskaper.

Applikationen använder sig av fyra huvudbibliotek. Dessa tillhandahåller funktionalitet för grafik, kommunikation och datahantering.

Cocoa — Cocoa [15] är en objektorienterad miljö som används för programutveckling i Apple MacOS X. Cocoa består främst av två ramverk; Foundation Kit och Application Kit. Dessa ramverk tillhandahåller bland annat strängar, värdemanipulering, containers samt grafiska element.

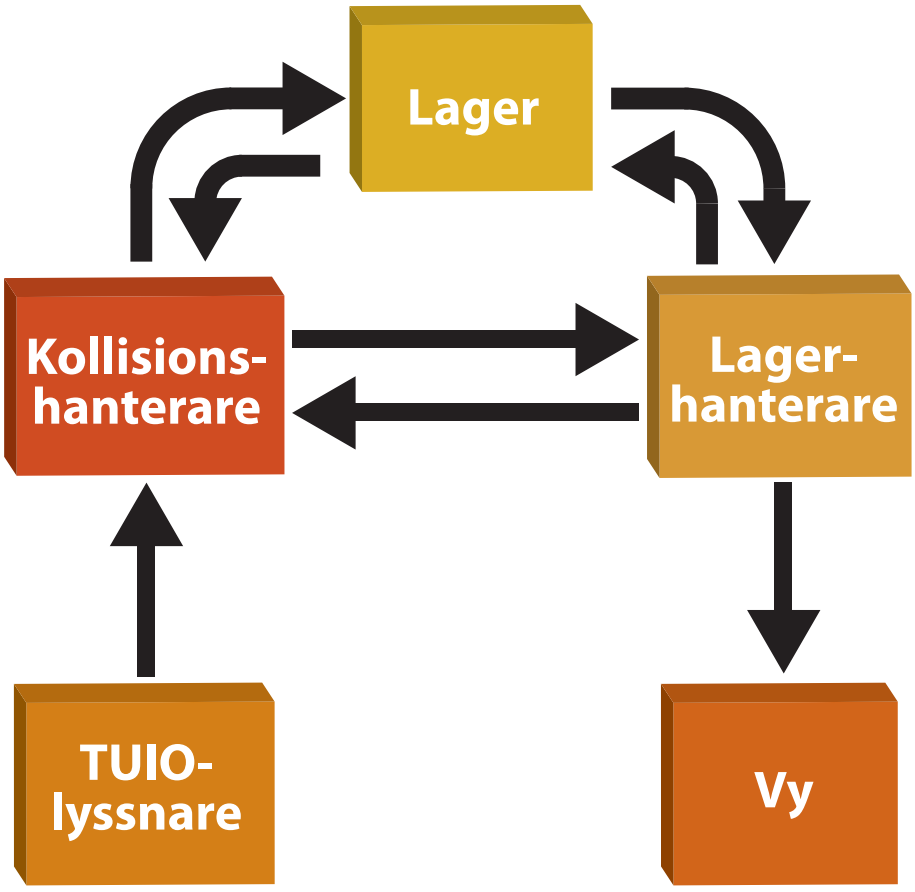
Quartz 2D — Quartz 2D [16] är ett två-dimensionellt API för att rita grafik i Apple MacOS X. Quartz 2D tillhandahåller Portable Document Format (PDF)s ritmodell.

Core Animation — Core Animation [17] är ett API för att hantera och animera grafiklager vilka kan innehålla olika sorters information, ex. bilder.

BBOSC — BBOSC [40] används för att ta emot data från spåraren, ytterligare information finns i avsnitt 5.3.2.1.

5.3.2 Implementation

Mjukvaran består av fem delar (se figur 5.5) och bygger på en teknik med lager. Varje lager representerar en viss mängd information och lagren är helt separerade från varandra. Dessa delars funktionalitet beskrivs i tur och ordning nedan.



Figur 5.5: Mjukvaruprototypens delar

TUIO-lyssnare — Detta är den komponent som lyssnar efter meddelanden från spåraren.

Lager — Lager är en sammansättning av operationer och innehåll. Innehållet kan till exempel vara grafik som ritas till vyn.

Lagerhanterare — Lagerhanterarens uppgift är att ha hand om alla de lager som finns i systemet och kontrollera ordningen de skrivs ut i.

Kollisionshanterare — Kollisionshanterare har till uppgift att upptäcka kollisioner mellan pekpunkter och lager.

Vy — Detta är den komponent som visas på skärmen.

Nedan beskrivs dessa komponenter i tur och ordning.

5.3.2.1 TUIO-lyssnare

Applikationen använder sig enbart av en profil (se avsnitt 5.2.1.2) från TUIO. Profilen som används är "/tuio/2Dcur" och denna representerar punkter i ett tvådimensionellt plan. I samma klass som "/tuio/2Dcur" finns även "/tuio/2Dobj" vilken kan användas för att unikt identifiera olika sorters objekt, men eftersom unikt identifierbara objekt inte är intressant i denna implementation så kommer denna profil att utelämnas.

I mjukvaruprototypen har en klass, vid namn `TUIOListener`, implementerats. Klassens uppgift är att ta emot TUIO-meddelanden från nätverket och översätta dessa till de meddelanden som återfinns i programkod 5.1. Dessa meddelanden skickas sedan vidare till kollisionshanteraren (se avsnitt 5.3.2.4). För mjukvaruprototypen har biblioteket BBOSC används för att ta emot och behandla OSC-meddelanden över nätverk. Biblioteket tar emot alla slags OSC-meddeleanden och en filtrering måste göras för att se till att endast relevanta meddelanden behandlas.

Implementation, TUIOListener

En lyssnare upprättas med hjälp av BBOSC och denna lyssnar på en specifik nätverksport. När data tas emot behandlas denna för att först och främst avgöra om det är ett TUIO-meddelande och dessutom ett TUIO-meddelande med rätt profil ("/tuio/2Dcur"). Efter detta behandlas pekpunktsinformationen och skickas sedan vidare till kollisionshanteraren (se avsnitt 5.3.2.4) via protokollet i programkod 5.1.

```
@interface NSObject (TUIOTouching)

   -(void)touch:(NSInteger)touchID
             at:(NSPoint)touchPoint;

   -(void)touch:(NSInteger)touchID
         moveTo:(NSPoint)touchPoint;

   -(void)unTouch:(NSInteger)touchID;

@end
```

Programkod 5.1: Meddelanden från TUIO-lyssnaren

Data tas emot i form av paket vilka kan innehålla flera TUIO-meddelanden. BBOSC kallar dessa paket för `BBOSCBundle` och meddelanden för `BBOSCMessage`. Programkod 5.2 nedan visar hur `BBOSCMessage` extraheras ur ett `BBOSCBundle` och sedan kontrolleras för att innehålla rätt adress.

När alla meddelanden i ett `BBOSCBundle` har samlats in kommer var och ett av dem att utföras. De meddelanen som är intressantast att utföra är "set" och "alive" eftersom de innehåller verklig pekpunktsinformation.

Meddelanden av typ "set" kontrolleras mot en intern lista över aktiva pekpunkter. Om det inkomna meddelandets pekpunktsidentitet inte tidigare existerar i listan skickas `touch:at:` (ett *Aktivera pekpunkt*-meddelande) till kollisionshanteraren och i annat fall skickas `touch:moveTo:` (ett *Flytta pekpunkt*-meddelande) till densamma.

```
NSArray* messages = [aBBOSCBundle attachedObjects];

for(BBOSCMessage* aMessage in messages)
{
   //Accept only 2D "cursor" information
   if([[[aMessage address] address]
                           isEqualToString:@"/tuio/2Dcur"])
   {
      //Handle message, check for and
      // handle set, alive, fseq and source
   }
}
```

Programkod 5.2: Hantering av meddelanden

När det gäller meddelanden av typen “alive” jämförs de aktiva pekpunkterna i meddelandet mot de som finns i den interna listan. Om punkter finns med i den interna listan men inte i “alive”-meddelandet kommer `unTouch:` (ett *Avaktivera pekpunkt*-meddelande) skickas till kollisionshanteraren för varje sådan punkt.

5.3.2.2 Lager

Ett lager är en entitet med vilken det är möjligt att interagera via pekpunkter. Lagret kan även innehålla grafisk information som visas i gränssnittet. Lagret har en position, en storlek samt en ordning i djupled.

Varje lager har en så kallad *ankarpunkt*[1] vilken är en koordinat i lagrets lokala koordinatsystem (se figur 5.6, där ankarpunkten är märkt A) kring vilken operationer verkar. Lagret har en koordinat i det globala koordinatsystemet ((30,20) i figur 5.6) i vilken lagrets ankarpunkt placeras.

Ordningen i djupled talar om vilket lager, om två eller fler lager befinner sig på samma position, som skall visas överst och vilket som skall döljas (se figur 5.7). Lägre ordningstal i djupled gör att lagret visas framför ett lager av högre ordningstal.

[1]Engelska: Anchor point

```
//touchDelegate is a deltegate to the collisionhandler
- (void)processSetMessage:(BBOSCMessage*)message
{
   NSArray* arguments = [message attachedObjects];
   int touchID;
   TUIOPoint touchPoint;

   touchID = [[arguments objectAtIndex:1] intValue];
   touchPoint.x = [[arguments objectAtIndex:2] floatValue];
   touchPoint.y = [[arguments objectAtIndex:3] floatValue];

   if([activeTouchIds containsObject: [NSNumber numberWithInt:
       touchID]])
   {
      //Send "touch-point" move
      if([touchDelegate respondsToSelector:
                        @selector(tuioTouch:moveTo:)])
         [touchDelegate tuioTouch:touchID moveTo:touchPoint];
   }
   else
   {
      //Send "touch-point" down
      if([touchDelegate respondsToSelector:
                        @selector(tuioTouch:at:)])
         [touchDelegate tuioTouch:touchID at:touchPoint];
   }
}
```

Programkod 5.3: Hantering av ”set”-meddelande

Tre stycken elementära operationer har skapats - rotation, förflyttning och skalning vilka beskrivs i avsnitt 5.3.2.6.

Lagerinnehåll

Den grafiska representationen av lagret beskrivs med hjälp av ett så kallat `CALayer` [18] vilket är en datamedlem i lagret. Detta `CALayer` kan sedan placeras i vyn (se avsnitt 5.3.2.5) och på så sätt visas på skärmen. `CALayer` är en klass som tillhandahålls av Core Animation och kan innehålla olika sorters media så som texter, bilder och film.

```
//touchDelegate is a deltegate to the collisionhandler
- (void)processAliveMessage:(BBOSCMessage*)message
{
   NSArray* arguments = [message attachedObjects];
   NSMutableSet* tempTouchIds = [[NSMutableSet alloc] init];

   for(BBOSCArgument* anArgument in arguments)
   {
      [tempTouchIds addObject:[NSNumber numberWithInt:
                                       [anArgument intValue]]];
   }

   for(NSNumber* aTouchID in activeTouchIds)
   {

      if(![tempTouchIds containsObject:aTouchID])
      {
         //Send up
         if([touchDelegate respondsToSelector:
                           @selector(tuioUnTouch:)])
            [touchDelegate tuioUnTouch: [aTouchID integerValue]];
      }

   }

   [activeTouchIds release];
   activeTouchIds = tempTouchIds;
}
```

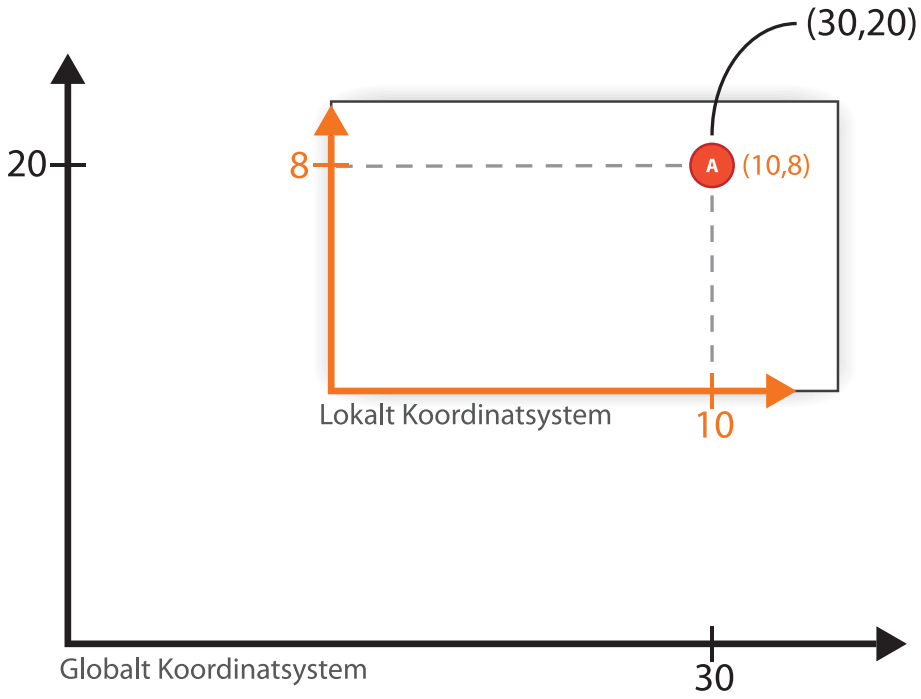
Programkod 5.4: Hantering av ”alive”-meddelande

5.3.2.3 Lagerhanterare

Lagerhanteraren har en central roll i applikationen eftersom den har ansvar för samordningen av samtliga lager. Lagerhanteraren bestämmer lagrens ordning i djupled, tillhandahåller dem till kollisionshanteraren samt lägger dem i vyn.

Lagerhanteraren har en intern lista i form av en så kallad `NSMutableArray` [19], en redigerbar vektor tillhandahållen av Cocoa, som håller samtliga lager i den ordning de presenteras. Lager med djupled `0` ligger alltså överst i listan. Denna lista tillhandahåller lagerhanteraren även till kollisionshanteraren som behöver information om vilka lager som finns i systemet (se avsnitt 5.3.2.4).

Utöver detta är det lagerhanterarens uppgift att placera lagrens grafiska representation i vyn (se figur 5.8). Lagerhanteraren tar lagrens `CALayer` och placerar



Figur 5.6: Koordinatsystem och ankarpunkt för lager

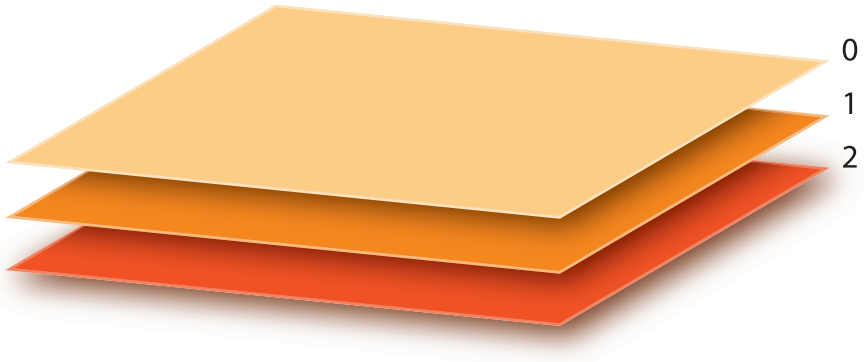
dessa i vyns `CALayer` (se avsnitt 5.3.2.5).

5.3.2.4 Kollisionshanterare

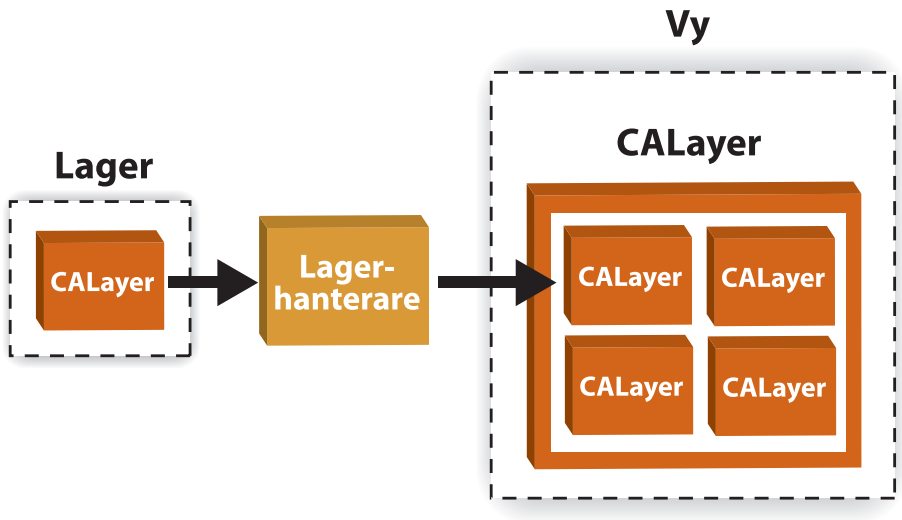
Kollisionshanterarens uppgift är att finna de lager som är intresserade av de punkter som registreras och förmedlats till kollisionshanteraren från `TUIOListener` via protokollet i programkod 5.1 samt dessutom vidarebefordra dessa pekpunkter till de intresserade lagren.

För att finna vilka lager som är intresserade av vilka pekpunkter hämtar kollisionshanteraren en lista över alla lager i systemet från lagerhanteraren (se avsnitt 5.3.2.3). Därefter ställs ett antal fördefinierade frågor till lagren med uppgift att avgöra ifall lagren är intresserade av pekpunkterna eller inte.

För att ett lager skall fungera i systemet måste det svara på dessa frågor och dessutom ha möjlighet att ta emot tre typer av meddelanden innehållandes pekin-



Figur 5.7: Djupledsordning för lager



Figur 5.8: Lager läggs i vyn via lagerhanteraren

formation. Frågorna och meddelandena är definierade via protokollet i programkod 5.5.

Frågan `containsPoint` syftar till att ge kollisionshanteraren information om ifall ett lager innehåller den specifika punkten eller inte. Trots att lagret alltid är rektangulärt ger denna funktionalitet lagret möjlighet till att rita upp valfri form och enbart tala om vilka punkter som ligger i formen.

Frågan `interestedInPoint` ska ge svar på ifall ett lager är intresserad av en specifik pekpunkt då lagret ligger på ett visst ordningsnummer i djupled. Detta ger

```
@interface NSObject (Touching)

   -(void)touch:(NSInteger)touchID
             at:(NSPoint)touchPoint;

   -(void)touch:(NSInteger)touchID
         moveTo:(NSPoint)touchPoint;

   -(void)unTouch:(NSInteger)touchID;

   -(bool)interestedInPoint:(NSPoint)point
              withAZOrderOf:(NSInteger)order;

   -(bool)containsPoint:(NSPoint)point;

   -(bool)transparentToTouch;

@end
```

Programkod 5.5: Protokollet Touching

lagret möjligheter att själv bestämma vilka pekpunkter som är intressanta. För att illustrera detta kan två lager läggas ovanpå varandra. Det översta lagret kommer få frågan `interestedInPoint:aPoint withAZOrderOf:0;` som denna kan svara på. Lagret under kommer däremot att få frågan `interestedInPoint:aPoint withAZOrderOf:1;`, eftersom denna ligger dolt bakom det första lager. Det viktiga här är att lagret fortfarande har möjlighet att vara intresserad av punkter trots att det inte är överst.

Den sista frågan, `transparentToTouch`, är till för att indikera att lagret är genomskinlig för pekningar. Om lagret svarar `YES` på denna fråga kan det fortfarande acceptera en pekpunkt men när pekpunkten skickas till nästa lager har inte ordningstalet för djupled räknats upp. Ett exempel för att illustrera detta kan vara två lager som ligger ovanpå varandra. Det översta lagret har satt sin `transparentToTouch` till `YES` och tar emot frågan `interestedInPoint:aPoint withAZOrderOf:0;`. Lagret under detta lager kommer också att få frågan `interestedInPoint:aPoint withAZOrderOf:0;` eftersom ordningstalet inte har räknats upp.

Pekpunktsdata

Efter att kollisionshanteraren har gjort klart för sig, via ovanstående frågor, vilka lager som är intresserade av pekpunkten sparas denna information lokalt i kollisionshanteraren för att användas senare. Pekpunkten har, så länge den existerar, en unik identitet och denna identitet lagras tillsammans med vilka lager som är intresserade av just denna pekpunkt. På så sätt behöver inte frågan om pekpunkten är intressant för lagret ställas mer än en gång.

Det finns tre olika meddelanden (se programkod 5.5), hämtade från avsnitt 3.3.2.2, vilka representerar tre olika händelser i en pekpunkts livstid.

När en pekpunkt har inkommit till kollisionshanteraren skickas `touch:at:` till de lager som tidigare talat om för kollisionshanteraren att de är intresserade av denna pekpunkt. Den information som skickas till lagret är pekpunktens identitet och dess placering i planet.

Meddelandet `touch:moveTo:` skickas till intresserade lager för att tala om att en pekpunkt med ett viss identitet har flyttat sig.

Det sista meddelandet som skickas under en pekpunkts livstid är `unTouch:` vilket indikerar för ett lager att pekpunkten inte längre existerar.

5.3.2.5 Vyn

Vyn är implementerad som en `NSView`[20], vilket är vyklass tillhandahållen av Cocoa. Denna har ett `CALayer` i vilken lagerhanteraren placerar sina registrerade lagers grafiska representation, det vill säga deras `CALayer`. Vyn är i sin tur placerad i ett fönster på skärmen och detta medför att de lager som lagerhanteraren placerar i vyns `CALayer` visas på skärmen (se figur 5.8).

5.3.2.6 Standardlager

Applikationen definierar ett lager som innehåller den grundläggande funktionaliteten som ett lager kan tänkas behöva, detta lager kallas `LimeBasicContentLayer`. Denna lagertyp kan ärvas och ger då funktionalitet som besvarar frågorna om vilka pekpunkter som är intressanta (se programkod 5.5) med standardsvar, vilket i praktiken innebär att lagret är intresserad av alla pekpunkter inom dess område när lagret är överst i djupled.

Vad det gäller de meddelanden som levererar information om pekpunkter (se programkod 5.5) håller `LimeBasicContentLayer` ordning på vilka aktiva pekpunkter som är placerade på lagret samt i vilken ordning dessa uppkom. Utöver detta finns även funktionalitet för att hantera grafiskt innehåll (se avsnitt 5.3.2.2).

Detta lager har grundstommen för att visa grafiskt innehåll, det vill säga att den har ett `CALayer` vilket går att tilldela innehåll. Lagret har även ett datamedlemsobjekt som har till uppgift att generera innehållet i lagrets `CALayer`, detta datamedlemsobjekt kan tilldelas när lagret initieras.

Applikationen definierar även ett lager med några grundläggande operationer (se nedan) och dessa finns implementerade i `LimeBasicManipulatableLayer`. Denna lagertyp ärver från `LimeBasicContentLayer` vilket innebär att även detta lager kan ha innehåll på samma sätt som beskrivits ovan. Det är möjligt att ärva från `LimeBasicManipulatableLayer` för att vidareutveckla funktionaliteten, exempelvis genom införande av nya operationer.

Flyttningsoperation - flytta lager mellan två punkter

Flyttningsoperationen är en relativt simpel operation som är till för att kunna flytta ett lager från en position till en annan. Operationen kräver enbart en pekpunkt för att fungera och förflyttningen sker genom att beräkna skillnaden i x- och y-led när pekpunkten flyttats. Därefter adderas skillnaden till positionen på

`CALayer`. Detta medför att `CALayer` får en ny position och representerar där med förflyttningen.

Flyttningsoperationen finns i `LimeBasicManipulatableLayer` implementerad som `-(void)moveFromPoint:(NSPoint)from toPoint:(NSPoint)to`.

Rotationsoperation - rotera kring ankarpunkt

Rotationsoperationen är en relativt avancerad operation. Rotationsoperationen syftar till att kunna rotera ett lager kring en valbar ankarpunkt i lagret (se figur 5.9) och finns i `LimeBasicManipulatableLayer` implementerad som `-(void)rotateFromPoint:(NSPoint)from toPoint:(NSPoint)to roundPoint:(NSPoint)anchor`.

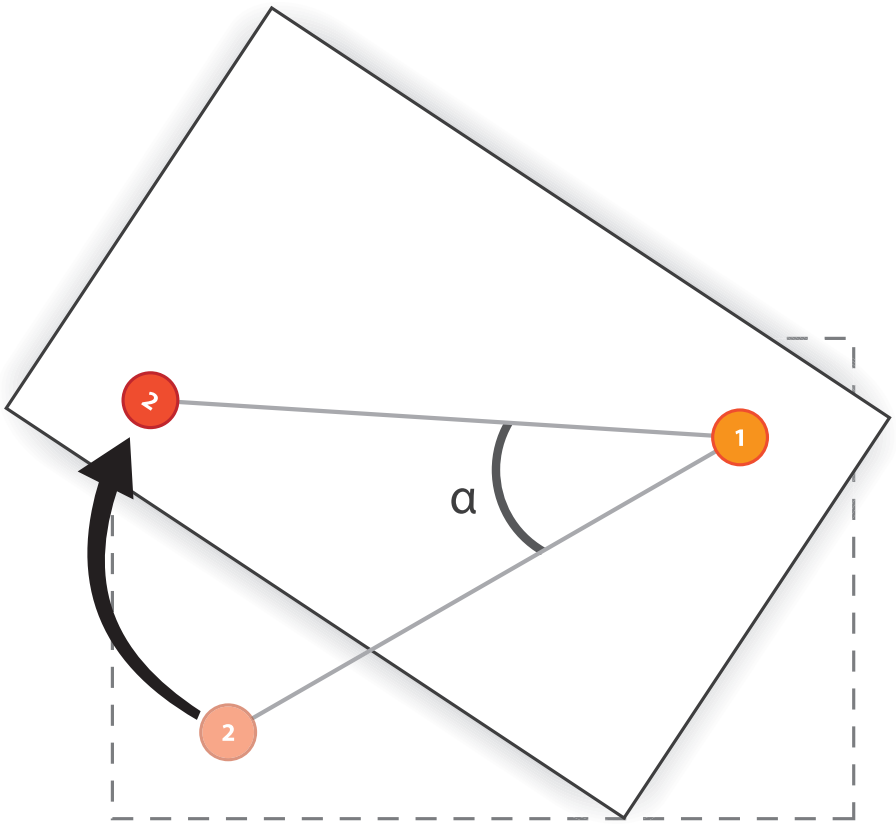
För att utföra en rotation krävs två pekpunkter, där en av pekpunkterna agerar ankarpunkt och den andra pekpunkten ger rotationsvinkeln vid förflyttning. Den pekpunkt som inte flyttas benämns nedan med ankarpunkt och den andra pekpunkten, som kommer att utföra en förflyttning, kallas nedan rotationspunkt. För att en rotation skall uppstå måste denna punkt göra en förflyttning.

När rotationspunkten har gjort en förflyttning finns två koordinater tillgängliga för den. En koordinat vilken punkten flyttats från och en dit punkten flyttats. Först skapas ett nytt koordinatsystem för `CALayer` med origo i ankarpunkten (se figur 5.10).

Vinkeln mellan start och stopp på rotationspunkten, här kallat α, beräknas med formel 5.1 [1] och kan ses implementerad i programkod 5.6.

$$\vec{A} \cdot \vec{B} = |\vec{A}||\vec{B}| \cos(\alpha) \tag{5.1}$$

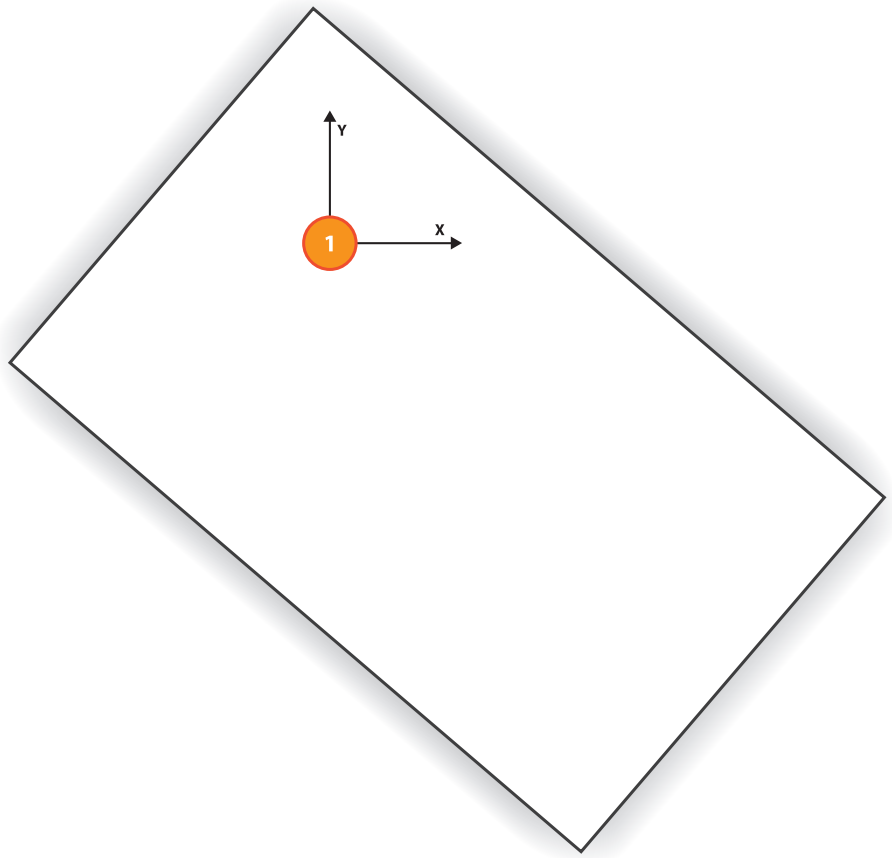
När vinkeln har besräknats är det viktigt att avgöra vilket håll rotationen skall ske åt. Formeln 5.1 ger endast den minsta vinkeln mellan två punkter och därmed ingen riktning. För att bestämma riktningen roteras alla punkter i koordinatsystemet på ett sådant sätt att start för rotationspunkten hamnar på y-axeln.



Figur 5.9: Rotation kring ankarpunkt

Därefter kontrolleras tecknet på x-värdet för stopp-rotationspunkten och i vilken kvadrant som start-rotationspunkten ligger i. Två fall uppstår, ett vid kvadrant I och II samt ett i kvadrant III och IV. I kvadrant I och II ger ett positivt värde på stopp-rotationspunktens x-värde en rotation åt höger och ett negativt värde en rotation åt vänster. I kvadrant III och IV är detta det motsatta, alltså ett positivt x-värde ger vänsterrotation och ett negativt x-värde ger högerrotation. Detta kan ses implementerat i programkod 5.7.

Rotationen appliceras sedan direkt på `CALayer`.



Figur 5.10: Nytt koordinatsystem vid ankarpunkten

Skalningsoperation - skala lagret

Skalning av lager innebär att lagrets storlek ändras proportionellt och för detta behövs två pekpunkter. En av pekpunkterna kommer här att användas som referenspunkt och den andra bestämmer skalfaktorn. Skalningsoperationen finns i `LimeBasicManipulatableLayer` implementerad som `-(void)scaleFromPoint:(NSPoint) from toPoint:(NSPoint)to withReferencePoint:(NSPoint)reference`.

När pekpunkten som bestämmer skalfaktorn flyttas så registreras koordinaten som punkten flyttas från samt koordinaten som punkten flyttas till. Detta ger två sträckor (A och B) och den procentuella skillnaden mellan dessa två sträckor ger skalningen av lagret (se figur 5.11).

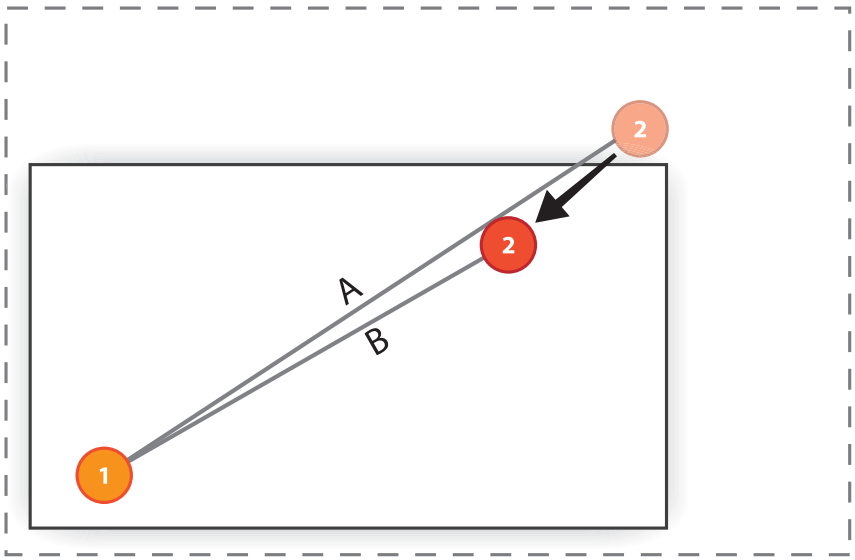
```
//The angle between to vectors
// A dot B = |A||B|cos(a)

CGFloat angle = 0.0f;
CGFloat aX = fromTranslated.x;
CGFloat aY = fromTranslated.y;
CGFloat bX = toTranslated.x;
CGFloat bY = toTranslated.y;

CGFloat aDotB = aX * bX + aY * bY;
CGFloat absA = sqrtf(powf(aX, 2) + powf(aY, 2));
CGFloat absB = sqrtf(powf(bX, 2) + powf(bY, 2));

angle = acosf(aDotB / (absA * absB));
```

Programkod 5.6: Beräkna rotationsvinkeln



Figur 5.11: Skalning av lager

Det kan vara så att `CALayer` har skalats tidigare och detta innebär att det redan finns en skalfaktor applicerad. Skalfaktorn måste därför beräknas utifrån att det redan kan finnas en skalning, vilket syns i programkod 5.8. Efter detta kan den nya skalfaktorn (`newScaleFactor`) appliceras direkt på `CALayer`.

```
//Calculate direction for rotation.
CGFloat direction = 1.0f;

//Rotate points in such a way that from is positioned on the y-
    axis.
CGFloat angleFromXAxisToFrom = angleBetweenPoints(CGPointMake(1.0
    f,0.0f), fromTranslated);
CGFloat angleToRotate = M_PI_2 - angleFromXAxisToFrom;

//If we are in one of the lower quadrants the direction is
    inverted.
CGFloat invertDirection = 1.0f;
if(fromTranslated.y < 0)
   invertDirection = -1.0f;
angleToRotate *= invertDirection;

CGAffineTransform rotatePoints = CGAffineTransformMakeRotation(
    angleToRotate);
CGPoint toRotated = CGPointApplyAffineTransform(toTranslated,
    rotatePoints);

//The direction is indicated by the x value of the rotated to
    point.
direction *= -1.0f * invertDirection * toRotated.x / fabs(
    toRotated.x);

//Apply the direction to the angle.
angle *= direction;
```

Programkod 5.7: Applicera en riktning på vinkeln

Egendefinierade operationer

Genom att introducera en ny lagertyp i systemet finns det möjlighet att definiera nya operationer utan att behöva ändra i omgivande systemstruktur.

Genom att skapa ett nytt lager som implementerar det tidigare nämnda protokollet (se programkod 5.5) kan valfritt beteende skapas. Det är också möjligt att ärva från `LimeBasicContentLayer` (eller `LimeBasicManipulatableLayer`) för att erhålla den grundläggande funktionaliteten. Det är i så fall möjligt att åsidosätta[2] den funktionalitet som implementeras av `touch: at:`, `touch moveTo:` och `unTouch:` och därefter använda sig av metoderna för flytt, skalning och rotation.

[2]Engelska: override

```
CGFloat originalDistance =
   distanceBetweenPoints(NSPointToCGPoint(reference),
       NSPointToCGPoint(from));

CGFloat newDistance =
   distanceBetweenPoints(NSPointToCGPoint(reference),
       NSPointToCGPoint(to));

CGFloat currentScaleFactor = [[[self primaryLayer]
                              valueForKeyPath:@"transform.scale"]
                                  floatValue];

CGFloat calculatedScale = currentScaleFactor *
                          (newDistance / originalDistance);

CGFloat newScaleFactor = calculatedScale;
```

Programkod 5.8: Beräkning av ny skalfaktor

5.3.2.7 Problem

Ett antal problem uppstod under programmeringen av logik för olika lager. Nedan beskrivs problematiken samt lösningen.

Insättning av ankarpunkt

Ett lagers position bestäms av två egenskaper, det ena är koordinaten för lagret självt och det andra är var denna koordinat skall appliceras på lagret (exempelvis i mitten). `CALayer` är implementerat så att koordinaten för lagret appliceras i ankarpunkten för densamma.

När en godtycklig punkt sätts som en ankarpunkt på ett `CALayer` uppstår en förflyttning av lagret. Detta är ett logiskt beteende eftersom ankarpunkten har flyttats. För att lagret inte visuellt skall flyttas måste positionen ändras för att kompensera. Positionen måste ändras lika mycket som ankarpunkten har flyttats, detta gör att lagret visuellt ligger kvar.

Roteringsalgoritm

Någon algoritm för att bestämma hur mycket ett lager skall roteras utifrån ett antal punkter var inte tillgänglig. I stället fick en algoritm utarbetas med en matematisk formel som grund. Utifrån formel 5.1 kunde en vinkel mellan två vektorer erhållas och problemet var bestämma ifall en höger- eller vänsterrotation skulle utföras.

Flera olika lösningsmetoder provades innan lyckat resultat uppnåddes i samtliga fall. Utifrån de olika lösningsmetoderna kunde gemensamma nämnare härledas vilket gav en funktionsduglig algoritm. Denna algoritm implementerades med framgång.

Skalningsalgoritmen

När skalningsalgoritmen implementerades uppstod stora problem på olika ställen som kunde härledas till insättning av ankarpunkten. Exempel på problem som uppstod var att rotation inte längre skedde kring ankarpunkten.

Anledningen till problemet med insättningen av ankarpunkten var att ingen hänsyn togs till skalfaktorn när lagrets position kompenserades för ankarpunktens förflyttning. Problemet avhjälptes genom att ta hänsyn till lagrets skalfaktor.

5.4 Sammanfattning

Mjukvaruprototypen använder sig av en spårare för att läsa av den bildström som kan erhållas från hårdvaruprototypen. Spåraren överlämnar information om pekpunkter via TUIO-protokollet till applikationen. I applikationen översätts meddelandena till ett protokoll.

Applikationen har tre inbyggda operationer; skalning, rotation och förflyttning. Dessa typer kan användas via ett lager vilket också kan ta emot ett valfritt

innehåll. Egendefinierade operationer kan införas i applikationen genom att implementera en ny lagertyp, på detta sätt blir applikationen utbyggbar.

Kapitel 6
Utvärdering av prototyp

I det här kapitlet kommer en utvärdering av både hårdvaru- och mjukvaruprototypen att ges. Utvärderingen syftar först och främst till att kontrollera prototyperna mot de krav som finns. Först utvärderas hårdvaran och därefter mjukvaran. Sist i detta kapitel sammanfattas utvärderingen.

6.1 Utvärdering av krav

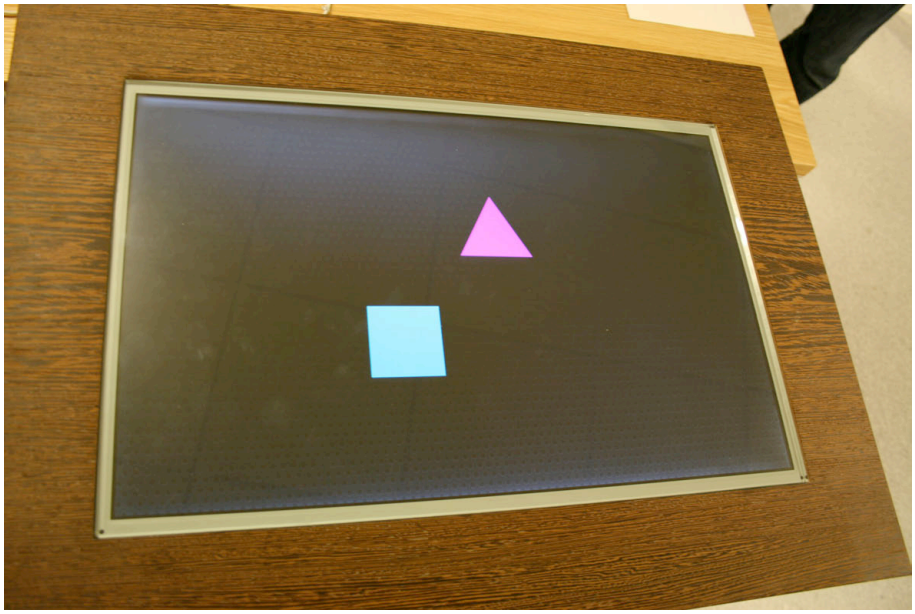
6.1.1 Hårdvara

Det fanns totalt tre krav ställda på hårdvaran (enligt avsnitt 3.1) och dessa var:

1. Visa gränssnitt från videokälla
2. Användas för att detektera pekpunkter
3. Max 4000 SEK i tillverkingskostnad (inkl. moms)

6.1.1.1 Grafiskt gränssnitt

Hårdvaran skall enligt första kravet visa det grafiska gränssnittet för applikationen. För att uppfylla detta krav använder sig hårdvaran av en plattskärm som är ansluten till en dator via en D-Sub-15-kontakt (se avsnitt 4.4.3). Kravet kontrolleras genom att se att plattskärmen kan visa det gränssnitt som programmerats. I figur 6.1 syns applikationens gränssnitt visat på hårdvaruprototypen.



Figur 6.1: En ögonblicksbild av gränssnittet

6.1.1.2 Detektering av pekpunkter

Det andra kravet bestod i att att pekpunkter skall gå att detektera. För att uppfylla detta krav levereras en serie stillbilder (se avsnitt 4.1.1) över pekytan till en spårare. För att skapa pekpunkter som är detekterbara av spåraren används pennor med inbyggda infraröda lysdioder (se avsnitt 4.2.3).

För att testa att hårdvaran uppfyller kravet om att det skall vara möjligt att detektera pekpunkter med hårdvaran så aktiveras kameran och spåraren. En testapplikation lyssnar på TUIO-meddelanden från spåraren. En pekpenna (se avsnitt 4.2.3) trycks ned mot ytan på hårdvaran och testapplikationen tar emot meddelanden från spåraren att en pekpunkt med en specifik koordinat är aktiv. När pennan lyfts erhåller testapplikationen ett meddelande om att pekpunkten är borta.

6.1.1.3 Kostnad

Det sista kravet för hårdvaruprototypen var att den inte får kosta mer än 4000 SEK (inkl. moms) kr att tillverka. Nedan finns (i tabell 6.1) en sammanställning på inköpskostnader för de komponenter som ingår i hårdvaruprototypen.

Komponent	*Pris (inkl. moms)*
LCD-skärm	1890, 00 SEK
Kamera	149, 00 SEK
Glas & spegel	170, 00 SEK
Trä till chassi	500, 00 SEK
IR-dioder	22, 40 SEK
Tangentbordsswitch	15, 00 SEK
Batterier	30, 00 SEK
Plexiglasstav	30, 00 SEK
USB-adapter	67, 80 SEK
Apparatintag	31, 90 SEK
Isolationshuv	7, 99 SEK
Total	2914, 09 SEK

Tabell 6.1: Prislista

Som synes i tabell 6.1 överstiger *inte* den totala konstruktionskostnaden det övre taket på 4000 SEK. Alla priser i denna tabell är inklusive moms.

6.1.2 Mjukvara

Enligt avsnitt 3.1 fanns fem krav på mjukvaran, dessa är som följer:

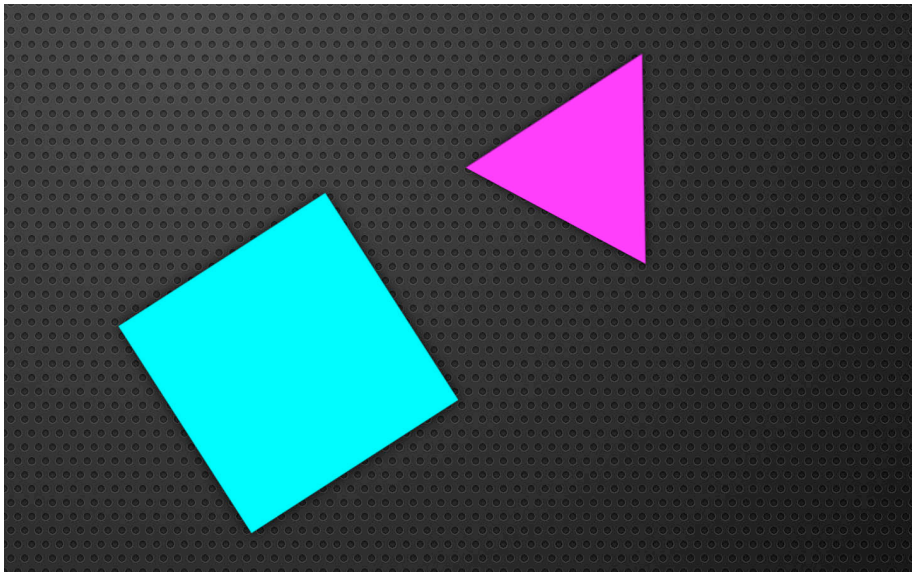
1. Minst två objekt skall kunna visas samtidigt
2. Rotationsoperation
3. Proportionell skalning
4. Förflyttning
5. Möjliggöra introduktion av nya lager

6.1.2.1 Minst två objekt

I figur 6.1 syns två separata objekt, en triangel och en rektangel. I bilaga C syns koden för hur två lager med figurer läggs in. Först skapas två innehåll, en rektangel samt en triangel. Därefter skapas två stycken `LimeBasicManipulatableLayer` (se avsnitt 5.3.2.6) och med dessa associeras innehållet. Sist registreras dessa hos lagerhanteraren som placerar ut innehållet i vyn.

6.1.2.2 Operationer

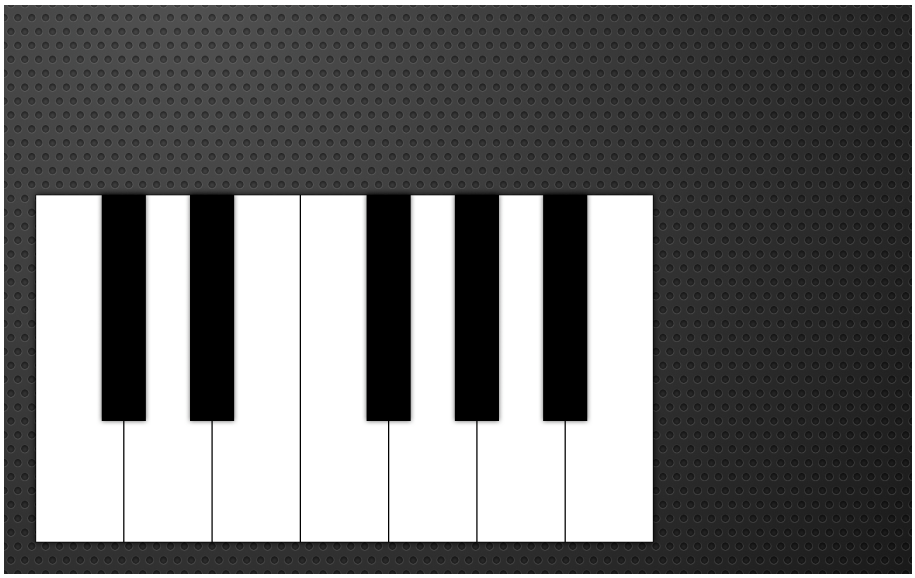
I krav nummer två till fyra på mjukvaran står det att det skall vara möjligt att rotera, skala och flytta kvadraten och triangeln ovan. I och med att `LimeBasicManipulatableLayer` (se avsnitt 5.3.2.6) används för att representera dessa figurer går detta att utföra. I figur 6.2 kan man se att både rotationen och skalningen fungerar som önskat.



Figur 6.2: Rotation och skalning

6.1.2.3 Introduktion av nya lager

Det andra kravet för mjukvaran beskriver att densamma skall vara implementerade på ett sådant sätt att det skall gå att införa nya typer av lager. För att visa detta har ett "piano"-lager implementerats som kan ta emot flera tangenttryckningar på en och samma gång.



Figur 6.3: Skärmavbild av pianot

Figur 6.3 visar en skärmavbild av användargränssnittet. En ny lagertyp har introducerats i *Lime* vilken har kallats `LimePiano`. `LimePiano` ärver från `LimeBasicContentLayer` för att inte behöva implementera hela protokollet `Touching` (se avsnitt 5.5). När lagret tar emot `touch:at:` eller `touch:moveTo:` kontrolleras vilken tangent som skall spelas och efter detta spelas tangentens associerade ljud upp.

Pianot spelar upp olika toner oberoende av varandra vid "anslag" av tangenter men endast en pekpunkt kan slå an en tangent vid ett och samma tillfälle. Däremot är det möjligt för godtyckligt antal pekpunkter att slå an olika tangenter samtidigt.

6.2 Iakttagelser och begränsningar

Nedan följer några brister som iakttagits under utvärderingen av systemet.

6.2.1 Fördröjningar i interaktionen

Interaktionen med applikationen via hårdvaruenheten kan upplevas ge långsam respons. Det uppstår en fördröjning på cirka $0,5$ sekunder från att användaren flyttar pekpunkten till att motsvarande operation har utförts. Orsaken till detta är inte fullt utredd, men tänkbara anledningar kan vara följande:

Kamera — Kameran har en bildhastighet på 30 bilder per sekund, detta borde vara fullt tillräckligt. Om kameran kopplas mot en applikation som visar kamerans bild i realtid kan en viss fördröjning mellan verkligheten och den visade bilden upplevas. Denna fördröjning skulle kunna bero på följande orsaker: tiden det tar från det att bilden tas tills den skickas ut på kabeln, drivrutinen som används för att kommunicera med kameran är långsam, mjukvara som används för att testa kameran är långsam.

Spårare — En stor mängd filtreringar på varje stillbild som kommer från kameran görs av spåraren. Dessa filtreringar är beräkningsintensiva och tar därför en viss mängd tid i anspråk.

Applikation — Även applikationen gör en del beräkningar för varje pekpunkt som spåraren tillhandahåller den och även dessa beräkningar tar tid i anspråk.

Nätverk — En flaskhals i sammanhanget skulle kunna vara nätverket. Vid testerna av systemet används dock *loopback*-gränssnittet och transportprotokollet *UDP*, med dessa förutsättningar bör inte nätverket vara en bidragande faktor till fördröjningarna.

Tester har utförts genom att en simulator genererar pekpunkter som sändes som TUIO-meddelanden över *loopback*-gränssnittet. De genererade pekpunkterna utför ett antal operationer i applikationen. Testerna är inte vetenskapligt utförda eftersom den exakta skillanden i svarstid då pekpunkter genereras av simulatorn och vid användandet av hårdvaran är inte relevant. Trots detta kunde man med ögat se en markant skillnad när hårdvaruenheten används i jämförelse med simulatorn, vilket tyder på att problemen med fördröjningen ligger i hårdvaruprototypen eller spåraren.

6.2.2 Grafikproblem

Den dator som hårdvaruprototypen var kopplad till hade stundvis problem med att rendera grafik. Problemen till detta tros ligga i hårdvaran och inte mjukvaran då samma problem inte uppstår på andra datorer. Då datorn som används för att visa gränssnittet är en bärbar Macintosh har ett grafikkortsutbyte uteslutits.

6.3 Sammanfattning

De krav som initalt var ställda på prototyperna har uppfyllts. Priset för hårdvaruprototypen var under de uppsatta 4000 SEK. Hårdvaruprototypen visar en bild av gränssnittet och tar emot pekpunkter som den vidarebefordrar till mjuvaruprototypen.

Mjukvaruprototypen är utvecklad så att det ska vara möjligt att introducera nya typer av innehåll men även nya operationer. För att visa att detta fungerar har ett “piano” implementerats vilket fungerar tillfredställande.

Kapitel 7
Slutsatser

7.1 Detta projekt

De prototyper som har konstruerats fungerar och uppfyller de krav som är ställda på dem. Det finns dock, enligt vår åsikt, vissa aspekter som kunde arbetas ytterligare med vilka beskrivs nedan.

7.1.1 Pekpunktsinmatning

Hårdvaruprototypen byggdes inte helt efter initiala planer då pekning med fingrarna misslyckades eftersom den valda konstruktionen inte gav tillräckligt framträdande pekpunkter. Istället introducerades pennor som används för pekning. Vår åsikt, kring pennorna, är att de inte är lika bra som fingrarna eftersom man förlorar känslan av att arbeta direkt med händerna.

Pennorna är dock en bra kompromiss då ingen ytterligare tid fanns för att implementera någon annan metod. Det är möjligt att modifiera prototypen för att använda sig av frustrerad total intern reflektion och på så sätt troligen möjliggöra att fingrar kan användas för att navigera på skärmen.

7.1.2 Optimering av mjukvara

Mjukvaran fungerar som det var planerat från början och det går att utöka den med nya lagertyper. Att skapa en ny lagertyp kräver tyvärr vissa kunskaper om hur systemet fungerar i övrigt, detta tycker vi är synd men det borde gå att göra modellen mer modulär än vad den är idag.

I övrigt är applikationen i vissa hänseenden långsam och kan behöva optimeras. Det finns säkerligen en del programkod i applikationen som kan reduceras eller bytas ut mot kod som är mindre processorkrävande.

7.2 Framtida vidareutveckling

Det finns, enligt oss, stöd för att använda applikationen som grund vid utveckling av nya applikationer vilket också är en av grundtankarna.

7.2.1 Användargränssnittsanpassning

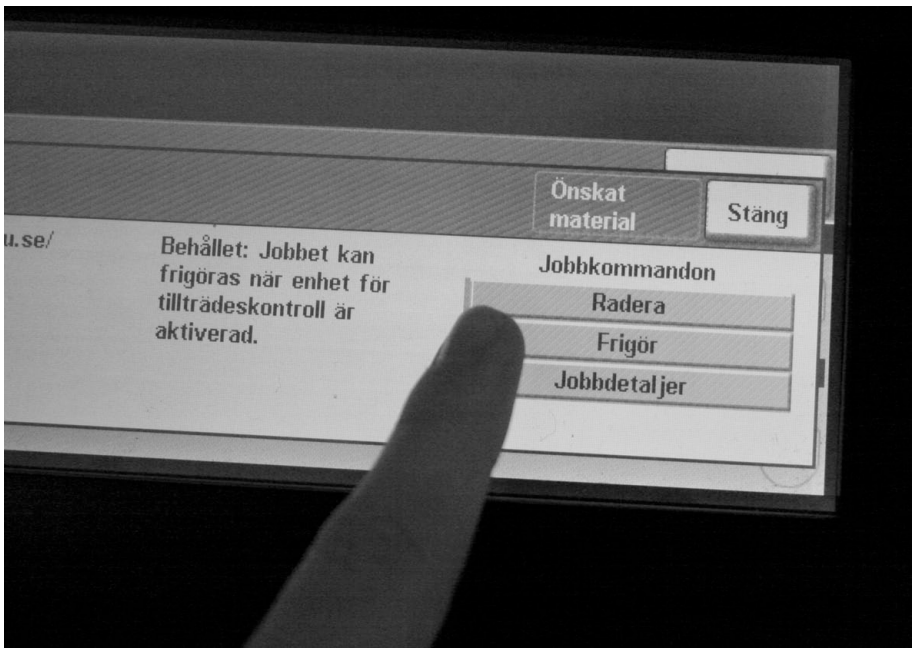
Applikationen som utvecklats under detta examensarbete kan anpassas för att fylla olika sorters behov när det gäller mjukvara till flerpunktspekskärmar. Det är därför möjligt att vidareutveckla mjukvaran till att passa de behov som för tillfället finns. När ett nytt användargränssnitt utvecklas till applikationen finns några viktiga aspekter att ta hänsyn till.

7.2.1.1 Storlek och position

Pekskärmen eller flerpunktspekskärmen kan inte ersätta musen rakt av eftersom en pekare på skärmen har andra egenskaper än mänskliga fingrar och pekpennor. När ett användargränssnitt utformas till pekskärmar eller flerpunktspekskärmar finns det flera fysiska begränsningar man måste ta hänsyn till. En vuxen män-

niskas fingertoppar är mellan 10 - 14 millimeter breda [36]. Detta medför att hänsyn måste tas när objekt storleksbestäms och positioneras.

Figur 7.1 visar ett exempel på en pekskärm där tre knappar ligger för nära varandra och dessutom är för små. En användare kan ha problem med att vara säker på att denne trycker på rätt knapp och i detta fall är skillnaden mellan "Radera" och "Frigör" mycket stor, där det ena alternativet raderar dokumentet medan det andra skriver ut.



Figur 7.1: Ett dåligt användargränssnitt för pekning

Det är också relevant att ta hänsyn till var objekt placeras på skärmen eftersom användarens hand kan dölja för tillfället intressanta objekt. Ett bra exempel på ett system med pekskärm är SJ ABs biljettkiosker där knappen som tar användaren vidare i köpprocessen är placerad längst ned på skärmen vilket innebär att användaren ser när skärmen uppdateras. I annat fall kan användaren missa att skärmen uppdateras och på så sätt tappa greppet om var i processen man är.

7.2.1.2 Fördröjning

För att en användare inte skall uppleva användandet av mjukvaran som långsamt eller "störande" måste hastigheten på systemet ökas. Det skulle kunna vara nog med att byta hårdvaruprototypen mot något som en tillverkare av flerpunktspekskärmar kan leverera och därefter implementera en tolk för att delge applikationen pekpunkter från denna.

Om det enbart räcker med en ny hårdvaruprototyp för att lösa problemet med hastigheten är inget som vi kan säga då enbart tester med simulator har utförts. Det finns dock stor anledning att tro att en sådan substitution skulle ge effekt eftersom en kommersiell produkt oftast är optimerad för sitt ändamål.

7.3 Lärdomar

Under arbetet har djupare kunskap erhållits inom följande områden:

Visuell spårning — Visuell spårning används för att identifiera pekpunkter på hårdvaruenheten och för att få dessa att fungera har vi behövt lära oss om hur spårare fungerar.

Objective-C - Programspråket Objective-C har använt för att implementera applikationen. Här har vi lärt oss om hur Objective-C är uppbyggt samt hur Apples fönsterhantering, via Cocoa, fungerar.

Historia — Historia kring inmatningssystem har gett oss kunskap om hur människor och datorer förr interagerade med varandra.

Dator-människa-interaktion — Detta är ett område som detta examensarbete har nosat lite på och vi har lärt oss några viktiga egenskaper kring hur människor interagerar med datorer. Dessutom har vi lärt oss om hur användargränssnitt för pekskärmar bör utformas.

Referenser

[1] Robert A. Adams. *Calculus, A Complete Course*. Pearson Education LTD., 2003.

[2] Bill Buxton. Multi-Touch Systems that I Have Known and Loved, Internetsida, Aug 2008. `http://www.billbuxton.com/multitouchOverview.html`, läst Okt. 2008.

[3] Microsoft Corporation. Microsoft Surface, SDK Introduction, Nov 2007. `http://blogs.msdn.com/surface/archive/2007/11/05/sdk-introduction.aspx`, läst Dec. 2008.

[4] Microsoft Corporation. Microsoft Surface, Sep 2008. `http://www.microsoft.com/surface`, läst Sept. 2008.

[5] RAND CORPORATION. Rand 2007 annual report. Årsrapport, 2007.

[6] M.R Davis and T.O Ellis. The rand tablet: A man-machine communication device. pages 1–21, May 1964.

[7] W den Boer, A Abileah, P Green, and T Larsson. 56. 3: Active matrix lcd with integrated optical touch screen. *SID Symposium Digest of Technical Papers*, Jan 2003.

[8] Morrison et al. Passive touch system and method of detecting user input. Patent 20 070 075 982, United States Patent and Trademark Office, Apr 2007.

[9] M Grossman and D Walter. Teaching with interactive computer capabilities (plato: Computer-based education for animal breeding). *Dairy Sci*, (61):1308 – 1311, Sep 1978.

[10] Jefferson Y Han. Low-cost multi-touch sensing through frustrated total internal reflection. *UIST'05*, pages 155–118, 2005.

[11] Apple Inc. Pressmedelande, SAN FRANCISCO, MACWORLD, Jan 2007. http://www.apple.com/pr/library/2007/01/09iphone.html, läst Sept. 2008.

[12] Apple Inc. Video om iPhone, California, CUPERTINO, Dec 2008. http://www.apple.com/se/iphone/guidedtour, läst Dec. 2008.

[13] Apple Inc. iPhone Dev. Center, California, CUPERTINO, Dec 2008. http://developer.apple.com/iphone/, läst Dec. 2008.

[14] Apple Inc. Pressmedelande, California, CUPERTINO, Jul 2008. http://www.apple.com/pr/library/2008/07/14iphone.html, läst Sept. 2008.

[15] Apple Inc. Leopard Guides, Cocoa, Aug 2008. http://developer.apple.com/documentation/Cocoa/index.html, läst Nov. 2008.

[16] Apple Inc. Leopard Reference: Graphics & Imaging, Quartz, Aug 2008. http://developer.apple.com/reference/GraphicsImaging/idxQuartz-date.html, läst Nov. 2008.

[17] Apple Inc. What Is Core Animation?, Maj 2008. http://developer.apple.com/documentation/Cocoa/Conceptual/CoreAnimation_guide/Articles/WhatisCoreAnimation.html, läst Nov. 2008.

[18] Apple Inc. CALayer Class Reference, Maj 2008. http://developer.apple.com/DOCUMENTATION/GraphicsImaging/Reference/CALayer_class/Introduction/Introduction.html, läst Dec. 2008.

[19] Apple Inc. NSMutableArray Class Reference, Okt 2008. http://developer.apple.com/documentation/Cocoa/Reference/Foundation/Classes/nsmutablearray_Class/Reference/Reference.html, läst Dec. 2008.

[20] Apple Inc. NSView Class Reference, Okt 2008. http://developer.apple.com/documentation/Cocoa/Reference/ApplicationKit/Classes/NSView_Class/Reference/NSView.html, läst Dec. 2008.

[21] Tyco Electronics Inc. Internetsida, Sep 2008. http://www.elotouch.com/AboutElo/History, läst Sept. 2008.

[22] S Jordà. Sonigraphical instruments: from fmol to the reactable. *Proceedings of the 2003 conference on New interfaces for ...*, Jan 2003.

[23] M Kaltenbrunner, T Bovermann, and R Bencina. Tuio: A protocol for table-top tangible user interfaces. *Proc. of the The 6th Int'l Workshop on Gesture in Human- ...*, Jan 2005.

[24] M Kaltenbrunner, G Geiger, and S Jordà. Dynamic patches for live musical performance. *Proceedings of the 2004 conference on New interfaces for ...*, Jan 2004.

[25] G Roberts Lawrence. The lincoln wand. *Proceedings of the AFIPS Fall Joint Computer Conference*, pages 223–227, Sep 1966.

[26] S.K Lee, W Buxton, and K.C Smith. A multi-touch three dimensional touch-sensitive tablet. *C H I ' 85 PROCEEDINGS*, pages 21–25, 1985.

[27] Asahi Spectra Co. Ltd. Produkt, California, Torrance, Jan 2007. `http://www.asahi-spectra.com/opticalfilters/syousaik2_ver2.asp?key=YBPA880`, läst Nov. 2008.

[28] Nobuyuki Matsushita and Jun Rekimoto. Holowall: designing a finger, hand, body, and object sensitive wall. *UIST 97*, pages 209–210.

[29] Nima Motamedi. Hd touch: multi-touch and object sensing on a high definition lcd tv. *CHI '08: CHI '08 extended abstracts on Human factors in computing systems*, Apr 2008.

[30] Brad A Myers. A brief history of human computer interaction technology. *ACM interactions*, 5(2):44–54, Mar 1998.

[31] N-trig. Pressmedelande, N-trig's DuoSense™ Digitizer Embedded in Intel's New UrbanMax Mobile Computer, Aug 2008. `http://www.n-trig.com/Content.aspx?Page=PressReleases&PressReleaseId=315`, läst Sept. 2008.

[32] Russell Owen, Gordon Kurtenbach, George Fitzmaurice, Thomas Baudel, and Bill Buxton. When it gets more difficult, use both hands – exploring bimanual curve manipulation. page 8, Mar 2005.

[33] Frank L. Pedrotti and Leno D. Pedrotti. *Introduction to Optics*. Pearson Education LTD., 1993.

[34] Robert Lubkoll Björn Bollensdorff Prof. Marc Alexa, Karl Wessel. LEDs and IR Filters, Project Multi-touch Display, Jan 2008. `http://user.cs.tu-berlin.de/~karlw/LEDSpdf2.pdf`, läst Nov. 2008.

[35] J Rekimoto and N Matsushita. Perceptual surfaces: Towards a human and object sensitive interactive display. *Workshop on Perceptual User Interfaces (PUI'97)*, Jan 1997.

[36] Dan Saffer. *Interactive Gestures: Designing Gestural Interfaces*. O'REILLY, 2008.

[37] George C. Shapiro, Linda G. & Stockman. *Computer Vision*. Prentice Hall, 2002.

[38] B Shneiderman. Touch screens now offer compelling uses. *IEEE Software*, 8(2):93–94, Mar 1991.

[39] Ben Shneiderman and Cathrene Plaisant. *Designing the user inteface*. Addison-Wesley, fourth edition.

[40] Ben Britten Smith. benbritten.com, Nov 2008. `http://benbritten.com/blog/?s=BBOSC`, läst Nov. 2008.

[41] Norbert A Streitz, Jörg Geißler, Torsten Holmer, Shinichi Konomi, Christian Müller-Tomfelde, Wolfgang Reischl, Petra Rexroth, Peter Seitz, and Ralf Steinmetz. i-land: an interactive landscape for creativity and innovation. *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 120–127, 1999.

[42] Entertainment Design TED, Technology. Speakers Jeff Han: Human-computer interface designer, Sep 2008. `http://www.ted.com/index.php/speakers/jeff_han.html`, läst Sept. 2008.

[43] P Wellner. The digitaldesk calculator: Tangible manipulation on a desk top display. *ACM UIST*, pages 27–33, Nov 1991.

[44] Matthew Wright. Open Sound Control 1.0 Specification, Sep 2002. `http://opensoundcontrol.org/spec-1_0`, läst Okt. 2008.

[45] Matthew Wright, Adrian Freed, and Ali Momeni. Opensoundcontrol: Stateof theart 2003. *Proceedings of the 2003 Conference on New Interfaces for Musical Expression*, pages 154–159, May 2003.

Bilaga A
Figurreferenser

Figurer nämnda i denna bilaga ägs av respektive upphovsman nedan, övriga figurer är framtagna för denna uppsats och får användas fritt.

Karlstads universitets sol: Karlstads universitet

Figur 2.2: RAND Corporation

Figur 2.3: Wikipedia, GNU Free Documentation License

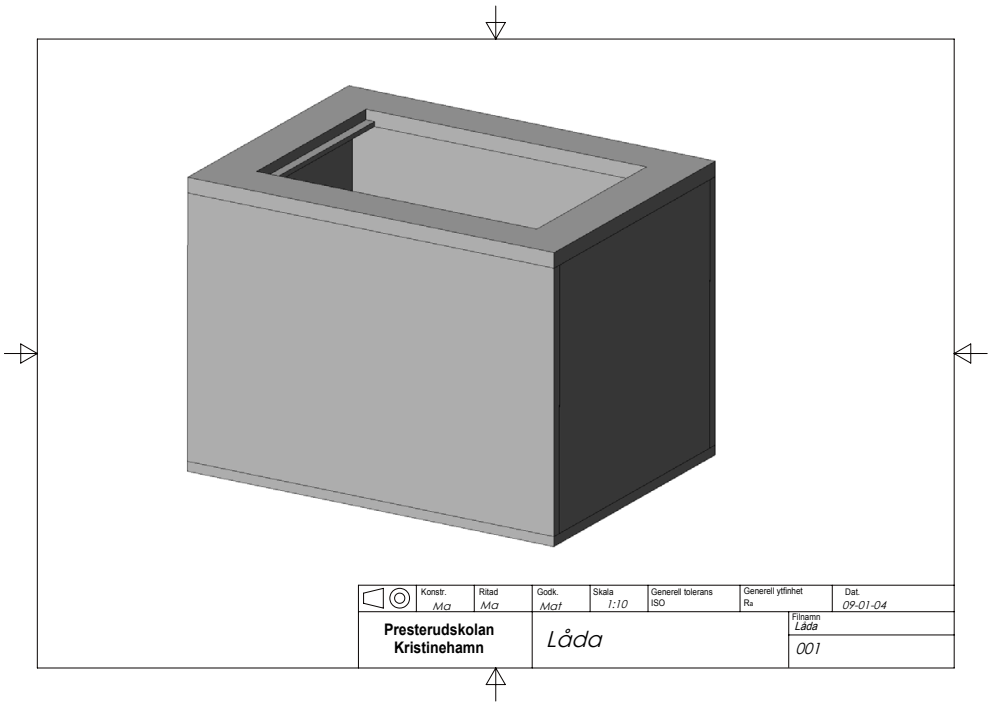
Figur 2.4: Xavier Sivecas, ReactTable*

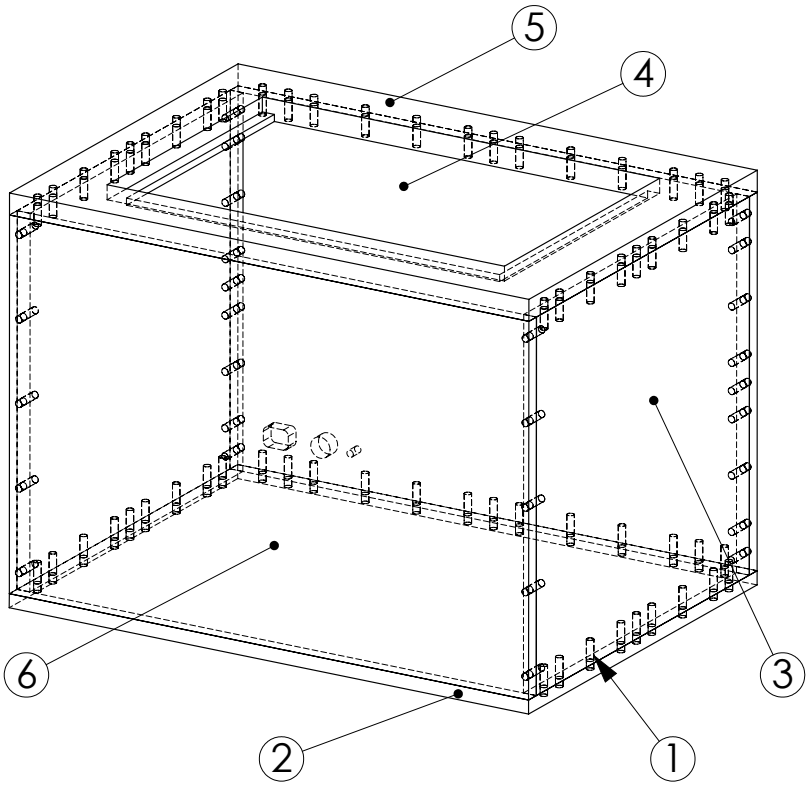
Figur 2.5: Apple Inc.

Figur 2.6: Microsoft Corporation

Figur 4.2: Microsoft Corporation

Bilaga B
Ritningar för chassi

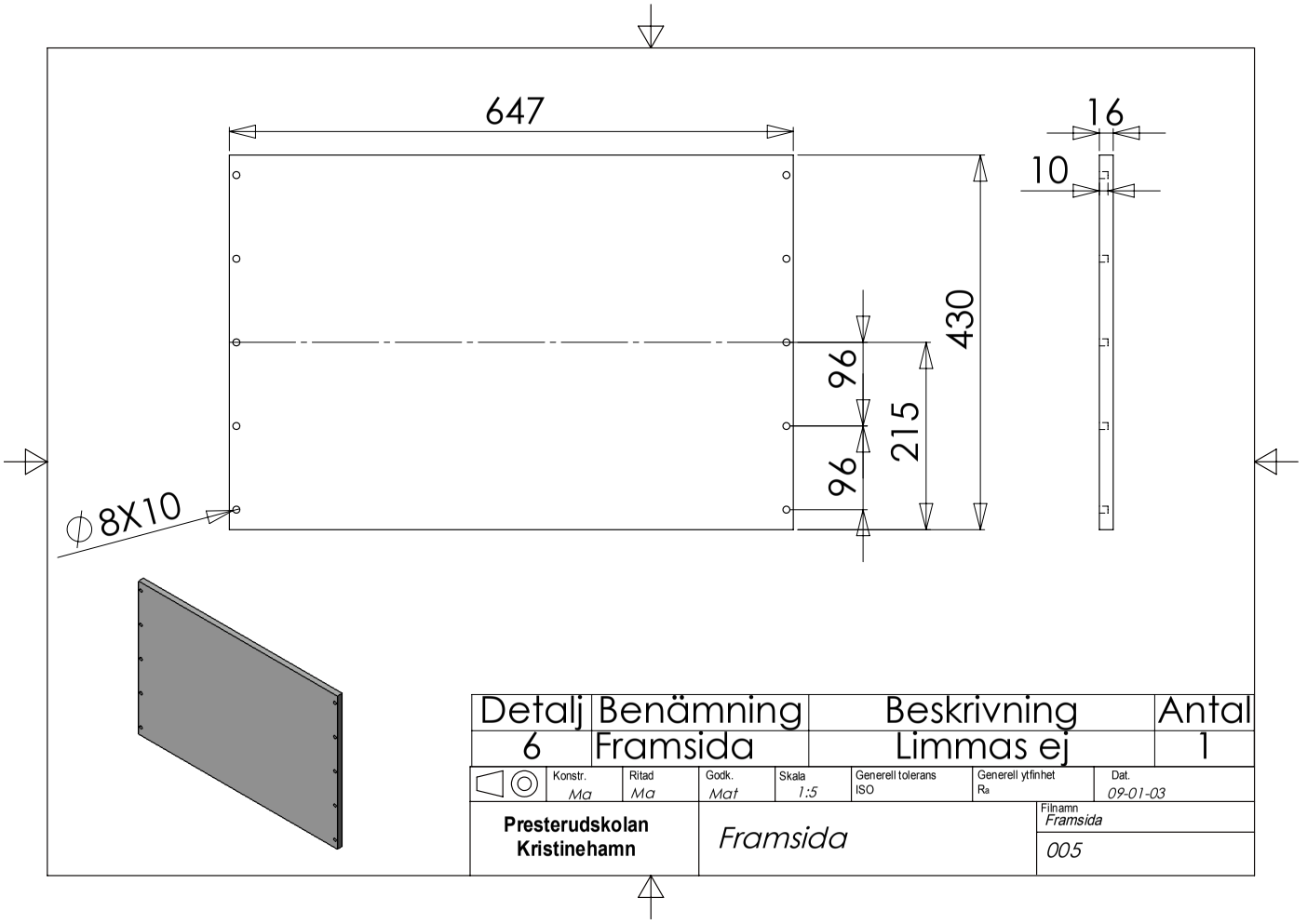




Detalj	Benämning	Beskrivning	Antal
1	plugg 8 X30		90
2	Botten		1
3	Sida		2
4	Baksida		1
5	Lock		1
6	Framsida	Limmas ej	1

Konstr.	Ritad	Godk.	Skala	Generell tolerans	Generell ytfinhet	Dat.
Ma	*Ma*	*Mat*	*1:5*	ISO 2768	R_a	*09-01-04*

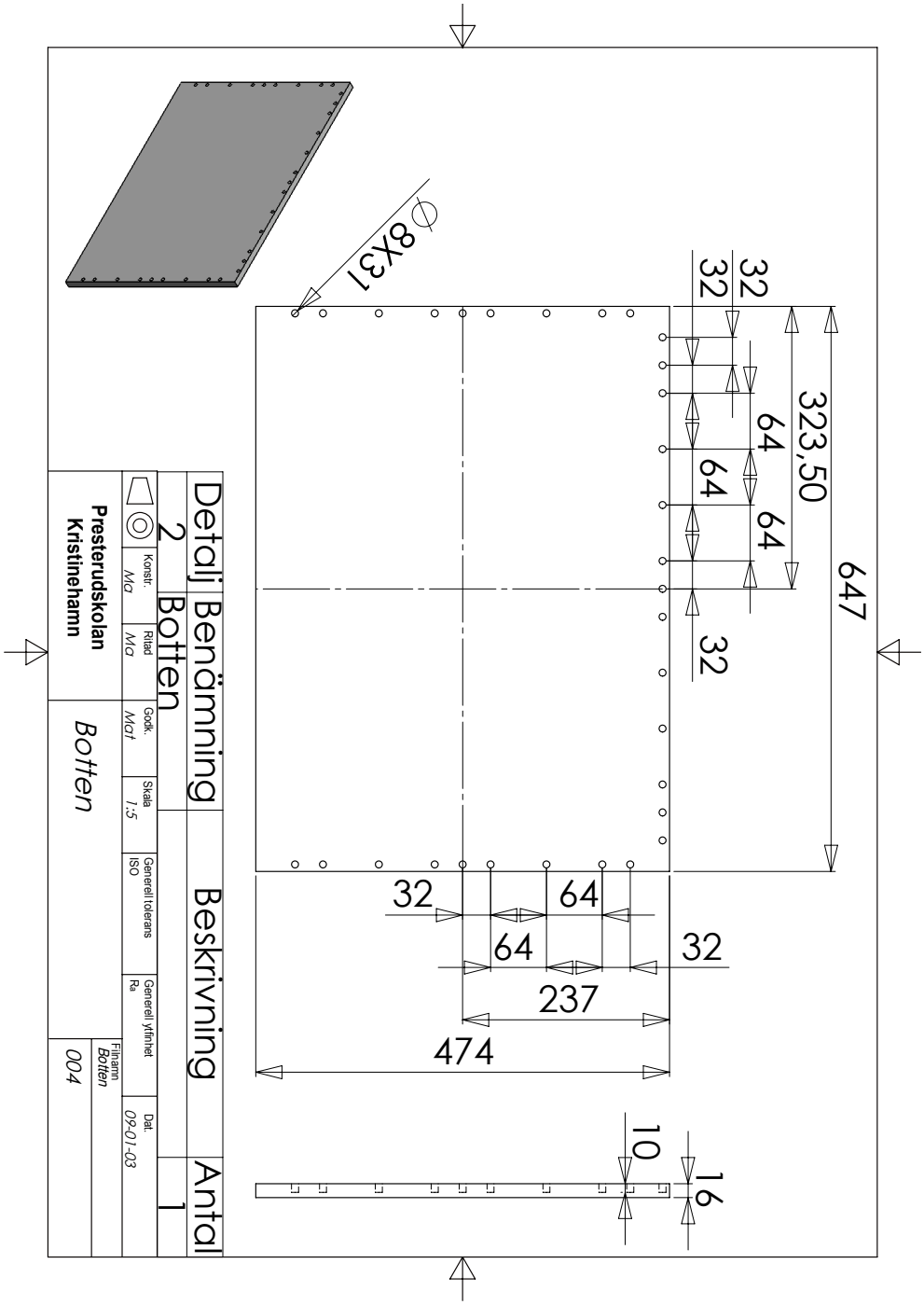
Presterudskolan Kristinehamn	*Låda*	Filnamn *Låda sammanställningsritning*
		002

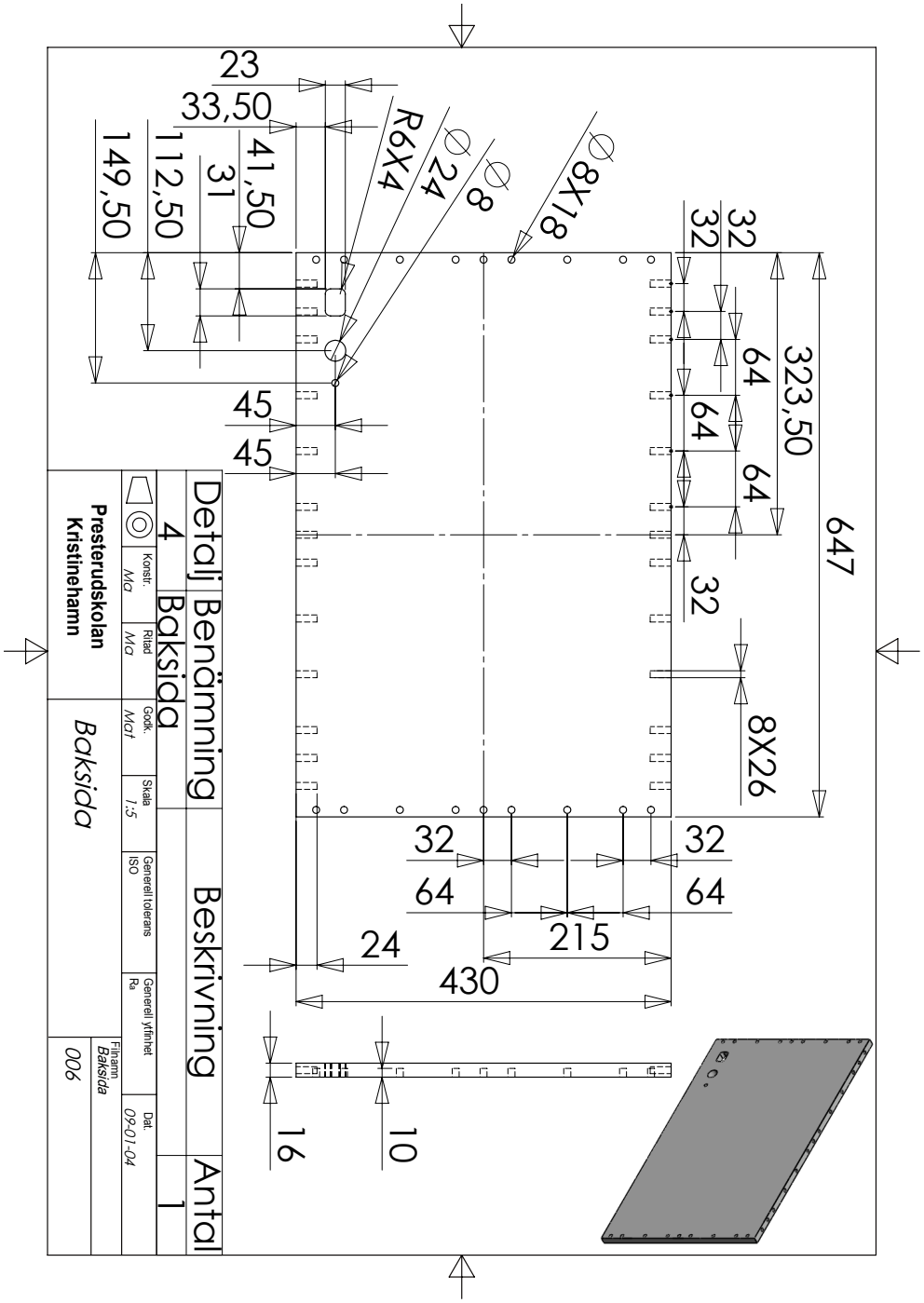


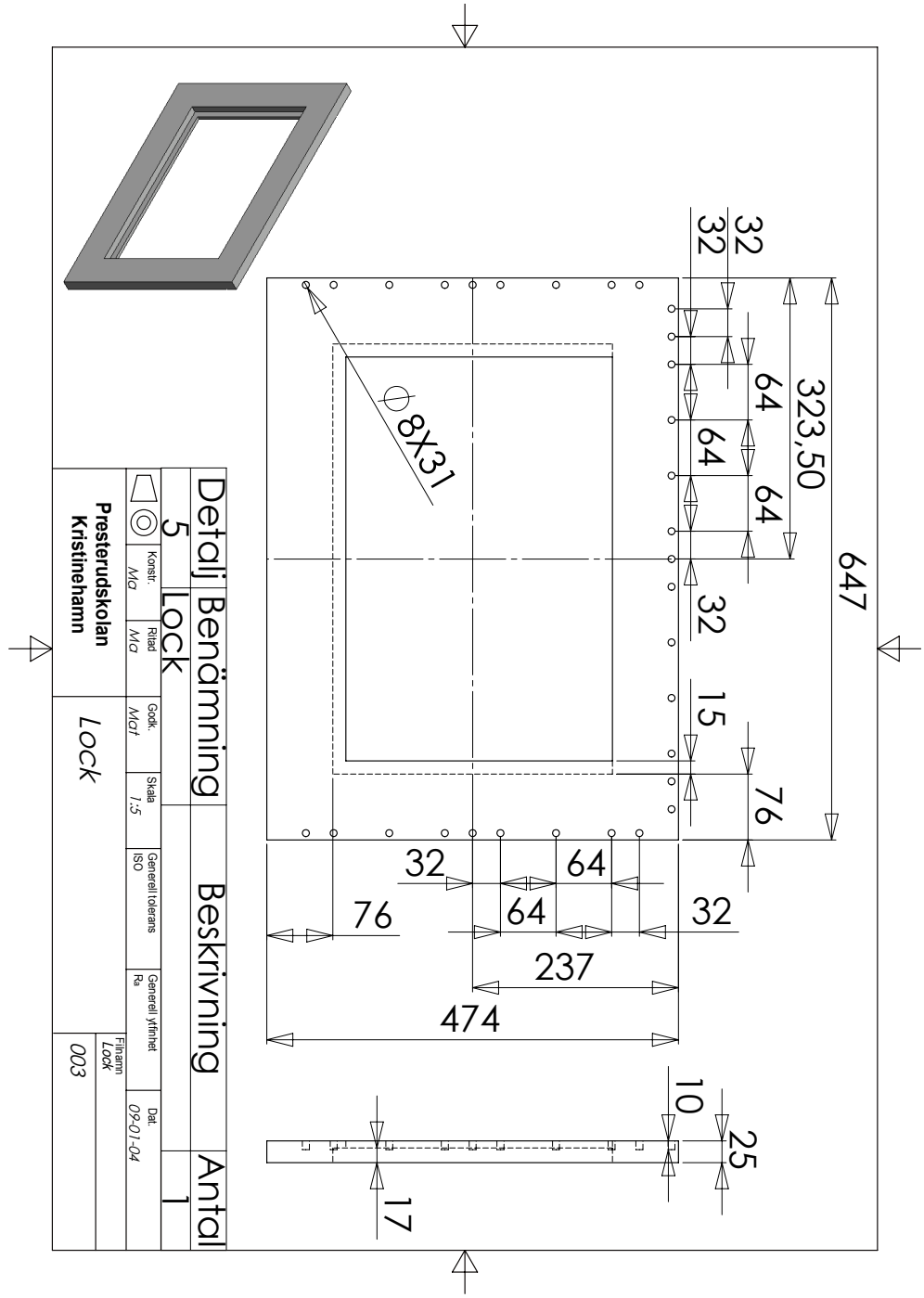
Detalj	Benämning	Beskrivning	Antal
6	Framsida	Limmas ej	1

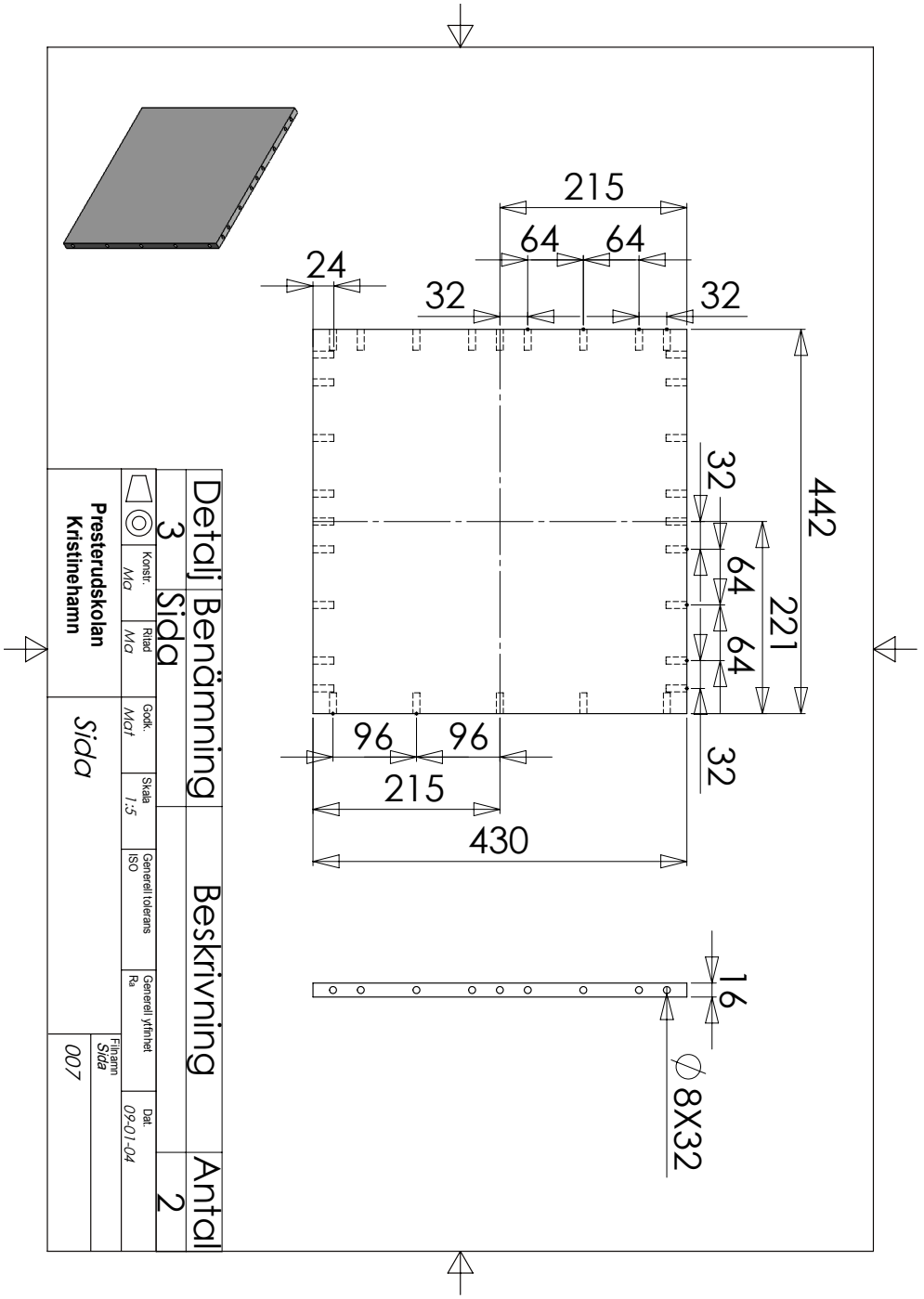
Konstr.	Ritad	Godk.	Skala	Generell tolerans	Generell ytfinhet	Dat.
Ma	Ma	Mat	1:5	ISO	Ra	09-01-03

Presterudskolan Kristinehamn	*Framsida*	Filnamn *Framsida*
		005


Detalj Benämning Beskrivning Antal
2 Botten 1
Presteruds kolan
Kristinehamn
Botten
Konstr.
Ma
Ritad
Ma
Godk.
Mat
Skala
1:5
Generell tolerans
ISO
Generell ytfinhet
Ra
Dat.
09-01-03
Filnamn
Botten
004
Ø8x31
32
32
323,50
64
64
64
32
647
32
64
64
32
237
474
10
16


23
33,50
41,50
31
112,50
149,50
R6X4
Ø24
Ø8
Ø8X18
32
32
64
64
64
323,50
647
45
45
32
8X26
32
32
64
64
24
215
430
16
10
Detalj Benämning Beskrivning Antal
4 Baksida 1
Presterudskolan
Kristinehamn
Konstr. MG
Ritad MG
Godk. MGt
Skala 1:5
Generell tolerans ISO
Generell ytfinhet Ra
Dat. 09-01-04
Filnamn Baksida
Baksida
006


Detalj Benämning Beskrivning Antal
5 Lock 1
Presteruds kolan
Kristinehamn
Lock
Konstr: MCa
Ritad MCa
Godk. MCt
Skala 1:5
Generell tolerans ISO
Generell ytfinhet Ra
Dat. 09-01-04
Filnamn Lock
003
Φ8X31
32
32
64
64
64
323,50
647
32
15
76
32
64
76
64
32
237
474
10
25
17


215
64
64
24
32
32
442
32
64
64
221
32
96
96
215
430
16
⌀ 8X32
Detalj Benämning
Beskrivning
Antal
3
Sida
2
Konstr.
MG
Ritad
MG
Godk.
MGt
Skala
1:5
Generell tolerans
ISO
Generell ytfinhet
Ra
Dat.
09-01-04
Filnamn
Sida
007
Presterudskolan
Kristinehamn
Sida

Bilaga C
Programkod, två objekt placeras

```
/*
 * Create a square content.
 * This object is responsible for drawing a square in the CALayer
 */
LimeRectangleContent *mySquare;
mySquare = [[LimeRectangleContent alloc]
           initWithColor:[NSColor cyanColor]
               andHeight:180
                andWidth:180];
/*
 * Create a triangle content.
 * This object is responsible for drawing a
 *  triangle in the CALayer.
 */
LimeTriangleContent *myTriangle;
myTriangle = [[LimeTriangleContent alloc]
             initWithColor:[NSColor magentaColor]
                 andHeight:180
                   andBase:200];
/*
 * Create a layer with the basic manipulation operations.
 * Let mySquare be the content of this layer.
 */
LimeBasicManipulatableLayer *mySquareLayer;
mySquareLayer = [[LimeBasicManipulatableLayer alloc]
                initWithManipulationType:LimeMoveRotateScale
                              andContent:mySquare
                                andFrame:NSMakeRect(100, 100,
                                    180, 180)];
/*
```

```
 * Create a layer with the basic manipulation operations.
 * Let myTriangle be the content of this layer.
 */
LimeBasicManipulatableLayer *myTriangleLayer;
myTriangleLayer = [[LimeBasicManipulatableLayer alloc]
                  initWithManipulationType:LimeMoveRotateScale
                                andContent:myTriangle
                                  andFrame:NSMakeRect(180, 200,
                                      200, 200)];
/*
 * Register the layers with the layercontroller.
 * This will make them appear on screen.
 */
[layerController addLayer:mySquareLayer];
[layerController addLayer:myTriangleLayer];

[mySquare release];
[myTriangle release];
[mySquareLayer release];
[myTriangleLayer release];
```