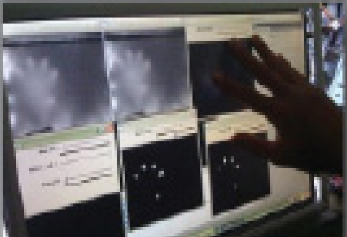
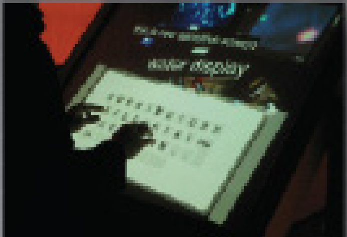
Multitouch Technologies





Multitouch Technologies
Version 1.0

Multi-Touch Technologies

NUI Group Authors

1st edition [Community Release]: May 2009

Cover design: Justin Riggio

Contact: the-book@nuigroup.com

Contents

Introduction to Multi-Touch

In this chapter:

About this book
Authors

About this book

Founded in 2006, Natural User Interface Group (~ NUI Group) is an interactive media community researching and creating open source machine sensing techniques to benefit artistic, commercial and educational applications. It's the largest online (~5000+) open source community related to the open study of Natural User Interfaces.

NUI Group offers a collaborative environment for developers that are interested in learning and sharing new HCI (Human Computer Interaction) methods and concepts. This may include topics such as: augmented reality, voice/handwriting/gesture recognition, touch computing, computer vision, and information visualization.

This book is a collection of wiki entries chapters written exclusively for this book, making this publication the first of its kind and unique in its field. All contributors are listed here and countless others have been a part the community that has developed and nurtured the NUI Group multi-touch community, and to them we are indebted.

We are sure you'll find this book pleasant to read. Should you have any commnts, criticisms or section proposals, please feel free to contact us via the-book@nuigroup.com. Our doors are always open and looking for new people with similar interests and dreams.

About the authors

Alex Teiche is an enthusiast that has been working with Human-Computer Interaction for several years, with a very acute interest in collaborative multiuser interaction. He recently completed his LLP Table, and has been contributing to the PyMT project for several months. He is now experimenting with thin multi-touch techniques that are constructable using easily accessible equipment to the average hobbyist.

Ashish Kumar Rai is an undergraduate student in Electronics Engineering at Institute of Technology, Banaras Hindu University, India. His major contribution is the development of the simulator QMTSim. Currently his work consists of development of innovative hardware and software solutions for the advancement of NUI.

Chris Yanc has been a professional interactive designer at an online marketing agency in Cleveland, OH for over 4 years. After constructing his own FTIR multi-touch table, he has been focusing on using Flash and Actionscript 3.0 to develop multi-touch applications from communication with CCV and Touchlib. Chris has been sharing the progress of his multi-touch Flash experiments with the NUIGroup community.

Christian Moore is a pioneer in open-source technology and is one of the key evangelists of Natural User Interfaces. As the founder of NUI Group, he has led a community that has developed state-of-the-art human sensing techniques and is working on the creation of open standards to unify the community's efforts into well-formatted documentation and frameworks.

Donovan Solms is a Multimedia student at the University of Pretoria in Gauteng, South Africa. After building the first multitouch screen in South Africa in the first quarter of 2007, he continued to develop software for multitouch using .NET 3 and ActionScript 3. He can currently be found building commercial multitouch surfaces and software at Multimedia Solutions in South Africa.

Görkem Çetin defines himself an open source evangelist. His current doctorate studies focus on human computer interaction issues of open source software. Çetin has authored 5 books, written numerous articles for technical and sector magazines and spoken at various conferences.

Justin Riggio is a professional interactive developer and designer that has been working freelance for over 5 years now. He lives and works in the NYC area and can be found at the Jersey shore surfing at times. He has designed and built a 40" screen DI multi touch table and has set up and tested LLP LCD set ups and also projected units. You can find on his drafting table a new plan for a DSI system and AS3 multi touch document reader.

Nolan Ramseyer is a student at University of California, San Diego in the United States studying Bioengineering: Biotechnology with a background in computers and multimedia. His work in optical-based multi-touch has consumed a larger part of his free time and through his experimentations and learning has helped others achieve their own projects and goals.

Paul D'Intino is the Technical Training Manager at Pioneer Electronics in Melbourne, Australia. In 2003 he completed a Certificate III in Electrotechnology Entertainment and Servicing, and went onto repairing Consumer Electronics equipment, specializing in Plasma TV's. His work in both hardware and software for multi-touch systems grew from his passion for electronics, which he aims to further develop with help from the NUI Group.

Laurence Muller (M.Sc.) is a scientific programmer at the Scientific Visualization and Virtual Reality Research Group (Section Computational Science) at the University of Amsterdam, Amsterdam in the Netherlands. In 2008 he completed his thesis research project titled: "Multi-touch displays: design, applications and performance evaluation". Currently he is working on innovative scientific software for multi-touch devices.

Ramsin Khoshabeh is a graduate Ph.D. student in the University of California, San Diego in the Electrical and Computer Engineering department. He works in the Distributed-Cognition and Human-Computer Interaction Laboratory in the Cognitive Science department. His research is focused on developing novel tools for the interaction with and analysis of large datasets of videos. As part of his work, he has created MultiMouse, a multi-touch Windows mouse interface, and VideoNavigator, a cross-platform multi-touch video navigation interface.

Rishi Bedi is a student in Baltimore, Maryland with an interest in emerging user interaction technologies, programming in several languages, and computer hardware. As an actively contributing NUI Group community member, he hopes to build his own multi-touch devices & applications, and to continue to support the community's various endeavors. Also interested in graphic design, he has designed this book and has worked in various print and electronic mediums.

Mohammad Taha Bintahir is an undergraduate student at University of Ontario Institute of Technology, in Toronto Canada studying Electrical Engineering and Business and focusing on microelectronics and programming. He's currently doing personal research for his thesis project involving, both electronic and optical based Multi-touch and multi-modal hardware systems.

Thomas Hansen is PhD. Student at the University of Iowa and open source enthusiast. His research interests include Information Visualization, Computer Graphics, and especially Human Computer Interaction directed at therapeutic applications of novel interfaces such as multi-touch displays. This and his other work on multi-touch interfaces has been supported by the University of Iowa's Mathematical and Physical Sciences Funding Program.

Tim Roth studied Interaction Design at the Zurich University of the Arts. During his studies, he focussed on interface design, dynamic data visualization and new methods of human computer interaction.His diploma dealt with the customer and consultant situation in the private banking sector. The final output was a multitouch surface, that helped the consultant to explain and visualize the complexeties of modern financial planning.

Seth Sandler is a University of California, San Diego graduate with a Bachelors degree in Interdisciplinary Computing and the Arts, with an emphasis in music. During Seth's final year of study he began researching multitouch and ultimatley produced a musical multi-touch project entitled 'AudioTouch.' Seth is most notably known for his presence on the NUI Group forum, introduction of the 'MTmini', and CCV development.

Editors

This book was edited & designed by Görkem Çetin, Rishi Bedi and Seth Sandler.

Copyright

Chapter 1

Multi-Touch Technologies

In this chapter:

1.1 Introduction to Hardware

Multi-touch (or multitouch) denotes a set of interaction techniques that allow computer users to control graphical applications with several fingers. Multi-touch devices consist of a touch screen (e.g., computer display, table, wall) or touchpad, as well as software that recognizes multiple simultaneous touch points, as opposed to the standard touchscreen (e.g. computer touchpad, ATM), which recognizes only one touch point.

The Natural User Interface and its influence on multi-touch gestural interface design has brought key changes in computing hardware design, especially the creation of "true" multi-touch hardware systems (i.e. support for more than two inputs). The aim of NUI Group is to provide an open platform where hardware and software knowledge can be exchanged freely; through this free exchange of knowledge and information, there has been an increase in development in regards to hardware.

On the hardware frontier, NUI Group aims to be an informational resource hub for others interested in prototyping and/or constructing, a low cost, high resolution open-source multi-input hardware system. Through the community research efforts, there have been improvements to existing multi-touch systems as well as the creation of new techniques that allow for the development of not only multitouch hardware systems, but also multi-modal devices. At the moment there are five major techniques being refined by the community that allow for the creation of a stable multi-touch hardware systems; these include: Jeff Han's pioneering Frustrated Total Internal Reflection (FTIR), Rear Diffused Illumination (Rear DI) such as Microsoft's Surface Table, Laser Light Plan (LLP) pioneered in the community by Alex Popovich and also seen in Microsoft's LaserTouch prototype, LED-Light Plane (LED-LP) developed within the community by Nima Motamedi, and finally Diffused Surface Illumination (DSI) developed within the community by Tim Roth.

These five techniques being utilized by the community work on the principal of Computer Vision and optics (cameras). While optical sensing makes up the vast majority of techniques in the NUI Group community, there are several other sensing techniques that can be utilized in making natural user interface and multitouch devices. Some of these sensing devices include proximity, acoustic, capacitive, resistive, motion, orientation, and pressure. Often, various sensors are combined to form a particular multitouch sensing technique. In this chapter, we explore some of the mentioned techniques.

1.2 Introduction to Optical Multi-Touch Technologies

Optical or light sensing (camera) based solutions make up a large percentage of multi-touch devices. The scalability, low cost and ease of setup are suggestive reasoning for the popularity of optical solutions. Stereo Vision , Overhead cameras, Frustrated Total Interal Reflection, Front and Rear Diffused Illumination, Laser Light Plane, and Diffused Surface Illumination are all examples of camera based multi-touch systems.

Each of these techniques consist of an optical sensor (typically a camera), infrared light source, and visual feedback in the form of projection or LCD. Prior to learning about each particular technique, it is important to understand these three parts that all optical techniques share.

1.2.1 Infrared Light Sources

Infrared (IR in short) is a portion of the light spectrum that lies just beyond what can be seen by the human eye. It is a range of wavelengths longer than visible light, but shorter than microwaves. 'Near Infrared' (NIR) is the lower end of the infrared light spectrum and typically considered wavelengths between 700nm and 1000nm (nanometers). Most digital camera sensors are also sensitive to at least NIR and are often fitted with a filter to remove that part of the spectrum so they only capture the visible light spectrum. By removing the infrared filter and replacing it with one that removes the visible light instead, a camera that only sees infrared light is created.

In regards to multitouch, infrared light is mainly used to distinguish between a visual image on the touch surface and the object(s)/finger(s) being tracked. Since most systems have a visual feedback system where an image from a projector, LCD or other display is on the touch surface, it is important that the camera does not see this image when attempting to track objects overlyaed on the display. In order to separate the objects being tracked from the visual display, a camera, as explained above, is modified to only see the infrared spectrum of light; this cuts out the visual image (visible light spectrum) from being seen by the camera and therefore, the camera is able to see only the infrared light that illuminates the object(s)/finger(s) on the touch surface.

Many brands of acrylic sheet are intentionally designed to reduce their IR transmission above 900nm to help control heat when used as windows. Some cameras sensors have either dramatically greater, or dramatically lessened sensitivity to 940nm, which also has a slightly reduced occourance in natural sunlight.

IR LEDs are not required, but a IR light source is. In most multitouch optical techniques (especially LED-LP and FTIR), IR LEDs are used because they are efficient and effective at providing infrared light. On the other hand, Diffused Illumination (DI) does not require IR LEDs, per se, but rather, some kind of infrared light source like an infrared illuminator (which may have LEDs inside). Laser light plane (LLP) uses IR lasers as the IR light source.

Most of the time, LEDs can be bought in forms of "single LEDs" or "LED ribbons":

- Single LEDs: Single through-hole LEDs are a cheap and easy to create LED frames when making FTIR, DSI, and LED-LP MT setups. They require a knowledge in soldering and a little electrical wiring when constructing. LED calculators make it easy for people to figure out how to wire the LEDs up. The most commonly through-hole IR LED used are the OSRAM SFH 485 P. If you are trying to make a LCD FTIR, you will need brighter than normal IR LEDs, so these are probably your best bet.

- LED ribbons: The easiest solution for making LED frames instead of soldering a ton of through-hole LEDs, LED ribbons are FFC cables with surface mount IR LEDs already soldered on them. They come with an adhesive side that can be stuck into a frame and wrapped around a piece of acrylic with a continuous LED ribbon. All that is required is wrap and plug the ribbon into the power adapter and done. The best quality ribbons can be found at environmentallights.com.

- LED emitters: When making Rear DI or Front DI setups, pre-built IR emitters are mush easier than soldering a ton of single through-hole LEDs together. For most Rear DI setups, 4 of these are usually all that is needed to completely cover the insides of the box. By buying the grouped LEDs, you will have to eliminate "hot spot" IR blobs caused by the emitters by bouncing the IR light off the sides and floor of the enclosed box.

Before buying LEDs, it's strongly advised to check the data sheet of the LED. Wavelength, angle, and radiant intensity are the most important specifications for all techniques.

- Wavelength: 780-940nm. LEDs in this range are easily seen by most cameras and visible light filters can be easily found for these wavelengths. The lower the wavelength, the higher sensitivity which equates to a higher ability to determine the pressure.

- Radiant intensity: Minimum of 80mw. The ideal is the highest radiant intensity you can find, which can be much higher than 80mw.
- Angle for FTIR: Anything less than +/- 48 will not take full advantage of total internal reflection, and anything above +/- 48 degrees will escape the acrylic. In order to ensure there is coverage, going beyond +/- 48 degrees is fine, but anything above +/- 60 is really just a waste as (60 - 48 = +/- 12 degrees) will escape the acrylic.
- Angle for diffused illumination: Wider angle LEDs are generally better. The wider the LED angle, the easier it is to achieve even illumination.

For the DI setup, many setups bump into problem of hotspots. In order to eliminate this, IR light needs to be bounced off the bottom of the box when mounted to avoid IR hot spots on the screen. Also, a band pass filter for the camera is required – homemade band pass filters such as the exposed negative film or a piece of a floppy disk will work, but they'll provide poor results. It's always better to buy a (cheap) band pass filter and putting it into the lens of the camera.

1.2.2 Infrared Cameras

Simple webcams work very well for multitouch setups, but they need to be modified first. Regular webcams and cameras block out infrared light, letting only visible light in. We need just the opposite. Typically, by opening the camera up, you can simply pop the filter off, but on expensive cameras this filter is usually applied directly to the lens and cannot be modified.

Most cameras will show some infrared light without modification, but much better performance can be achieved if the filter is replaced.

The performance of the multi-touch device depends on the used components. Therefore it is important to carefully select your hardware components. Before buying a camera it is important to know for what purpose you will be using it. When you are building your first (small) test multi-touch device, the requirements may be lower than when you are building one that is going to be used for demonstration purposes.

Resolution: The resolution of the camera is very important. The higher the resolution the more pixels are available to detect finger or objects in the camera image. This is very important for the precision of the touch device. For small multi-touch surfaces a low resolution webcam (320 x 240 pixels) can be sufficient. Larger surfaces require cameras with a resolution of 640x480 or higher in order to maintain the precision.

Frame rate: The frame rate is the number of frames a camera can take within one second. More snapshots means that we have more data of what happened in a specific time step. In order to cope with fast movements and responsiveness of the system a camera with at least a frame rate of 30 frames per second (FPS) is recommended. Higher frame rates provide a smoother and more responsive experience.

Interface: Basically there are two types of interfaces that can be used to connect a camera device to a computer. Depending on the available budget one can choose between a consumer grade webcam that uses a USB interface or a professional camera that is using the IEEE 1394 interface (which is also known as FireWire). An IEEE 1394 device is recommended because it usually has less overhead and lower latency in transferring the camera image to the computer. Again, lower latency results in a more responsive system.

Lens type: Most consumer webcams contain an infrared (IR) filter that prevents IR light from reaching the camera sensor. This is done to prevent image distortion. However for our purpose, we want to capture and use IR light. On some webcams it is possible to remove the IR filter. This filter is placed behind the lens and often has a red color. If it is not possible to remove the IR filter, the lens has to be replaced with a lens without coating. Webcams often use a M12 mount. Professional series cameras (IEEE 1394) often come without a lens. Depending on the type it is usually possible to use a M12, C or CS mount to attach the lens.

Choosing the right lens can be a difficult task, fortunately many manufactures provide an online lens calculator. The calculator calculates the required focal length based on two input parameters which are the distance between the lens and the object (touch surface) and the width or height of the touch surface. Be sure to check if the calculator chooses a proper lens. Lenses with a low focal length often suffer from severe image distortion (Barrel distortion / fish eye), which can complicate the calibration of the touch tracking software.

Camera sensor & IR bandpass filter: Since FTIR works with IR light we need to check if our webcam is able to see IR light. Often user manuals mention the sensor name. Using the sensor name one can find the data sheets of the camera sensor. The data sheet usually contains a page with a graph similar as below. This image is belongs to the data sheet of the Sony ICX098BQ CCD sensor [Fig. 1].

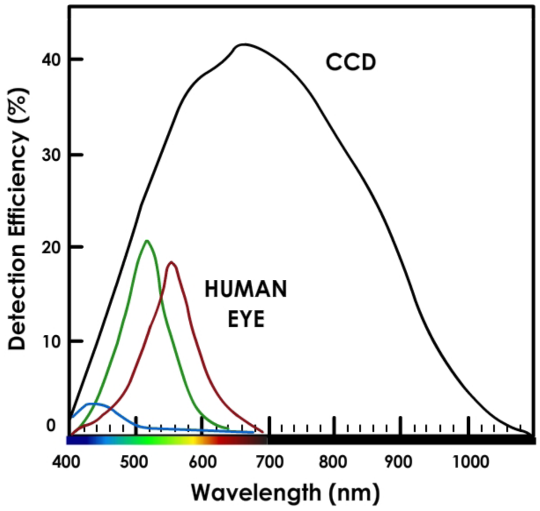


Figure 1: Sensitivity of a Sony CCD sensor

The graph shows the spectral sensitivity characteristics. These characteristics show how sensitive the sensor is to light from specific wavelegths. The wave length of IR light is between 700 and 1000 nm. Unfortunately the image example only shows a range between 400 and 700 nm.

Before we can actually use the camera it is required to add a bandpass filter. When using the (IR sensitive) camera, it will also show all other colors of the spectrum. In order to block this light one can use a cut-off filter or a bandpass filter. The cut-off filter blocks light below a certain wave length, the bandpass filter only allows light from a specific wavelength to pass through.

Bandpass filters are usually quite expensive, a cheap solution is to use overexposed developed negatives.

Recommended hardware

When using a USB webcam it is recommended to buy either of the following:

- The Playstation 3 camera (640x480 at >30 FPS). The filter can be removed and higher frame rates are possible using lower resolution.
- Philips SPC 900NC (640x480 at 30 FPS). The filter cannot be removed, it is

required to replace the lens with a different one.

When using IEEE 1394 (Firewire) it is recommended to buy:

- Unibrain Fire-i (640x480 at 30 FPS) a cheap low latency camera. Uses the same sensor as the Philips SPC 900NC.
- Point Grey Firefly (640x480 at 60 FPS)

Firewire cameras have some benefits over normal USB webcams:

- Higher framerate
- Capture size
- Higher bandwidth
- Less overhead for driver (due to less compression)

1.3 Frustrated Total Internal Reflection (FTIR)

FTIR is a name used by the multi-touch community to describe an optical multi-touch methodology developed by Jeff Han (Han 2005). The phrase actually refers to the well-known underlying optical phenomenon underlying Han's method. Total Internal Reflection describes a condition present in certain materials when light enters one material from another material with a higher refractive index, at an angle of incidence greater than a specific angle (Gettys, Keller and Skove 1989, p.799). The specific angle at which this occurs depends on the refractive indexes of both materials, and is known as the critical angle, which can be calculated mathematically using Snell's law.

When this happens, no refraction occurs in the material, and the light beam is totally reflected. Han's method uses this to great effect, flooding the inside of a piece of acrylic with infrared light by trapping the light rays within the acrylic using the principle of Total Internal Reflection. When the user comes into contact with the surface, the light rays are said to be frustrated, since they can now pass through into the contact material (usually skin), and the reflection is no longer total at that point. [Fig. 2] This frustrated light is scattered downwards towards an infrared webcam, capable of picking these 'blobs' up, and relaying them to tracking software.

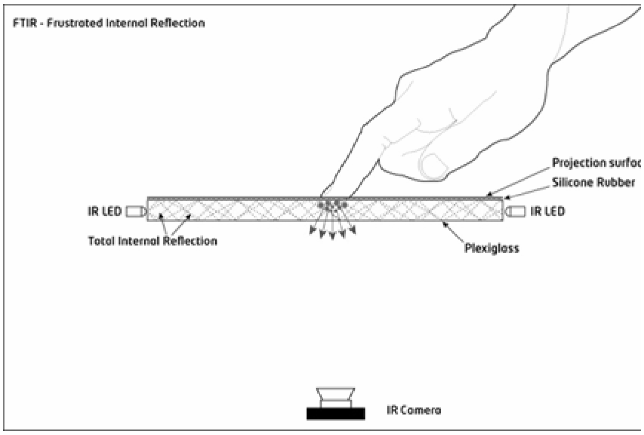


Figure 2: FTIR schematic diagram depicting the bare minimum of parts needed for a FTIR setup.

1.3.1 FTIR Layers

This principle is very useful for implementing multi-touch displays, since the light that is frustrated by the user is now able to exit the acrylic in a well defined area under the contact point and becomes clearly visible to the camera below.

Acrylic

According to the paper of Han, it is necessary to use acrylic for the screen. The minimum thickness is 6 mm however large screens should use 1 cm to prevent the screen from bending.

Before a sheet of acrylic can be used for a multi-touch screen it needs to be prepared. Because acrylic often gets cut up roughly, it is required to polish the sides of the sheet. This is done to improve the illumination from the sides. To polish the sheet it is recommend to use different kinds of sandpaper. First start with a fine sandpaper to remove most of the scratches, after that continue with very fine, super fine and even wet sandpaper. To make your sheets shine you can use Brasso.

Baffle

The baffle is required to hide the light that is leaking from the sides of the LEDs. This can be a border of any material (wood/metal).

Diffuser

Without a diffuser the camera will not only see the touches, but also all objects behind the surface. By using a diffuser, only bright objects (touches) will be visible to the camera. All other 'noise data' will be left out.

Compliant layer

With a basic FTIR setup, the performance mainly depends on how greasy the fingertips of the user are. Wet fingers are able to make better contact with the surface. Dry fingers and objects won't be able to frustrate the TIR. To overcome this problem it is recommended to add a 'compliant layer' on top of the surface. Instead of frustrating the total internal reflection by touch, a compliant layer will act as a proxy. The complaint layer can be made out of a silicon material such as ELASTOSIL M 4641. To protect and improve the touch surface, rear projection material such as Rosco Gray #02105 can be used. With this setup it is no longer required to have a diffuser on the rear side.

1.3.2 Compliant Surfaces

The compliant surface is an overlay placed above the acrylic waveguide in a FTIR based multi-touch system. The compliant surface overlay needs to be made of a material that has a higher refractive index than that of the acrylic waveguide, and one that will "couple" with the acrylic surface under pressure and set off the FTIR effect, and then "uncouple" once the pressure is released. Note that compliant surfaces are only needed for FTIR - not for any other method (DI, LLP, DSI). The compliant surface overlay can also be used as a projection screen.

The compliant surface or compliant layer is simply an additional layer between the projection surface and the acrylic. It enhances the finger contact and gives you more robust blobs, particularly when dragging as your finger will have less adhesion to the surface. In the FTIR technique, the infrared light is emitted into side the acrylic waveguide, the light travels inside the medium (due to total internal refection much like a fibre optic cable), when you touch the surface of the acrylic (you frustrate this TIR effect) causing the light to refract within the medium on points of contact and creating the blobs (bright luminescent objects). There is much experimentation ongoing in the quest for the 'perfect compliant layer'.

Some materials used to success include Sorta Clear 40 and similar catalyzed silicon rubbers, lexel, and various brands of RTV silicone. Others have used fabric based solutions like silicon impregnated interfacing and SulkySolvy.

The original successful method, still rather popular, is to 'cast' a smooth silicone surface directly on top of your acrylic and then lay your projection surface on that after it cures. This requires a material that closely approximates the optical properties of the acrylic as it will then be a part of the acrylic as far as your transmitted IR is concerned, hence the three 'rubber' materials mentioned earlier...they all have a refractive index that is slightly higher but very close to that of acrylic. Gaining in popularity is the 'Cast Texture' method. Tinkerman has been leading the pack in making this a simple process for DIY rather than an involved commercial process. But essentially, by applying the compliant layer to the underside of the projection surface, and texturing it, then laying the result on acrylic, you gain several benefits.

The compliant surface is no longer a part of the acrylic TIR effect so you are no longer limited to materials with a similar refractive index to that of acrylic, although RTV and Lexel remain the most popular choices, edging out catalyzed silicons here. Since it is largely suspended over the acrylic by the texture, except where you touch it, you get less attenuation of your refracted IR light, resulting in brighter blobs.

Fabric based solutions have a smaller following here, and less dramatic of a proven success rate, but are without question the easiest to implement if an appropriate material can be sourced. Basically they involve lining the edges of your acrylic with two sided tape, and stretching the fabric over it, then repeating the process to attach your display surface. Currently the hunt is on to find a compliant surface overlay that works as both a projection surface as well as a touch surface. Users have been experimenting with various rubber materials like silicone. Compliant surfaces, while not a baseline requirement for FTIR, present the following advantages:

- Protects the expensive acrylic from scratches
- Blocks a lot of light pollution.
- Provides consistent results (the effectiveness of the bare acrylic touch seems to be down to how sweaty/greasy yours hands are).
- Zero visual disparity between the touch surface and the projection surface
- Pressure sensitive
- Seems to react better for dragging movements (at least in my experience)
- Brighter blobs to track, as there is no longer a diffuser between the IR blob light and the camera.

Developing a compliant surface for a LCD FTIR setup is difficult, as the surface must be absolutely clear and distortion-free, so as to not obscure the LCD image. This difficulty is not present in projection-based FTIR setups, as the image is projected onto the compliant surface itself. To date, no one has successfully built a LCD FTIR setup with a 100% clear compliant surface. However, several individuals have had success with LCD FTIR setups, with no compliant surface whatsoever. It appears that the need for a compliant surface largely depends on how strong blobs are without one, and specific setups.

1.4 Diffused Illumination (DI)

Diffused Illumination (DI) comes in two main forms: Front Diffused Illumination and Rear Diffused Illumination. Both techniques rely on the same basic principles - the contrast between the silent image and the finger that touches the surface.

1.4.1 Front Diffused Illumination

Visible light (often from the ambient surroundings) is shined at the screen from above the touch surface. A diffuser is placed on top or on bottom of the touch surface. When an object touches the surface, a shadow is created in the position of the object. The camera senses this shadow.

1.4.2 Rear Diffused Illumination

Depending on size and configuration of the table, it can be quite challenging to achieve a uniform distribution of IR light across the surface for rear DI setups. While certain areas are lit well and hence touches are deteced easily, other areas are darker, thus requiring the user to press harder in order for a touch to be detected. The first approach for solving this problem should be to optimize the hardware setup, i.e. positioning of illuminators, changing wall materials, etc. However, if there is no improvement possible anymore on this level, a software based approach might help.

Currently, Touchlib applies any filter with the same intensity to the whole input image. It makes sense, though, to change the filter's intensity for different areas of the surface to compensate for changing light conditions. A gradient based approach (http://eis.comp.lancs.ac.uk/~dominik/cms/index.php?pid=24) can be applied where a grayscale map is used to determine on a per-pixel-base how strong the respective filter is supposed to be applied to a certain pixel. This grayscale map can be created in an additional calibration step.

Infrared light is shined at the screen from below the touch surface. A diffuser is placed on top or on bottom of the touch surface. When an object touches the surface it reflects more light than the diffuser or objects in the background; the extra light is sensed by a camera. [Fig. 3] Depending on the diffuser, this method can also detect hover and objects placed on the surface. Rear DI, as demonstrated in the figure below, requires infrared illuminators to function. While these can be bought pre-fabricated, as discussed in the Infrared Light Source section, they can also be constructed manually using individual LEDs. Unlike other setups, Rear DI also needs some sort of a diffuser material to diffuse the light, which frequently also doubles as the projection surface.

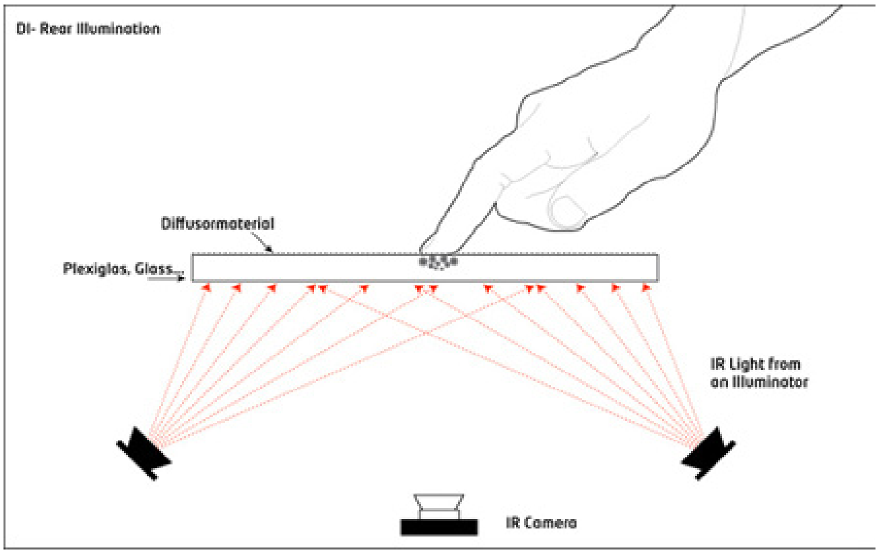


Figure 3: Rear DI schematic

1.5 Laser Light Plane (LLP)

Infrared light from a laser(s) is shined just above the surface. The laser plane of light is about 1mm thick and is positioned right above the surface, when the finger just touches it, it will hit the tip of the finger which will register as a IR blob. [Fig. 4]

Infrared lasers are an easy and usually inexpensive way to create a MT setup using the LLP method. Most setups go with 2-4 lasers, postioned on the corners of the touch surface. The laser wattage power rating (mW,W) is related to the brightness of the laser, so the more power the brighter the IR plane will be.

The common light wavelengths used are 780nm and 940nm as those are the wavelengths available on the Aixiz.com website where most people buy their laser modules. Laser modules need to have line lenses on them to create a light plane. The 120 degree line lens is most commonly used, so as to reduce the number of lasers necessary to cover the entire touch surface.

Safety when using lasers of any power is important, so exercise common sense and be mindful of where the laser beams are traveling.

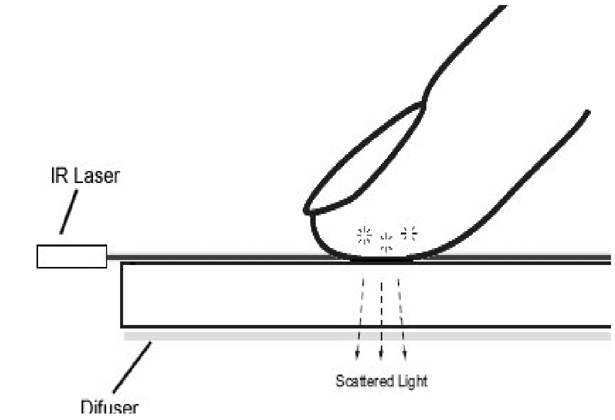


Figure 4: LLP schematic

1.5.1 Laser Safety

Infrared lasers are used to achieve the LLP effect, and these lasers carry some inherent risk. For most multi-touch setups, 5mW-25mW is sufficient. Even lasers of this grade, however, do possess some risk factors with regard to eye damage. Visible light lasers, while certainly dangerous if looked directly into, activate a blink-response, minimizing the laser's damage. Infrared lasers, however, are imperceptible to the human eye and therefore activate no blink response, allowing for greater damage. The lasers would fall under these laser safety categories [Wikipedia]:

> ***A Class 3B laser is hazardous if the eye is exposed directly, but diffuse reflections such as from paper or other matte surfaces are not harmful. Continuous lasers in the wavelength range from 315 nm to far infrared are limited to 0.5 W. For pulsed lasers between 400 and 700 nm, the limit is 30 mJ. Other limits apply to other wavelengths and to ultrashort pulsed lasers. Protective eyewear is typically required where direct viewing of a class 3B laser beam may occur. Class-3B lasers must be equipped with a key switch and a safety interlock.***

As explained in Appendix B, a line lens is used to expand a laser's 1-dimensional line into a plane. This line lens reduces the intensity of the laser, but risk is still present. It is imperative that laser safety goggles matching the wavelength of the lasers used (i.e. 780 nm) are used during setup of the LLP device. Following initial setup and alignment, caution should be exercised when using a LLP setup. No objects that could refract light oddly should be placed on the setup - for example, a wine glass's cylindrical bottom would cause laser light to be reflected all over the area haphazardly. A fence should be erected around the border of the setup to enclose stray laser beams and no lasers should ever be looked at directly. Goggles vary widely in their protection: Optical density (OD) is a measure of protection provided by a lens against various wavelengths. It's logarithmic, so a lens providing OD 5 reduces beam power by 100 times more than an OD 3 lens. There is merit to setting up a system with low-power visible red lasers for basic alignment, etc, since they are safer and optically behave similarly. When switching into IR, "detection cards" are available that allow observation of low power IR lasers by absorbing the light and re-emitting visible light - like an index card to check a visible beam, these are a useful way to check alignment of the IR laser while wearing the goggles.

To reiterate the most important safety precautions enumerated above: always use infrared safety goggles when setting up, and never shine a laser directly in your eye. If extreme caution is exercised and these rules not broken, LLP setups are quite safe. However, violation of any of these safety guidelines could result in serious injury and permanent damage to your retina.

1.6 Diffused Surface Illumination (DSI)

DSI uses a special acrylic to distribute the IR evenly across the surface. Basically use your standard FTIR setup with an LED Frame (no compliant silicone surface needed), and just switch to a special acrylic. [Fig. 5] This acrylic uses small particles that are inside the material, acting like thousands of small mirrors. When you shine IR light into the edges of this material, the light gets redirected and spread to the surface of the acrylic. The effect is similar to DI, but with even illumination, no hotspots, and same setup process as FTIR.

Evonic manufactures some different types of Endlighten. These vary in their thickness and also in the amount of particles inside the material. Available thickness ranges between 6-10mm, follwing “L”, “XL” and “XXL” for particle amount. The 6 mm (L) is too flexible for a table setup, but the 10 mm (XXL) works nicely.

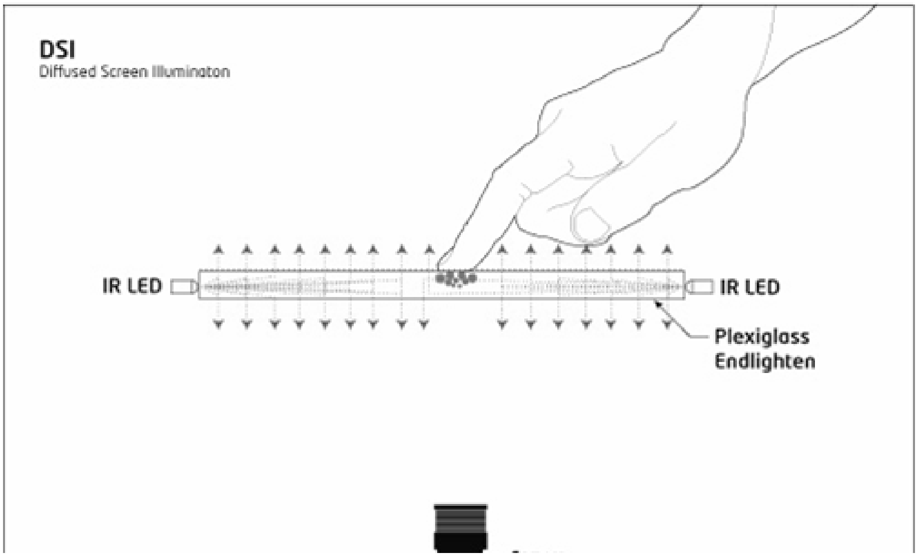


Figure 5: DSI Schematic

1.7 LED Light Plane (LED-LP)

LED-LP is setup the same way as an FTIR setup except that the thick acrylic that the infrared light travels through is removed and the light travels over the touch surface. This picture [Fig. 6] shows the layers that are common in an LED-LP setup.

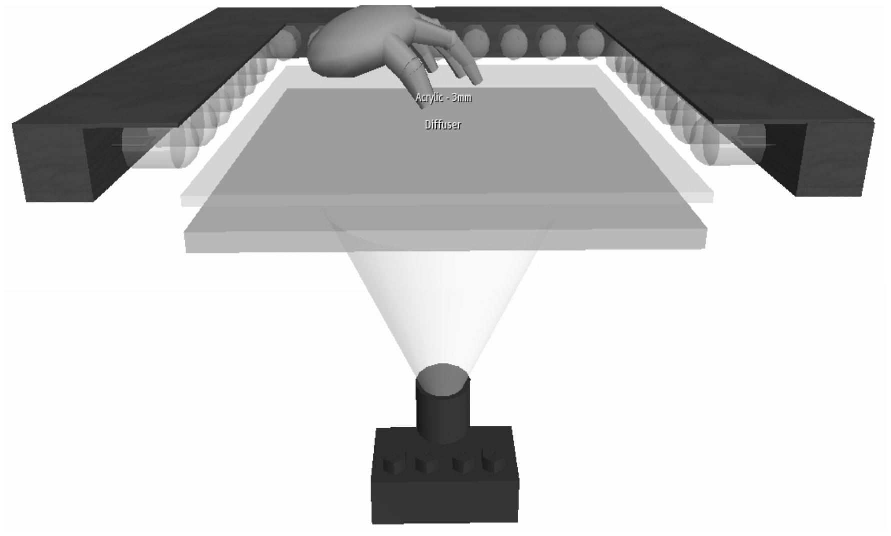


Figure 6: LED-LP 3D Schematic created in SecondLife

The infrared LEDs are placed around the touch surface; with all sides being surrounding preferred to get a more even distribution of light. Similar to LLP, LED-LP creates a plane of IR light that lays over the touch surface. Since the light coming from the LEDs is conical instead of a flat laser plane, the light will light up objects placed above the touch surface instead of touching it. This can be adjusted for by adjusting filter settings in the software (touchlib/Community Core Vision) such as the threshold levels to only pick up objects that are lit up when they are very close to the touch surface. This is a problem for people starting with this type of setup and takes some patience. It is also recommended that a bezel (as can be seen in the picture above) is put over the LEDs to shield the light into more of a plane.

LED-LP is usually only recommended when working with an LCD screen as there are better methods such as Rear DI when using a projector that usually don't work with an LCD screen. Like Rear DI and LLP the touch surface need not be thick like in FTIR, but only as strong as it needs to support the forces from working on the touch surface.

1.7.1 Layers used

First, a touch surface, which is a strong, durable surface that can take the pressure of user interaction that is optically clear should be used. This is usually acrylic or glass. If using in a projector setup, the image is stopped and displayed on the projection layer. If using in an LCD setup, the diffuser is placed below the LCD screen to evenly distribute the light from the LCD backlight.

The source of infrared light for an LED-LP setup comes from infrared LEDs that are placed around at least 2 sides of the acrylic right above the touch surface. Typically the more sides surrounded, the better the setup will be in IR prevalent lighting conditions. Refer to the LED section for more information on IR LEDs.

A computer webcam is placed on the opposite site of the touch surface so that is can see the blobs. See the camera section for more information on cameras that are commonly used.

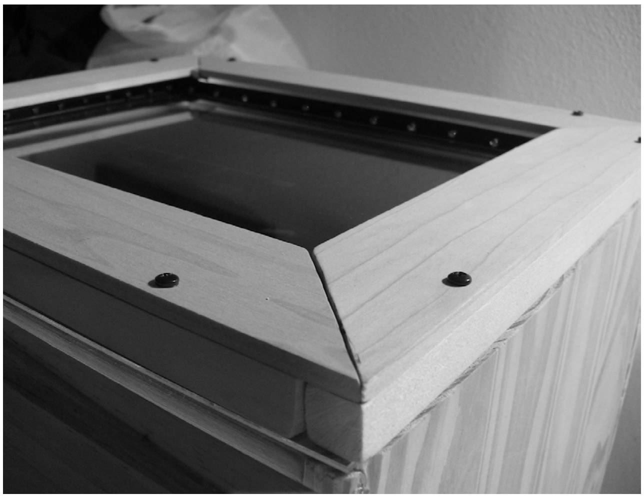


Figure 7: Picture of LED-LP setup

1.8 Technique comparison

A frequent question is "What is the best technique?" Unfortunately, there is no simple answer. Each technique has its advantages and disadvantages. The answer is less about what is best and more about what is best for you, which only you can answer.

FTIR

Advantages	Disadvantages
• An enclosed box is not required • Blobs have strong contrast • Allows for varying blob pressure • With a compliant surface, it can be used with something as small as a pen tip	• Setup calls for some type of LED frame (soldering required) • Requires a compliant surface (silicone rubber) for proper use - no glass surface • Cannot recognize objects or fiducial markers

Rear DI

Advantages	Disadvantages
• No need for a compliant surface, just an diffuser/projection surface on top/bottom • Can use any transparent material like glass (not just acrylic) • No LED frame required • No soldering (you can buy the IR-Illuminators ready to go) • Can track objects, fingers, fiducials, hovering	• Difficult to get even illumination • Blobs have lower contrast (harder to pick up by software) • Greater chance of 'false blobs' • Enclosed box is required

Front DI

Advantages	Disadvantages
• No need for a compliant surface, just an diffuser/projection surface on top/bottom • Can use any transparent material like glass (not just acrylic) • No LED frame required • No soldering (you can buy the IR-Illuminators ready to go) • Can track fingers and hovering • An enclosed box is not required • Simplest setup	• Cannot track objects and fiducials • Difficult to get even illumination • Greater chance of 'false blobs' • Not as reliable (relies heavily on ambient lighting environment)

LLP

Advantages	Disadvantages
• No compliant surface (silicone) • Can use any transparent material like glass (not just acrylic) • No LED frame required • An enclosed box is not required • Simplest setup • Could be slightly cheaper than other techniques	• Cannot track traditional objects and fiducials • Not truly pressure sensitive (since light intensity doesn't change with pressure). • Can cause occlusion if only using 1 or 2 lasers where light hitting one finger blocks another finger from receiving light.

DSI

Advantages	Disadvantages
• No compliant surface (silicone) • Can easily switch back and forth between DI (DSI) and FTIR • Can detect objects, hovering, and fiducials • Is pressure sensitive • No hotspots • Even finger/object illumination throughout the surface	• Endlighten Acrylic costs more than regular acrylic (but the some of the cost can be made up since no IR illuminators are needed) • Blobs have lower contrast (harder to pick up by software) than FTIR and LLP • Possible size restrictions due to plexiglass's softness

LED-LP

Advantages	Disadvantages
• No compliant surface (silicone) • Can use any transparent material like glass (not just acrylic) • No LED frame required • An enclosed box is not required • Could be slightly cheaper than other techniques	• Hovering might be detected • Does not track objects or fiducials • LED frame (soldering) required • Only narrow-beam LEDs can be used - no ribbons.

1.9 Display Techniques

1.9.1 Projectors

The use of a projector is one of the methods used to display visual feedback on the table. Any video projection device (LCDs, OLEDs, etc.) should work but projectors tend to be the most versatile and popular in terms of image size.

There are two main display types of projectors: DLP and LCD.

- LCD (Liquid Crystal Displays) are made up of a grid of dots that go on and off as needed. These use the same technology that is in your flat panel monitor or laptop screen. These displays are very sharp and have very strong color. LCD projectors have a screen door effect and the color tends to fade after a few years.

- DLP (Digital Light Processing) is a technology that works by the use of thousands of tiny mirrors moving back and forth. The color is then created by a spinning color wheel. This type of projector has a very good contrast ratio and is very small in physical size. DLP may have a slower response rate.

One thing to consider with projects is brightness. It's measured in ANSI lumens. The higher the number the brighter the projector and the brighter the setting can be. In a home theater setting you alway want a brighter screen but in a table setup it differs.

With the projector close to the screen and the screen size being smaller the projection can be too bright and give you a hotspot in the screen. This hotspot effect can be dazzling after looking at the screen for several minutes.

One of the biggest limiting points of a projector is the throw distance. This is the distance that is needed between the lens of the projector and the screen to get the right image size. For example in order to have a screen size of 34in, then you may need to have a distance of 2 feet between you projector and the screen. Check to see if your projection image can be adjusted to fit the size of the screen you are using.

In case you want to make your multi touch display into a boxed solution, a mirror can assist in redirecting the projected image. This provides the necessary throw length between the projector and screen while allowing a more compact configuration.

The aspect ratio is a measure of image width over image height for the projected image. for example if you have a standard television then your aspect ratio is 4:3. If you have a wide screen tv then you have a 16:9 ratio. In most cases a 4:3 ratio is preferred but it depends on the use of the interface.

1.9.2 LCD Monitors

While projection-based displays tend to be the most versatile in terms of setup size, LCD displays are also an option for providing visual feedback on multi-touch setups. All LCD displays are inherently transparent – the LCD matrix itself has no opacity. If all of theplastic casing of the display is removed and IR-blocking diffuser layers are discarded, this LCD matrix remains, attached to its circuit boards, controller, and power supply (PSU). When mounted in a multi-touch setup, this LCD matrix allows infrared light to pass through it, and at the same time, display an image rivaling that of an unmodified LCD monitor or projector.

Naturally, in order for an LCD monitor to produce an image when disassembled, it must be connected to its circuit boards. These circuit boards are attached to the matrix via FFC (flat flex cable) ribbon that is of a certain length. A LCD monitor is unusable for a multi-touch setup if these FFC ribbons are too short to extend all components of the LCD monitor far enough away from the matrix itself so as to not inhibit transparency of infrared light through it. Databases of FFC-compliant LCD monitors exist on websites such as LumenLabs, up to 19". LCD monitors that are not FFC compliant are still potentially usable for multi-touch setups, if the FFC ribbons are extended manually. FFC extensions can be found in electronics parts stores, and can be soldered onto existing cable to make it the necessary length.

Chapter 2

Software & Applications

In this chapter:

2.1: introduction to Software Programming

Programming for multi-touch input is much like any other form of coding, however there are certain protocols, methods, and standards in the multi-touch world of programming. Through the work of NUI Group and other organizations, frameworks have been developed for several languages, such as ActionScript 3, Python, C, C++, C#, and Java. Multi-touch programming is two-fold: reading and translating the "blob" input from the camera or other input device, and relaying this information through pre-defined protocols to frameworks which allow this raw blob data to be assembled into gestures that high-level language can then use to interact with an application. TUIO (Tangible User Interface Protocol) has become the industry standard for tracking blob data, and the following chapters discuss both aspects of multi-touch software: touch tracking, as well as the applications operating off of the tracking frameworks.

2.2. Tracking

Object tracking has been a fundamental research element in the field of Computer Vision. The task of tracking consists of reliably being able to re-identify a given object for a series of video frames containing that object (estimating the states of physical objects in time from a series of unreliable observations). In general this is a very difficult problem since the object needs to first be detected (quite often in clutter, with occlusion, or under varying lighting conditions) in all the frames and then the data must be associated somehow between frames in order to identify a recognized object.

Countless approaches have been presented as solutions to the problem. The most common model for the tracking problem is the generative model, which is the basis of popular solutions such as the Kalman and particle filters.

In most systems, a robust background-subtraction algorithm needs to first preprocess each frame. This assures that static or background objects in the images are taken out of consideration. Since illumination changes frequent videos, adaptive models, such as Gaussian mixture models, of background have been developed to intelligently adapt to non-uniform, dynamic backgrounds.

Once the background has been subtracted out, all that remains are the foreground objects. These objects are often identified with their centroids, and it is these points that are tracked from frame to frame. Given an extracted centroid, the tracking algorithm predicts where that point should be located in the next frame.

For example, to illustrate tracking, a fairly simple model built on a Kalman filter might be a linear constant velocity model. A Kalman filter is governed by two dynamical equations, the ‘state’ equation and the ‘observation’ equation. In this example, the set of equations would respectively be:

$$x_k = x_{k-1} + Vu_k + w_k$$

$$y_k = x_k + n_k$$

The state equation describes the propagation of the state variable, ‘xk,’ with process noise, ‘ k’ (often drawn from a zero-mean Normal distribution) If we say ‘xk-1’ represents the true position of the object at time ‘k-1,’ then the model predicts the position of the object at time ‘k’ as a linear combination of its position at time ‘k-1,’ a velocity term based on constant velocity, ‘Vuk,’ and the noise term, ‘ k.’ Now since, according to the model, we don’t have direct observations of the precise object positions, the ‘xk’s,’ we need the observation equation. The observation equation describes the actual observed variable, 'yk,' which in this case would simply define as a noisy observation of the true position, 'xk'. The measurement noise, ' k,' is also assumed to be zero-mean Gaussian white noise. Using these two equations, the Kalman filter recursively iterates a predicted object position given information about its previous state.

For each state prediction, data association is performed to connect tracking predictions with object observations. Tracks that have no nearby observation are considered to have died and observations that have no nearby tracks are considered to be a new object to be tracked.

2.2.1 Tracking for Multi-Touch

Tracking is very important to multi-touch technology. It is what enables multiple fingers to perform various actions without interrupting each other. We are also able to identify gestures because the trajectory of each finger can be traced over time, impossible without tracking.

Thankfully, today's multi-touch hardware greatly simplifies the task of tracking an object, so even the simplest Kalman filter in actuality becomes unnecessary. In fact much of the performance bottleneck of tracking systems tends to come from generating and maintaining a model of the background. Computational costs of these systems heavily constrict CPU usage unless clever tricks are employed. However, with the current infrared (IR) approaches in multi-touch hardware (e.g., FTIR or DI), an adaptive background model turns out to be overkill. Due to the fact that the captured images filter out (non-infrared) light, much of the background is removed by the hardware. Given these IR images, it is often sufficient to simply capture a single static background image to remove nearly all of the ambient light. This background image is then subtracted from all subsequent frames, the resultant frames have a threshold applied to them, and we are left with images containing 'blobs' of foreground objects (fingers or surface widgets we wish to track).

Furthermore, given the domain, the tracking problem is even simpler. Unlike general tracking solutions, we know that from one frame of video to another (~33ms for standard refresh rate of 30Hz), a human finger will travel only a limited distance. In light of this predictability, we can avoid modeling object dynamics and merely perform data association by looking for the closest matching object between two frames with a k-Nearest Neighbors (k-NN) approach. Normal NN exhaustively compares data points from one set to another to form correspondences between the closest points, often measured with Euclidian distance. With k-NN, a data point is compared with the set of 'k' points closest to it and they vote for the label assigned to the point. Using this approach, we are able to reliably track blobs identified in the images for higher-level implementations. Heuristics often need to be employed to deal with situations where ambiguity occurs, for example, when one object occludes another.

Both the Touchlib and CCV frameworks contain examples of such trackers. Using OpenCV, an open-source Computer Vision library that enables straightforward manipulation of images and videos, the frameworks track detected blobs in real-time with high confidence. The information pertaining to the tracked blobs (position, ID, area, etc.) can then be transmitted as events that identify when a new blob has been detected, when a blob 'dies,' and when a blob has moved. Application developers then utilize this information by creating outside applications that listen for these blob events and act upon them.

2.3: Gesture Recognition

The Challenge of NUI : "Simple is hard. Easy is harder. Invisible is hardest." - Jean-Louis Gassée

2.3.1 GUIs to Gesture

The future of Human-Computer-Interaction is the Natural User Interface which is now blurring its boundary with the present. With the advancement in the development of cheap yet reliable multi-touch hardware, it wouldn't be long when we can see multi-touch screens not only in highly erudite labs but also in study rooms to drawing rooms and maybe kitchens too.

The mouse and the GUI interface has been one of the main reasons for the huge penetration of computers in the society. However the interaction technique is indirect and recognition based. The Natural User Interface with multi-touch screens is intuitive, contextual and evocative. The shift from GUI to Gesture based interface will further make computers an integral but unobtrusive part of our lifestyle.

In its broadest sense, "the notion of gesture is to embrace all kinds of instances where an individual engages in movements whose communicative intent is paramount, manifest, and openly acknowledged" [3] Communication through gestures has been one of the oldest form of interaction in human civilization owing to various psychological reasons which, however, is beyond the scope of present discussion. The GUI systems leverages previous experience and familiarity with the application, whereas the NUI interface leverages the human assumptions and its logical conclusions to present an intuitive and contextual interface based on gestures. Thus a gesture based interface is a perfect candidate for social and collaborative tasks as well for applications involving an artistic touch. The interface is physical, more visible and with direct manipulation.

However, only preliminary gestures are being used today in stand-alone applications on the multi-touch hardware which gives a lot of scope for its critics. Multi-touch interface requires a new approach rather than re-implementing the GUI and WIMP methods with it. The form of the gestures determines whether the type of interaction is actually multi-touch or single-touch multi-user. We will discuss the kind of new gesture widgets required, development of gesture recognition modules and the supporting framework to fully leverage the utility of multi-touch hardware and develop customizable, easy to use complex multi-touch applications.

2.3.2 Gesture Widgets

Multi-touch based NUI setups provide a strong motivation and platform for a gestural interface as it is object based (as opposed to WIMP) and hence remove the abstractness between the real world and the application. The goal of the interface should be to realize a direct manipulation, higher immersion interface but with tolerance to the lower accuracy implied with such an interface. The popular gestures for scaling, rotating and translating images with two fingers, commonly referred as manipulation gestures, are good examples of natural gestures. [Fig. 1]

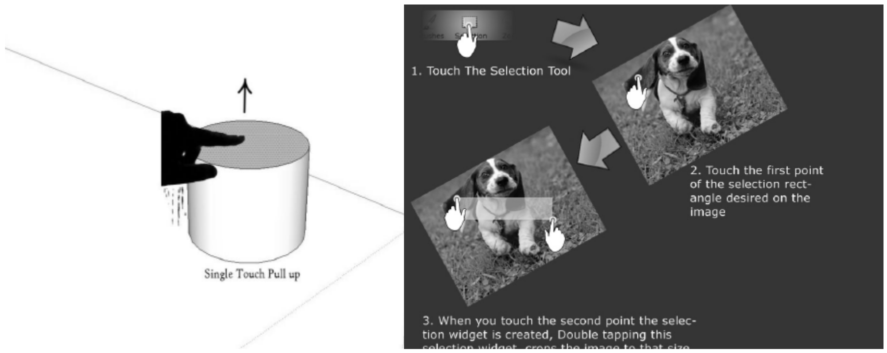

Single Touch Pull up
1. Touch The Selection Tool
2. Touch the first point of the selection rect-angle desired on the image
3. When you touch the second point the selec-tion widget is created, Double tapping this

Figure 1: Examples of gestures in multi-touch applications

New types of gesture-widgets [Fig. 2] are required to be build to fully implement the concept of direct manipulation with the objects (everything is an object in NUI with which the user can interact) and a evocative and contextual environment and not just trying to emulate mouse-clicks with a gesture. Gesture widgets should be designed with creative thinking, proper user feedback keeping in mind the context of the applica-

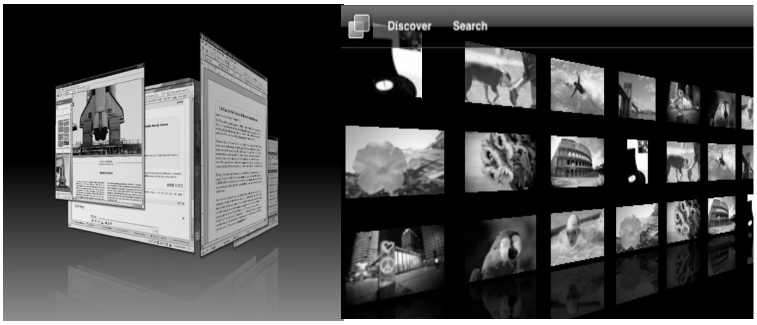


Figure 2: Gestural widget examples

tion and the underlying environment. These gesture widgets can then be extended by the application developer to design complex applications. They should also support customizable user defined gestures.

2.3.3 Gesture Recognition

The primary goal of gesture recognition research is to create a system which can identify specific human gestures and use them to convey information or for device control. In order to ensure accurate gesture recognition and an intuitive interface a number of constraints are applied to the model. The multi-touch setup should provide the capability for more than one user to interact with it working on independent (but maybe collaborative) applications and multiple such setups working in tandem.

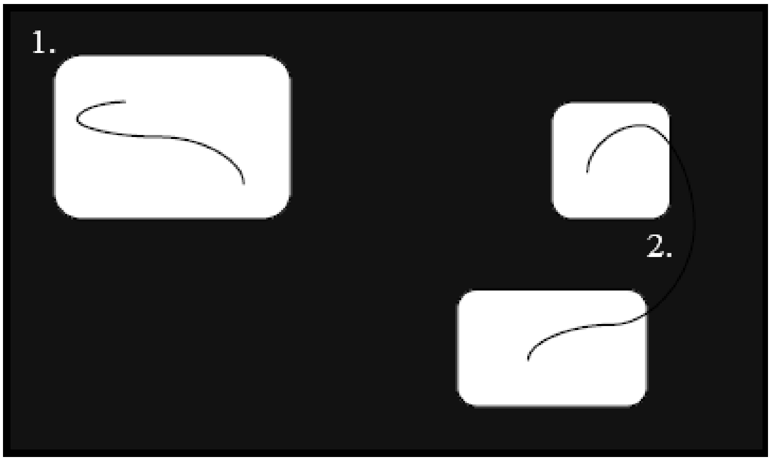


Figure 3: Gesture examples - (1) interpreted as 1 gesture, (2) as 3.

Gestures are defined by the starting point within the boundary of one context, end point and the dynamic motion between the start and end points. With multi-touch input it should also be able to recognize meaning of combination of gestures separated in space or time. The gesture recognition procedure can be categorized in three sequential process:

- Detection of Intention: Gestures should only be interpreted when they are made within the application window. With the implementation of support for multi-touch interface in X Input Extension Protocol and the upcoming

XI2, it will be the job of the X server to relay the touch event sequence to the correct application.

- Gesture Segmentation: The same set of gestures in the same application can map to several meanings depending on the context of the touch events. Thus the touch events should be again patterned into parts depending on the object of intention. These patterned data will be sent to the gesture recognition module.
- Gesture Classification: The gesture recognition module will work upon the patterned data to map it to the correct command. There are various techniques for gesture recognition which can be used alone or in combination like Hidden Markov Models, Artificial Neural Networks, Finite State Machine etc.

One of the most important techniques is Hidden Markov Models (HMM). HMM is a statistical model where the distributed initial points work well and the output distributions are automatically learned by the training process. It is capable of modeling spatio-temporal time series where the same gesture can differ in shape and duration. [4]

An artificial neural network (ANN), often just called a "neural network" (NN), is a computational model based on biological neural networks [Fig. 4]. It is very flexible in changing environments. A novel approach can be based on the use of a set of hybrid recognizers using both HMM and partially recurrent ANN. [2]

The gesture recognition module should also provide functionality for online and

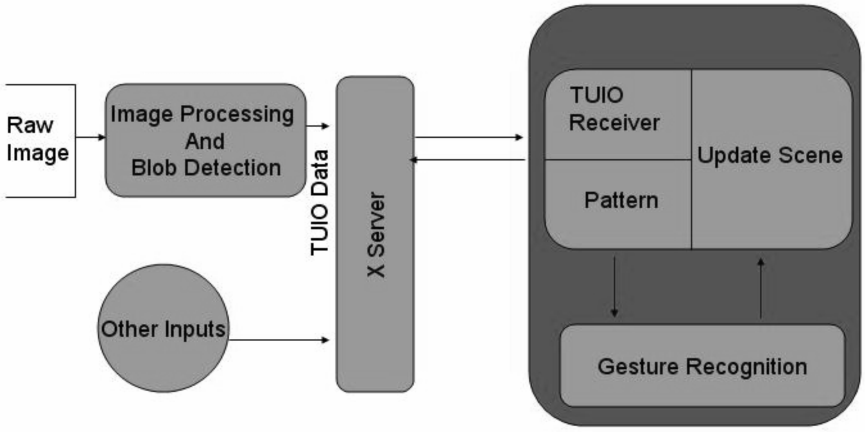


Figure 4: Blob detection to gesture recognition framework outline

offline recognition as well as shot and progressive gestures. A model based on the present discussion can be pictorially shown as:

The separation of the gesture recognition module from the main client architecture, which itself can be a MVC architecture, helps in using a gesture recognition module as per requirement eg say later on speech or other inputs are available to the gesture recognition module then it can map it to appropriate command for the interface controller to update the view and the model.

2.3.4 Development Frameworks

A number of frameworks have been released and are being developed to help in the development of multi-touch applications providing an interface for the management of touch events in an object-oriented fashion. However the level of abstraction is still till the device input management, in the form of touch events being sent through TUIO protocol irrespective of the underlying hardware. The gesture recognition task which will realize the true potential of multi-touch surfaces, is still the job of the client. Often, however, some basic gestures are already included, in particular those for natural manipulation (e.g. of photos), but in general these frameworks aren't focused on gestural interfaces. They rather tend to port the GUI and WIMP canons to a multitouch environment.[1]

Gesture and gesture recognition modules have currently gained a lot of momentum with the coming up of the NUI interface. Some of the important frameworks are:

Sparsh-UI - (http://code.google.com/p/sparsh-ui/)

Sparsh-UI [30], published under LGPL license, seems to be the first actual multitouch gesture recognition framework. It can be connected to a a variety of hardware devices and supports different operating systems, programming languages and UI frameworks. Touch messages are retrieved from the connected devices and then processed for gesture recognition. Every visual component of the client interface can be associated to a specific set of gestures that will be attempted to be recognized. New gestures and recognition algorithms can be added to the default set included in the package.

Gesture Definition Markup Language - (http://nuigroup.com/wiki/Getsure_Recognition/)

GDML is a proposed XML dialect that describes how events on the input surface are built up to create distinct gestures. By using an XML dialect to describe gestures, will allow individual applications to specify their range of supported gestures to the Gesture Engine. Custom gestures can be supported. This project envisages

1. Describing a standard library (Gesturelib) of gestures suitable for the majority of applications

2. A library of code that supports the defined gestures, and generates events to the application layer.

Grafiti

Grafiti is a C# framework built on top of the Tuio client that manages multi-touch interactions in table-top interfaces. The possible use of tangible objects is particularly contemplated. It is designed to support the use of third party modules for (specialized) gesture recognition algorithms. However a set of modules for the recognition of some basic gestures is included in this project.

NUIFrame

NUIFrame is a C++ framework based on the above discussed model (currently under development). It provides a separate gesture recognition module to which besides the touch-event sequence, contextual information regarding the view of the interface is also provided by the client application. This ensures that the same gesture on different objects can result in different operations depending on the context. It will also support custom gestures based on user specification for a particular command. The set of gesture widgets will also support automatic debugging by using pattern generation, according to the gestures supported by it.

AME Patterns Library

The AME Patterns library is a new C++ pattern recognition library, currently focused on real-time gesture recognition. It uses concept-based programming to express gesture recognition algorithms in a generic fashion. The library has recently been released under the GNU General Public License as a part of AMELiA (the Arts, Media and Engineering Library Assortment), an open source library collection. It implements both a traditional hidden Markov model for gesture recognition, as well as some reduced-parameter models that provide reduced training requirements (only 1-2 examples) and improved run-time performance while maintaining good recognition results.

There are also many challenges associated with the accuracy and usefulness of Gesture Recognition software. These can be enumerated as follows:

- Noise pattern in the gestures
- Inconsistency of people in use of their gestures.
- Finding the common gestures used in a given domain instead of mapping a difficult to remember gesture.
- Tolerance to the variability in the reproduction of the gestures
- Tolerance with respect to inaccurate pointing as compared to GUI system.
- Distinguishing intentional gestures from unintentional postural adjustments- known as the gesture saliency problem.

Gorilla arm: Humans aren't designed to hold their arms in front of their faces making small motions. After more than a very few selections, the arm begins to feel sore, cramped, and oversized -- the operator looks like a gorilla while using the touch screen and feels like one afterwards. It limits the development and use of vertically-oriented touch-screens as a mainstream input technology despite a promising start in the early 1980s. However it's not a problem for short-term usage.

Further improvement to the NUI interface can be added by augmenting the input to the gesture recognition module with the inputs from proximity and pressure sensing, voice input, facial gestures etc.

2.4. Python

Python is a dynamic object-oriented programming language that can be used for many kinds of software development. It offers strong support for integration with other languages and tools, comes with extensive standard libraries, and can be learned in a few days. Many Python programmers report substantial productivity gains and feel the language encourages the development of higher quality, more maintainable code.

2.4.1 Introduction

This section describes two great technologies: multi-touch user interfaces and the Python programming language[1]. The focus is specifically directed towards using Python for developing multi-touch software. The Python module PyMT[2] is introduced and used to demonstrate some of the paradigms and challenges involved in writing software for multi-touch input. PyMT is an open source python module for rapid prototyping and development of multi-touch applications.

In many ways both Python and multi-touch have emerged for similar reasons of making computers easier to use. Although as a programming language Python certainly requires advanced technical knowledge of computers, a major goal is to let developers write code faster and with less hassle. Python is often touted as being much easier to learn than other programming languages[3]. Many Python programmers feel strongly that the language lets them focus on problem solving and reaching their goals, rather than deal with arcane syntax and strict language properties. With it's emphasis on readability and a plethora of available modules available from the open source community, Python tries to make programming as easy and fun as multi-touch user interfaces make computers more natural and intuitive to use.

Although the availability of multi-touch interfaces is growing at an unprecedented rate, interactions and applications for multi-touch interfaces have yet to reach their full potential. Especially during this still very experimental phase of exploring multi-touch interfaces, being able to implement and test interactions or prototypes quickly is of utmost importance. Python's dynamic nature, rapid development capabilities, and plethora of available modules, make it an ideal language for prototyping and developing multi-touch interactions or applications quickly.

2.4.2 Python Multi-touch Modules and Projects

The following is a brief overview of some multi-touch related python projects and modules available:

PyMT [2]

PyMT is a python module for developing multi-touch enabled media rich OpenGL applications. It was started as a research project at the University of Iowa[10], and more recently has been maintained and grown substantially thanks to a group of very motivated and talented open source developers [2]. It runs on Linux, OSX, and Windows. PyMT will be discussed in further detail in the following subsections. PyMT is licensed under the GPL license.

touchPy [4]

Python framework for working with the TUIO[8] protocol. touchPy listens to TUIO input and implements an Observer pattern that developers can use or subclass. The Observer pattern makes touchPy platform and module agnostic. For example it does not care which framework is used to draw to the screen. There are a series of great tutorials written for touchPy by Alex Teiche [9].

PointIR [5]

PointIR is a python based multi-touch system demoed at PyCon 2008. While it certainly looks very promising, it seems to have vanished from the (searchable?) realms of the Internet. Specific license information is unknown.

libAVG [6]

According to its web page, "libavg is a high-level media development platform with a focus on interactive installations" [4]. While not strictly a multi-touch module, it has been used successfully to receive TUIO input and do blob tracking to work as a multi-touch capable system. libavg is currently available for Linux and Mac OS X. It is open source and licensed under the LGPL.

pyTUIO [7]

A python module for receiving and parsing TUIO input. pyTUIO is licensed under the MIT license.

2.4.3 PyMT

PyMT is a python module for developing multi-touch enabled media rich OpenGL applications. The main objective of PyMT is to make developing novel, and custom interfaces as easy and fast as possible. This section will introduce PyMT in more detail, discuss its architecture, and use it as an example to discuss some of the challenges involved in writing software for multi-touch user interfaces.

Every (useful) program ever written does two things: take input and provide output. Without some sort of input, a program would always produce the exact same output (e.g "Hello World!"), which would make the program rather useless. Without output, no one would ever know what the program was doing or if it was even doing anything at all. For applications on multi-touch displays, the main method of input is touch. Graphics drawn on the display are the primary output. PyMT tries to make dealing with these two forms of input and output as easy and flexible as possible. For dealing with input, PyMT wraps the TUIO protocol [8] into an event driven widget framework. For graphical output PyMT builds on OpenGL to allow for hardware accelerated graphics and allow maximum flexibility in drawing.

2.4.4 PyMT Architecture

Figure 5 displays the main architecture of PyMT. As already mentioned it uses TUIO and OpenGL for input/output. The current version of PyMT relies on pyglet[11], which is a a cross-platform OpenGL windowing and multimedia library for Python. Especially pyglet's multimedia functionality makes dealing with images, audio and video files very easy. Most media files can be loaded with a single line of python code (and drawn/played to an OpenGL context with a single line as well). This subsection briefly discusses the role of OpenGL as a rendinering engine as well as the event system and widget hierarchy at the heart of PyMT.

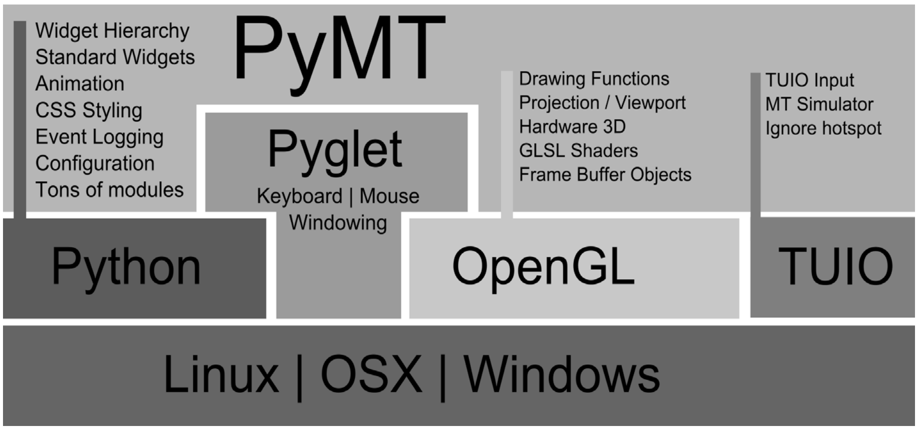


Figure 5: PyMT Architecture: PyMT builds on top of pyglet [11], which provides OpenGL windowing and multimedia loading for various file formats. It also implements a TUIO client that listens for input over the network. PyMT glues these technologies together and provides an object oriented library of widgets and a hierarchical system for layout and event propagation.

2.4.5 OpenGL

Using OpenGL as a drawing backend has both advantages and disadvantages. While it allows for ultimate performance and flexibility in terms of 2D and 3D rendering, it is also very low-level. Therefore it's learning curve can be rather steep and requires a good understanding of basic of computer graphics.

PyMT tries to counteract the need for advanced OpenGL knowledge by providing basic drawing functions that act as wrappers to OpenGL. For example PyMT includes functions such as drawCircle, drawRectangle, drawLine, drawTexturedRectangle and many others that require no knowledge of OpenGL. Using OpenGL as an underlying rendering engine however, lets more advanced users take full control over the visual output of their applications at peek performance. PyMT also provides helper functions and classes to aid advanced OpenGL programming. Creating, for example, a Frame Buffer Object or shaders from glsl source can be done in one line. PyMT also handles projection matrices and layout transformations under the hood, so that widgets can always draw and act in their local coordinate space without having to deal with outside parameters and transformations. The main idea is to do things the easy way most of the time; but if advanced techniques or custom drawing is required to provide easy access to raw OpenGL power.

Describing OpenGL itself far exceeds the scope of this document. For general information about OpenGL see [12][13], the standard reference "The Red Book"[14], or the classic Nehe OpenGL tutorials on nehe.gamedev.net [15]

2.4.6 Windows, Widgets and Events

Most Graphical User Interface (GUI) toolkits and frameworks have a concept called the Widget. A Widget is considered the building block of a common graphical user interface. It is an element that is displayed visually and can be manipulated through interactions. The Wikipedia article on Widgets for example says the following [16]:

> *In computer programming, a widget (or control) is an element of a graphical user interface (GUI) that displays an information arrangement changeable by the user, such as a window or a text box. [...] A family of common reusable widgets has evolved for holding general information based on the PARC research for the Xerox Alto User Interface. [...] Each type of widgets generally is defined as a class by object-oriented programming (OOP). Therefore, many widgets are derived from class inheritance.*
>
> *Wikipedia Article: GUI Widget[16]*

PyMT uses similar concepts as other GUI toolkits and provides an array of widgets for use in multi-touch applications as part of the framework. However, the main focus lies on letting the programmer easily implement custom widgets and experiment with novel interaction techniques, rather than providing a stock of standard widgets. This is motivated primarily by the observations and assumptions that:

- Only very few "Widgets" and Interactions have proven themselves standard within the natural user interface (NUI).
- The NUI is inherently different from the classic GUI / WIMP (Window, Icon, Menu, Pointing device).
- The NUI is very contextual. i.e. visual information and interactions should adapt to the context of user interaction.
- Real time multi-user and collaborative interactions have been largely impos-

sible on Mouse/Keyboard systems. They may require a complete rethinking of the user interface.

PyMT organizes widgets at runtime in a tree like hierarchy. At the root of the tree is an application window (usually of type MTWindow). Widgets can be added to such a window or to other widgets by use of the add_widget method which is defined as part of the base class of all widgets: MTwidget. Events, such as on_draw, on_resize, mouse and touch events are propagated along this tree to reach all of the Widgets. Programmers can take advantage of the hierarchy by defining container elements, which can handle layout or prevent unnesesary processing of events by widgets not affected by a particular event.

PyMT provdides a quite extensive list of widgets and utility objects with various functionality. For example all widgets can be styled using CSS. Instead of programatically building the widget tree using add_widget, the programmer can supply XML describing the hierarchy to be built automatically. Because listing all of the functionality included in PyMT would take too much space at this point, the readers should consult the PyMT API documentation for further information[2]

This following short example program will attempt to demonstrate some of the key ideas of PyMT. Figure 2 shows the result of running this example code in Listing 1 using a hand to give 5 input touches. The program logic can be divided into roughly four main sections. * The only way to truly evaluate novel interactions and interfaces for the NUI, is to implement them and actually experience them on a device. (PyMT also provides event logging functionality for the purposes of documenting and analyzing formal user studies)

1. Importing PyMT and initializing variables: The very first import line tells python to use PyMT, this loads all the PyMT objects and functions into the namespace. The program also creates a variable called touch_positions. This variable is a python dictionary (hashmap of key value pairs) that is used to store touch positions of individual touches.

2. Defining a new TestWidget class: The widget inherits from MTWidget and defines 4 event handlers. The on_touch_down and on_touch_up event handlers update the touch positions. The on_touch_up handler removes the touch from the touch_positions dictionary. The draw method is responsible for drawing a red circle of radius 40 for each touch at its current position. It does so by using a PyMT functions drawCirlce, and using the values stored in touch_positions.

3. Creating a window to hold the widget: An MTWindow is an application window, you can add widgets to this window using the add_widget function. An added widget will receive touch and draw events from the window and render to it.

4. Starting the application/main loop: The runTouchApp function starts PyMT's main event loop. This instatiates the window, the TUIO listener, and starts the dispatching of events.

```
from pymt import *
#a dictionary/hash map to store touch positions
touch_positions = {}
#A widget that will draw a circle for each touch
class TestWidget(MTWidget):

    #these functions handle the touch events
    def on_touch_down(self, touches, touchID, x, y):
        touch_positions[touchID] = (x,y)

    def on_touch_move(self, touches, touchID, x, y):
        touch_positions[touchID] = (x,y)

    def on_touch_up(self, touches, touchID, x, y):
        del touch_positions[touchID]

    #at each frame, the draw method draws the widget
    def draw(self):
        set_color(1,0,0)
        for position in touch_positions.values():
            drawCircle(position, radius = 40)

#create a window and add our widget to it
win = MTWindow()
widget = TestWidget()
win.add_widget(widget)

#run the program
runTouchApp()
```

Listing 1. Example source code for a PyMT application that draws a red circle at each touch position.

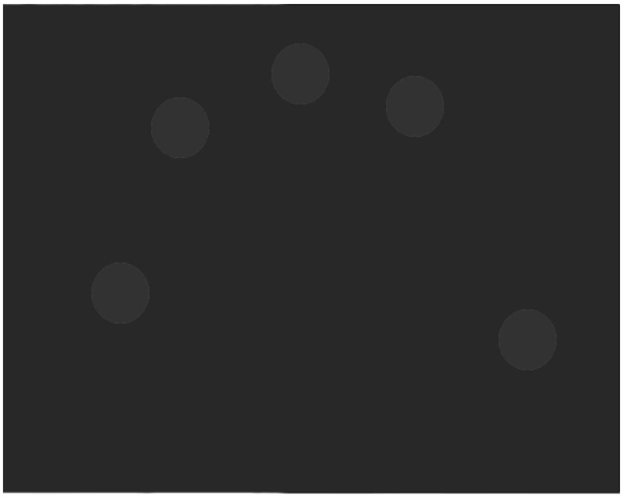


Figure 6: A screen shot of the example from the program in Listing 1. Taken with 5 concurrent touches (on a very small screen: 640x480).

2.4.7 Programming for Multi-Touch Input

Programming applications and interactions for multi-touch devices is very different from developing classic mouse based interfaces. The previous subsection discussed some reasons the NUI is inherently different from the classical WIMP based GUI. The major differences lies in having to handle multiple simultaneous touches/cursors. While being able to interpret multiple touches hugely expands the possibilities for potential interactions and interfaces, it also adds a layer complexity to the programming and computational aspect of development that is far from trivial.

The way TUIO, and most other multi-touch protocols and frameworks handle the issue of receiving event information about touch input defines three types of basic messages/events. These signal a new touch, movement of an existing touch, or the removal of a touch. The messages are always annotated with a "touch ID", so that the program and its callback function can handle the event in the context of how that specific touch and others have been presented in prior events. As seen in the example from Listing1, these events arrive in PyMT as:

- on_touch_down(touches, touchID, x, y)
- on_touch_move(touches, touchID, x, y)
- on_touch_up(touches, touchID, x, y)

Each event carries in addition to the touchID the x and y location in pixels coordinates of the event relative to the containing window. Also a dictionary with touchID's of all currently alive touches as keys is provided with the touches hashmap. Using the touches dictionary, the programmer can inspect all current touches and their locations at the time of the current event. In fact the touches dictionary holds TUIO2DCursor objects as its values, which hold additional information such as relative movement and acceleration as defined by the TUIO protocol. PyMT also provides on_object_* events for TUIO object messages (e.g. used with fiducial markers).

It quickly becomes clear that interpreting multiple simultaneous touches becomes much more complex than handling a single pointer for e.g. a mouse. The context of the user interface can change with each individual touch, and subsequent event handlers must be aware of all the interactions happening before making a decision about how to deal with a touch event.

A couple of general programming techniques have proven helpful while experimenting with PyMT. For example the concept of giving a widget ownership of a specific touch has proven useful in various cases. When using this technique other widgets ignore subsequent touch_move or touch_up events with a specific touchID. Only the widget that has taken ownership will handle these events.

Also, the use of hashmaps (key value mappings), or dictionaries as they are called in Python can aid significantly in keeping track of the various states and contexts. A simple example of this was shown in Listing 1. More useful applications of dictionaries can be achieved when multiple dictionaries are used to seperate touches into certain groups responsible for one part of the interaction or a certain widget.

Ultimately, the possibilities for new interactions and way to interpret multiple touches is limited only by the creativity and imagination of the programmer. The multitude of configurations and states that a multi-touch display can find itself in at any given moment is gigantic even before considering prior context. As the next example will show, this allows for some very natural and intuitive interactions to be programmed, but it also means the logic and algorithmic complexity becomes much bigger and harder to cope with.

2.4.8 Example: Implementing the Rotate/Scale/Move Interaction

For it's conclusion, this section discusses the implementation of a well known interaction technique. The Rotate/Scale/Move interaction [Fig. 7] is often seen in various multi-touch demos. This intuitive way to rotate, move and scale a two dimensional objects using two fingers mimics the way one can e.g. move a piece of paper on a table top. It feels so natural and intuitive because the two relative points of the object being touched remain underneath the points of contact wherever the fingers move.

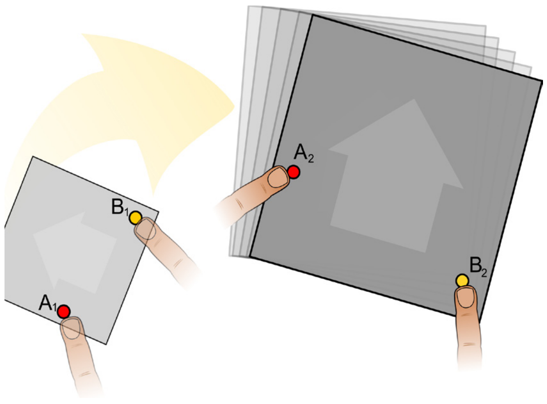


Figure 7: Rotate/Scale/Move Interaction. A user can touch two points on an object and move it another place.

In PyMT this interaction is implemented as the ScatterWidget object, to which the programmer can add any additional widgets to make them behave as part of the scatter widget. Granted, this interaction has been implemented many times. Some might argue that it has been overused or lost its novel appeal. It is included here not necessarily to showcase the particular interaction, but because it is a great example of some of the complexities that can arise in programming multi-touch interactions. Although this interaction uses only two concurrent pointers/touches and feels very natural, the computations and math involved in achieving it are somewhat more complex than most mouse interactions.

To tackle the implementation of this interaction, some basic understanding of matrix transformations is required. By using matrix multiplication, arbitrary transformations can be modeled by a single matrix (usually 4x4 in the context of computer graphics). For example a matrix representing a translation of 5 units along the x axis, can be multiplied by a matrix representing a rotation of 90 degrees around the y axis. The resulting matrix represents a transformation that does both a translation by 5 units on x and a rotation around the y axis by 90 degrees. To transform a point (its coordinates) one simply multiplies the matrix by the point. For more information on matrix transformations see [18].

2.4.9 Drawing with Matrix Transforms

Listing 2a, describes the first part needed for the rotate/scale/move interaction. Here the drawing function draws the object its working with. transform_mat is a transformation matrix. So far it only holds the identity matrix, which leaves points unchanged when multiplied. Implementing the interaction is now a question of modifying the transformation matrix appropriately based on the touch input.

```
    #a matrix that holds the transformation we are apply-
ing to our object

    self.transform_mat = IdentityMatrix()

    #a hashmap to keep track of the touchID's we have seen

    self.touches = {}

      #this function draws the object using the transfor-
mation matrix

    def draw(self):

            glPushMatrix() #save the current matrix state
```

```
        glMultMatrixf(self.transform_mat)  #apply transfor
              self.draw_object() # draw the object as usual
         glPopMatrix() #restore the current matrix state
```

Listing 2a. Example code for Rotate/Scale/Move interaction. When drawing the object, a transformation matrix is applied. The matrix is modified by following functions on touch events. For complete code of entire class see pymt/ui/scatter.py from [2].

2.4.10 Parameterizing the Problem and Computing the Transformation

Next, the problem lies in determining how to transform the transformation matrix of the object. Figure 8 depicts some of the parameters needed to transform the object. An important observation to make is that for any given event only one of the two touches can have moved. This is because every event only caries information for one touch at a time.

To compute the new transformation the following parameters are required. The distance between the two fingers before and after the event (d1 and d2). The angle R by which to rotate the object. And the point Pi around which to rotate/scale (By our observation this is one of the two original points).

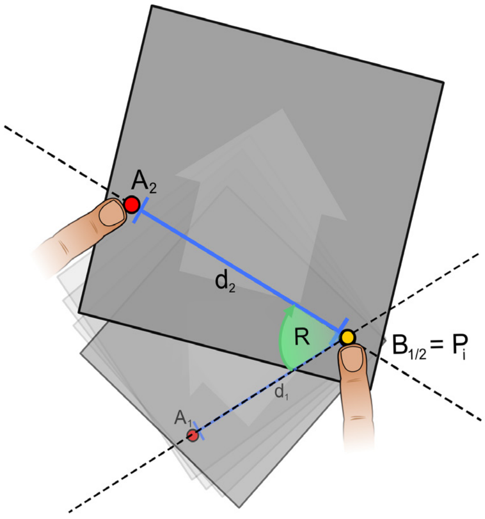


Figure 8. Rotate/Scale/Move Interaction. From two touches (A and B), various pa-

rameters have to be computed. The scale by which to enlarge/shrink the object (d2/d1), the angle by which to rotate the object (R), and the point around which to perform these transformations (Pi). An important observation is that only one of the two touch locations will change at each received event (i.e wither A=Pi or B=Pi)

Assuming the parameters are computed somehow, the appropriate changes will need to applied to the transformation matrix. Listing 2b shows PyMT / OpenGL code to do this. In OpenGL, matrices are always pre-multiplied, so that the transformation closest to the drawing call is the first one applied (although called last). Here the new transformation is computed and stored as follows:

1. Start with the old transformation stored in transform_mat
2. Translate so that the point around which to rotate is at the origin (0,0), since rotations and scales are always relative to the origin in OpenGL
3. Rotate by the computed amount of degrees around the z axis (going into/out of the screen)
4. Apply the computed scale factor
5. Move the entire thing back to the point it came from

Another noteworthy observation is that no translation (plain move) is ever applied. Using this technique, the movement of the of the object is achieved by rotating around the point that didn't change in the last event, or scaling and shrinking it in different directions subsequently. The translation happens automatically as events occur after each other. However, to do e.g. a one finger drag, which translates the object, a translation would have to be added as it is in the actual implementation of ScatterWidget in PyMT. It is left out here for keeping the example as simple as possible.

```
#this functions applies the newly computed transformations
#to the old matrix
   def apply_angle_scale_trans(self, angle, scale, point):
            glPushMatrix() #save the current matrix state
        glLoadIdentity()
         glTranslated(point.x, point.y,0)    #5. move back
to intersection
    glScaled(scale, scale,1)  #4. Scale around (0,0)
    glRotated(angle,0,0,1)  #3. Rotate around (0,0)

    glTranslated(-point.x, -point.y,0) #2. Move intersec-
tion to (0,0)

   glMultMatrixf(self.transform_mat)  #1. apply transform
as was


        #now save the modified matrix back to self.trans-
form_mat

         glGetFloatv(GL_MODELVIEW_MATRIX,self.transform_
mat)

        glPopMatrix() #restore the current matrix state
```

Listing 2b. Example code for applying transformation parameters. This function applies the transformation parameters to the transformation matrix. For complete code of entire class see pymt/ui/scatter.py from [2].

2.4.11 Interpreting Touch Locations to Compute the Parameters

Finally, the program must compute the parameters solely based on the location provided by the touch events. Listing 2c shows code to do this. For simplicity, this code assumes a function get_second_touch to get the information about the second touch involved in the interaction. A function like this can be implemented e.g. by using a dictionary to keep track of the touchID's, making sure only two touches are in it at any given point in time, and then returning the one not equal to the argument passed.

The code also assumes some basic vector functions for computing lpoint distances and angles between lines. Implementations for these can be found in PyMT's vector class [2]. Alternatively see [20] [21]. The computations basically boil down to the magnitude of a vector and the dot product between two vectors.

To angle of rotation is given by the angle between the lines of A1-B (where the current touch used to be (A1) and the second touch) and A2-B (where the touch is now and the second touch). The scale factor is easily computed by the ratio of the new distance d2 divided by the old d1. Where d1 a= |A1, B| and d1 a= |A2, B|.

```
    def rotate_zoom_move(self, touchID, x, y):
        intersect = Vector(0,0)
        rotation = 0
        scale = 1
         #we definitly have one point, so if nothing else
we drag/translate
        A1= Vector(*self.touches[touchID])
        A2= Vector(x,y)
        #get the second touch
        #(not the on with touchID of current event)
        second_touch = self.get_second_touch(touchID)
        B = Vector(*self.touches[second_touch])
              #compute scale factor (d1 and d2 are floats)
        d1= A1.distance(B)
        d2= A2.distance(B)
        scale = d1/d2
        #compute rotation angle
        old_line = A1 - B
        new_line = A2 - B
        rotation = -1.0 * old_line.angle(new_line)
        #apply to our transformation matrix
        self.apply_angle_scale_trans(rotation, scale, B)
        #save new position of the current touch
        self.touches[touchID] = Vector(x,y)
```

Listing 2c. Example code for computing parameters of Rotate/Scale/Move interaction. For complete code of entire class see pymt/ui/scatter.py from [2].

2.5 ActionScript 3 & Flash

Back to about 1996, the web was relatively young and people were looking for a tool to help make website animated and vibrant. FutureSplash was invented, mentioned as a simple way to add vector-based animation and release it on sites through Macromedia's Shockwave player. Macromedia itself soon acquired FutureSplash Animator and re-dubbed it Macromedia Flash. As the years went on, Flash evolved, its capabilities increased and spread across computers all over the world. In 2000, Macromedia added Actionscript 1.0 a way to edit and manipulate objects with a programming language rather than only using the timeline animation. In 2005, Actionscript 2.0 started to shift the idea of programming to an object oriented programming model rather than a straight forward syntax. In 2007, Adobe acquires the rights to Macromedia, including Flash which is released as Adobe Flash CS3. Adobe adds a fully revised version of the last version of Actionscript and labels this one 3.0 [1].

By now, Flash is one of the most powerful tools in a web designer's arsenal, but it no longer stops at the web. Because of new coding platforms, namely Adobe Flex and Adobe AIR, developers can use Actionscript 3.0 to create cross-platform desktop applications. And now, Flash can be used to create multi-touch applications via communication with a computer vision and multi-touch sensing application, such as Touchlib, CCV and reacTIVision.

2.5.1 Opening the Lines of Communication

Before we start developing applications in Flash, it's important to understand the flow of information from the multi-touch sensing applications. When you open the applications it's all prepared to start transmitting the blob information for any application that can read it using Tangible UI Object protocol (TUIO). However, Flash doesn't understand the blob data right away. Why is this? "TUIO is a very simple UDP-based protocol. So if you wish to create responsive applications using Adobe Flash Professional or ActionScript 3.0, you need to have a bridge that reads this UDP socket and converts it to a TCP connection." [2]

So what can make a UDP to TCP connection? Flash OSC can. [3] It makes the bridge we need between Touchlib and Flash so that you can create multi-touch Flash applications.

Inside MyFirstApp.as, paste in this code:

```
package app.demo.MyTouchApp{
      import flash.display.*;
      public class MyFirstApp extends Sprite {
      public function MyFirstApp():void {
      trace("MyFirstApp Loaded");
      }
      }
}
```

After your Flash file's properties are adjusted, there is one more piece to add to it. Find the Document Class property and type in app.demo.MyTouchApp.MyFirstApp this is the Flash file's connection to the MyFirstApp.as file.

There's only one last thing to do, right now the Flash file is empty, and the way TUIO works, there needs to be something on the stage for it to recognize that a user is touching the screen. All we need to do now is put a Shape over the stage. So use the Rectangle Tool and cover the entire stage area. And set your color, in Flash CS3, you can even chance the alpha to be fully transparent. TUIO will still be able to recognize the shape and react to it.

Now it's time to run a test movie. In the Control drop down.

This is to verify that the Flash file and the Actionscript file are in sync. So long as you don't get any errors, you should see an empty Flash movie and the output panel will pop up and read "MyFirstApp Loaded".

Now we need to add the TUIO class to the .as file.

```
...
      public function MyFirstApp():void {
      TUIO.init(this,'localhost',3000,'',true);
      }
      }
}
```

Test the movie again, you should see the white, red and green squares in the upper left corner, the touch information in the upper right, and circles under your fingers where you touch on the stage.

So, everything it ready to go. Now we need a solution to gather the touch information into an array. We could create an array in BlobLines.as, and on every touch down add that blob to the array, then on a touch up remove the blob from an array, and on a touch movement update all the information in that array. We could do that, and I've done that before, it's not a pleasant thing to write. BUT! I just so happens that TUIO.as actually contains that information in an array, OBJECT_ARRAY, and is much easier to pass through to our AS file than one may think. And since TUIO.as is gathering the information directly from FLOSC, it seems to me that it is more reliable and accurate to work with.

So how do we do that? We need to add a little something to flash/events/TUIO.as

```
public static function returnBlobs():Array{
      return OBJECT_ARRAY;
}
```

OBJECT_ARRAY is a private variable. The creator of this file intended only for this AS file to be able to change that value. Which makes sense, you don't want some rogue function changing value in the array when your trying to use it specifically for multi-touch objects. But, we don't want to change it, we only want to be able to get the value at any time. So, around line 119 or so, add in this function.

Save the file and return to MyFirstApp.as

Now we want to test that the information we want is coming through.

What has been added/changed?

addEventListener(Event.ENTER_FRAME, test_returnBlobs); is added to run a function every frame to do so, import flash.events.Event; needs to be added to the imports so this AS file knows what Event.ENTER_FRAME is. and the function is added to trace TUIO.returnBlobs().

Test the Movie.

You should see the number of touch points on the screen for each frame displayed in the document's Properties panel. This is good, one we have this information coming through, we can begin using however we need to.

```
package app.demo.MyTouchApp{
      import flash.display.Sprite;
      import flash.events.TUIO;
      import flash.events.Event;
      public class MyFirstApp extends Sprite {
          public function MyFirstApp():void {
              TUIO.init(this,'localhost',3000,'',true);
             addEventListener(Event.ENTER_FRAME, test_re-
turnBlobs);
          }
          public function test_returnBlobs(e:Event):void
{
              trace(TUIO.returnBlobs().length);
          }
      }
}
```

2.6 .NET/C#

According to Wikipedia, Microsoft .NET Framework is a software framework available with several Microsoft Windows operating systems. It includes a large library of coded solutions to prevent common programming problems and a virtual machine that manages the execution of programs written specifically for the framework. The .NET Framework is a key Microsoft offering and is intended to be used by most new applications created for the Windows platform.

2.6.1 Advantages of using .NET

The most noticable advantage of the .NET framework is that if you write an application using the framework, then that code is guaranteed to run on other machines that have the .NET framework installed. From the release of Microsoft Windows Vista, the .NET framework will be included with the operating system installation.

Another advantage is that the code runs as 'managed'. This means that it cannot crash your system, nor can it make it less stable. Further, compatibility issues are less that that of native C++ based application.

.NET also allows you to write code in a variety of languages. C# and Visual Basic are the most common but J# and C++ is also supported.

2.6.2 History of .NET and multi-touch

.NET 2.0 was never a real contender for multi-touch applications due to its lack of user-interface 'freedom'. It was designed to be a business application framework and not for rich user interfaces. Only with the coming of .NET 3, WPF and Silverlight provided enough attention for use in multi-touch applications. The XAML markup allows for the extensibility and 'freedom' that developers need to build rich, interactive interfaces that are aesthetically pleasing.

.NET 3 was not a supported platform from the beginning. Flash was the most commonly used design and development tool for many enthusiasts. The reason for that was lot of development took place using Touchlib and TUIO. This allowed for easy sending of touch information to client applications.

In 2007, Donovan Solms created the C# Touchlib Interface (CsTI). It allows touch data from Touchlib to be sent to .NET using a binary connection. CsTI is in its core a Touchlib application of sorts. It converts the touch events to actual .NET events that .NET programmers are so used to. Another common approach these days is to use the same TUIO that Flash uses to get the touch data into .NET.

Many .NET multi-touch frameworks have been created since the starting days. Using MultiTouch Vista, you can now control Windows 7 with a CCV or Touchlib based setup. The Microsoft Surface uses .NET as its application base. .NET 3, WPF and Silverlight all support 3D.

XNA, the new managed graphics API from Microsoft, is also another .NET possibility because of the better 3D support. But that has not yet been explored enough.

2.6.3 Getting started with .NET and multi-touch

It's required to decide whether it's best to use an existing framework, create your own or extend an existing one, as most of them are open-source. Below are listed the three options and how to get started on each:

1. Use an existing framework

There are quite a few .NET multi-touch frameworks available. They are also commonly named WPF multi-touch frameworks. As an example, MultiTouchVista allows multi-touch interaction with Windows 7.

2. Create your own

This option is for the experienced .NET developer. It requires to work with the raw touch data and one needs to figure out how to build an event system for the framework, as well as write algorithms to determine all the low level stuff that .NET usually handles.

There are two ways to get the raw touch data into .NET:

1. C# Touchlib Interface (CsTI) or
2. TUIO via XMLSocket

Other CsTI is a binary connection with only touchlib. On the other hand, TUIO via XMLSocket is a network connection with either Touchlib, CCV or ReacTIVision. You can find a basic implementation on the ReacTIVision website (http://reactivision.sourceforge.net/). Last but not least, the first .NET multi-touch framework in the community (http://code.google.com/p/dotnetmtf/) is now deprecated but the code on googlecode still shows some basics for a starting point. But it should only be used as a starting point. The correct way is to use IInputProvider. More on this topic can be found on MSDN.

3. Extend an existing framework

This is for the developer who finds a framework that he likes but it lacks one or two features. Using this approach you can only add what you need and spare some time when compared to writing your own. Make sure that the license used for the framework gives you a freedom to work on, either commercially or non-commercially. Also pay attention to the way it works, and whether it's still an active project or not.

4. Tools

Most .NET programmers prefer to use Microsoft's Visual Studio, which is a robust and versatile IDE for .NET development. One can also get free version labelled as "Express Editions". It is available here in web install format and offline ISO on Microsoft's web page.

2.7: Frameworks and Libraries

2.7.1 Computer Vision

BBTouch

BBTouch is an open source, OSX based tracking environment for optical multi-touch tables.

Programming Language: Cocoa (Mac)

License: GPL license

Page: http://benbritten.com/blog/bbtouch-quick-start/

Bespoke Multi-Touch Framework

The Bespoke Multi-Touch Framework is a feature-rich and extensible software framework for developing multi-touch interfaces. Licensed under the BSD open-source license, you are free to use and extend the source code to suit your purposes. The framework can be paired with any vision-based multi-touch hardware platform (e.g. FTIR, or Diffused Illumination). The package includes a number of example applications, a Windows mouse emulator, 2D Ink/Symbol recognition, 4-point calibration, and independent presentation layer (support for XNA and WinForms included), and OSC network support using unicast, multi-cast, and broadcast UDP/IP.

Programming Language: C#

License: BSD License

Page: http://www.bespokesoftware.org/multi-touch/

reacTIVision

reacTIVision is an open source, cross-platform computer vision framework for the fast and robust tracking of fiducial markers attached onto physical objects, as well as for multi-touch finger tracking. It was mainly designed as a toolkit for the rapid development of table-based tangible user interfaces (TUI) and multi-touch interactive surfaces.

Programming Language: C++

License: GPL license

Page: http://reactivision.sourceforge.net/

Community Core Vision (CCV)

Community Core Vision, CCV for short, and formerly tBeta, is a open source/ cross-platform solution for computer vision and multi-touch sensing. It takes an video input stream and outputs tracking data (e.g. coordinates and blob size) and touch events (e.g. finger down, moved and released) that are used in building multi-touch applications. CCV can interface with various web cameras and video devices as well as connect to various TUIO/OSC enabled applications and supports many multi-touch lighting techniques including: FTIR, DI, DSI, and LLP with expansion planned for the future (custom modules/filters).

Programming Language: C++

License: MPL or MIT (not defined)

Page: http://tbeta.nuigroup.com

Touché

Touché is a free, open-source tracking environment for optical multitouch tables. It has been written for MacOS X Leopard and uses many of its core technologies, such as QuickTime, Core Animation, Core Image and the Accelerate framework, but also high-quality open-source libraries such as libdc1394 and OpenCV, in order to achieve good tracking performance.

Programming Language: Cocoa (Mac)

License: LGPLv3

Page: http://gkaindl.com/software/touche http://code.google.com/p/touche/

Touchlib

Touchlib is a library for creating multi-touch interaction surfaces. It handles tracking blobs of infrared light, and sends your programs these multi-touch events, such as 'finger down', 'finger moved', and 'finger released'. It includes a configuration app and a few demos to get you started, and will interace with most types of webcams and video capture devices. It currently works only under Windows but efforts are being made to port it to other platforms.

Programming Language: C++

License: New BSD License

Page: http://nuigroup.com/touchlib/ http://code.google.com/p/touchlib/

2.7.2 Gateway Applications

FLOSC

FLOSC is an AS3 library that communicates with a "Flosc server" to enable flash apps to get OSC information

Programming Language: Java

License: MIT

Page: http://code.google.com/p/flosc/

2.7.3 Clients

Creative multi-touching

Creative Multitouching is a tool on a multi-touch environment to enhance creative projects together. Things to do are drawing, simple writing and search for photo's and video's on Flickr and Youtube and combining them together into a creative collage.

Programming Language: Actionscript 3 (Adobe Air)

Status: active

License: not specified

Page: http://www.multitouching.nl/page.asp?page=148

Grafiti

A multi-platform, extensible Gesture Recognition mAnagement Framework for Interactive Tabletop Interfaces. Built on top of the TUIO client, it supports the development of multitouch gestural interfaces, possibly including the use of tangible objects as targets.

Programming Language: C#

License: GNU General Public License (GPL) v3

Page: http://code.google.com/p/grafiti

Multi-Touch Vista

Multi-Touch Vista is a user input management layer that handles input from various devices (touchlib, multiple mice, wii remotes etc.) and normalises it against the scale and rotation of the target window. It will allow standard applications to be scaled and rotated in a multi-touch style and receive standardised input. It will also provide a framework on which to build multi-input WPF applications. It supports Windows Vista and XP.

Programming Language: C#

License: GNU General Public License (GPL) v2

Page: http://www.codeplex.com/MultiTouchVista

PyMT

PyMT is a python module for developing multi-touch enabled media rich OpenGL applications based on pyglet. Currently the aim is to allow for quick and easy interaction design and rapid prototype development. There is also a focus on logging tasks or sessions of user interaction to quantitative data and the analysis/visualization of such data.

Programming Language: Python

License: GPL v3

Page: http://pymt.txzone.net/

TouchPy

TouchPy is a bare bones light weight Python Multi-Touch framework that does not bind you to any specific GUI Toolkit. It is simple to use, and is the most versatile Python Multi-Touch framework.

Programming Language: Python

License: GPL

Page: http://touchpy.googlecode.com

2DCur

A project for triggering events from OSC / TUIO Protocol 2DCur (2D cursor) messages. Currently a Python library is in the works, as is a set of externals for the Lily visual programming environment for Firefox.

Programming Language: Python, Lily (Javascript Visual Language on Mozilla Framework)

License: GPL3

Page: http://2dcur.googlecode.com

2.7.4 Simulators

SimTouch

SimTouch is another TUIO simulator build using the Adobe Air runtime. The core benefit to using SimTouch is the transparent background allowing the application developer to have a better grasp of what he/she is 'touching'.

Programming Language: Action Script 3 (Adobe Air)

License: MIT License

Page: http://code.google.com/p/simtouch/

ReacTIVision

reacTIVision is an open source, cross-platform computer vision framework for the fast and robust tracking of fiducial markers attached onto physical objects, as well as for multi-touch finger tracking. It was mainly designed as a toolkit for the rapid development of table-based tangible user interfaces (TUI) and multi-touch interactive surfaces. This framework has been developed by Martin Kaltenbrunner and Ross Bencina at the Music Technology Group at the Universitat Pompeu Fabra in Barcelona, Spain as part of the the reacTable project, a novel electronic music instrument with a table-top multi-touch tangible user interface.

Since version 1.4, reacTIVision implements basic multi-touch finger tracking, identifying simple white blobs as finger tips on the surface. This generally works with DI (diffuse illumination) as well as with FTIR solutions. Additionally the overall robustness and speed of the object tracking has been improved significantly.

Programming Language: Java

License: GNU General Public License

Page: http://mtg.upf.es/reactable/?software

QMTSim

The aim of this project is the development of a New TUIO Simulator for fast development and debugging of Multi-touch Applications.TUIO is a versatile protocol, designed specifically to meet the requirements of table-top tangible user interfaces.In order to develop applications one has to wait till one gets the required multi-touch screen.

Programming Language: c++

License: GNU General Public License

Page: http://code.google.com/p/qmtsim

Appendix A: Abbreviations

Multi-touch: An interactive technique that allows single or multiple users to control graphical displays with more than one simultaneous finger.

Multi-point: An interactive technique that makes use of points of contact rather than movement. A multi-point kiosk with buttons would be an example.

Multi-user: A multi-touch device that accepts more than one user. Larger multi-touch devices are said to be inherently multi-user.

Multi-modal: A form of interaction using multiple modes of interfacing with a system.

Tabletop Computing: Interactive computer displays that take place in the form of tabletops.

Direct manipulation: The ability to use the body itself (hands, fingers, etc) to directly manage digital workspaces.

Blob tracking: Assigning each blob an ID (identifier). Each frame we try to determine which blob is which by comparing each with the previous frame.

Blob detection: Process of detecting regions or areas of interest in an image that are either lighter or darker than their surrounding.

Tracker: The program which takes images from a camera, puts them through several filters, and finally reports the position, size, and relative movement of blobs over some protocol

TUIO: Tangible User Interface Objects - a protocol used for communicating the position, size, and relative velocity of blobs. It is built on OSC, which is built on UDP.

Touch event: A term used to describe when a system knows that an object has touched the multi-touch device.

Gesture: A physical movement that can be sensed, and often an action assigned to it. Some common gestures are single finger panning, and two finger zoom-pinching.

Sensor: A device that measures changes in an environment.

ZUI: Zoomable User Interface - a user interface which is infinitely zoomable. In theory this would give you infinite workspace, but memory constrictions limit this.

Diffuser: Something that spreads and scatters light. A diffuser is used in various multi-touch techniques to create even lighting.

FTIR: Frustrated Total Internal Reflection - a multi-touch technique that exploits the phenomena of Total Internal Reflection. Light within a transparent channel of low refractive index will reflect internally until an object with a higher refractiveindex, such as a finger, touches or frustrates the surface thus lighting up the frustrated area.

DI: Diffused Illumination - a multi-touch technique that makes use of a diffused surface to help filter shadows (Front DI) orilluminated fingers (Rear DI) from a touch surface. Sometimes this is referred to as Direct Illumination.

LLP: Laser Light Plane - a multi-touch technique that uses a laser and line generating lens to cast a beam over a touch surface.When the beam plane is broken by an object, the area is lit up.

DSI: Diffused Surface Illumination - a multi-touch technique that uses a special acrylic Endlighten to help disperse even light supplied by edge lighting the acrylic. The effect is similar to DI.

Stereo Vision or Stereoscopic: A two camera multi-touch technique.

Zero force: Refers to the amount of force or pressure needed to trigger a touch event. In this case, 'zero' means 'little.'

Appendix B: Building an LLP setup

LLP (Laser Light Plane) multi-touch varies from other technologies mentioned in this book in the sense that instead of using LEDs it uses lasers. These lasers are used to create a plane of infrared light above the surface, instead of inside it (like FTIR), or from below it (like DI). In this sense it can be seen as similar to LED-LP. LLP is known for its easy of setup and amazing (very high contrast) blobs with very little effort.

Step 1: Materials Needed

- Clear Surface (Acrylic, Glass, Air)
- IR Goggles
- Infrared (780 to 880nm) Lasers with Line Lenses
- Projection Surface/LCD Monitor
- Projector (Not necessary if using LCD Monitor)
- Camera with IR Bandpass Filter to match lasers

As shown in the diagram, all the IR light in an LLP setup is above the surface, as close to the surface as can be. When an object obstructs this plane, the light is scattered downward and picked up by the camera. [Fig. 1]

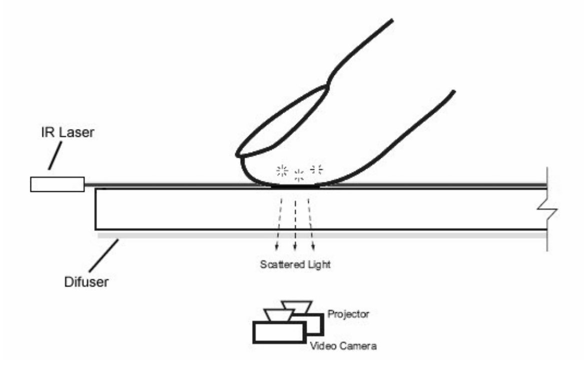


Figure 1: LLP schematic

A major pitfall with LLP is occlusions. Occlusion simply means "blockage or constriction", and it applies to LLP in the sense that with only one or two lasers, objects on the table can stop light from reaching other objects, and some objects may not be visible to the camera (See illustration below).

Step 2: Safety and infrared goggles

Before we go into the theory behind LLP and actual construction of an LLP device, some safety tips must be mentioned. When working with LLP setups you will inevitably be working with Infrared Lasers, which can be dangerous. Never point a laser at yourself or others, even with line lens attached.

Figure 2: Infrared blocking goggles example

Now that that is over with, here are two tips to make sure your table is safe during construction and use. The first is to use IR blocking goggles [Fig. 2] to match your lasers. These are essentially the opposite of a bandpass filter for your eyes, and let all light through except your laser light. This cannot be stressed enough; make sure your IR goggles and bandpass filter match your lasers, or they will be useless.

Another thing people often do is build a non-reflective "fence", or border around the touch area on their setup, just a few millimeters high. This will stop the IR light from escaping, and keep it contained within the touch area. People often use wide masking tape around the border, leaving a good inch or so sticking up.

Step 3: Lasers

Obviously if one took a standard laser and stuck it over a glass surface it would not work very well. What would result is a single-touch single-dimension slider, like a fader on an audio interface. There are two reasons for this: First, the touch closest to the laser would cause total occlusion, and any objects past that object would have no light to scatter downward (Fig. 3). As shown in the third panel, Finger one is scattering all the laser light down towards the camera, and not leaving any for Finger two. Finger one would show up as a brilliant blob, however Finger two would not show up at all. The obvious solution to this is to add another laser [Fig. 4]. While then would only allow for two touches, we can ignore this issue because when we move to 2D touching this becomes a non-issue.

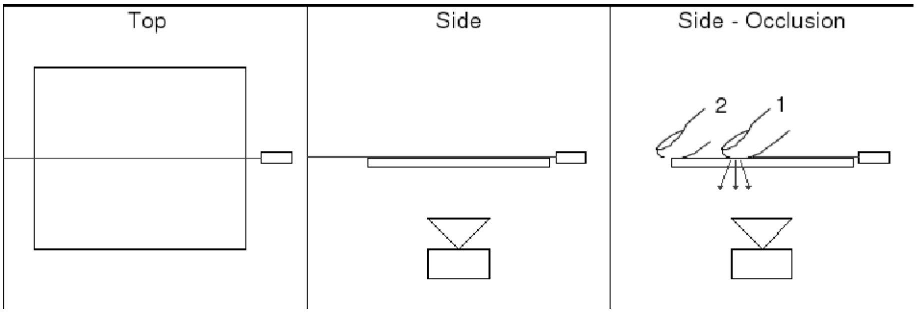


Figure 3: Demonstration of occlusion issues in LLP.

The most obvious problem to this setup is that it is single dimensional. In order to make this sense touches in the x and y plane we need a plane of light. To get this plane of light we use an optical lens called a Maddox Rod. A Maddox rod is a bunch of little semicircles lined up, and when a line of light enters it gets split up into a plane on light. From now on the Maddox Rod will be referred to as a "Line Lens", as that is the common term used around NUI Group.

The illustration above shows what a normal laser beam looks like, the type used in our example above. The next illustration shows a laser with the line lens attached. This laser is generating a plane of light, and we are only viewing one slice of it. It would be impossible for us to view the whole plane with current technology. This is the kind of laser that is used in an LLP multi-touch setup.

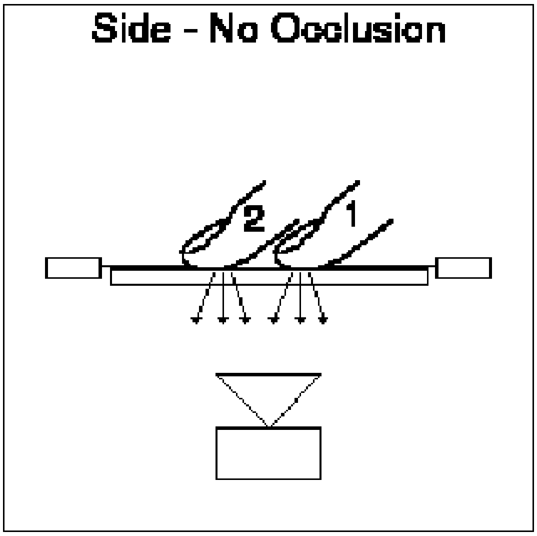


Figure 4: How occlusion can be avoided using multiple lasers

When using this there is still occlusion problems (see figure below). As seen in the illustration, the lighter object would show up to the camera, however it is blocking any light from reaching the darker object. The same obvious solution we used on the last example works here as well, use more lasers. Most people use four lasers in their LLP setup, and that provides protection from all but the weirdest and rarest occlusions. You generally do not have to worry about occlusion being too big of a deal when using two or more lasers, unless your setup is very large with more than one person using it at a given time. [Fig. 5]

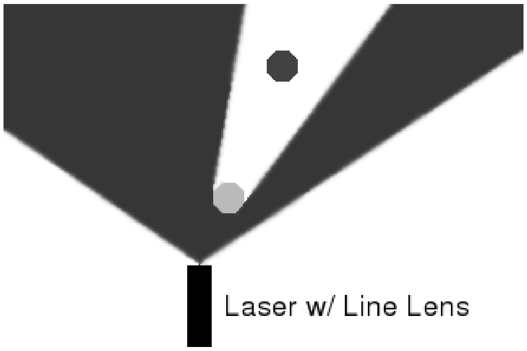


Figure 5: Line lenses and occlusion

Step 4: Selecting Your Lasers

There are many different kinds and specifications of lasers, and it is important to get the right for your setup. Like anything else, higher quality lasers cost more money. There are four key things to consider when buying lasers.

- Number of lasers being used
- Power of Lasers (in Milliwatts) being used
- Wavelength of light being emitted by lasers
- Angle of Line Lens

A common place people get their lasers is AixiZ (http://www.aixiz.com/). The first thing to think about is number of lasers. We recommend between two and four depending on the size of your setup, placed in the corners. It is possible to have occlusion problems if not using more than one laser. The ideal setup has a light plane generated from a laser in each of the four corners. When selecting lasers it is always important to have more rather then higher quality ones.

The next thing is the power of lasers. Common powers are 5mW, 10mW, and 25mW. Anything above 25 (and even 25 is most cases) is way overkill, and more dangerous then necessary. The higher power the lasers, the brighter the beam, and inherently the more dangerous that beam is. In order to maintain safety, the smallest laser that will work for your setup is desirable. A general consensus is 5mW and 10mW lasers work great as a starting point for most setups. For any screen over 20 inches (~51 cm) diagonal, 10mW should be preferred.

Less important, but still worth considering is your bandwidth. As long as it is between 780 and 900 it should work fine. Make sure you get matching IR goggles and a bandpass filter for your lasers. If your lasers are 780 nm (This is what most people use, they are very cheap), make sure to get 780 nm goggles and a 780 nm bandpass filter. Lots of people recommend going to 850 nm lasers, but there is no real reason for this. It will not work any better then a setup using 780 nm or 900 nm lasers.

Step 5: Choosing a Surface

Nothing really has to be said here, it is a complete matter of personal choice. As long as it allows IR light to pass through it should work fine.

Step 6: Getting a Power Supply

You will need a way to power these lasers. The first thing is voltage. Most lasers are either 3.3V or 5V. Your power supply must match the voltage of your lasers, or you risk damaging your lasers. Running a 5V laser at 3.3 volts probably won't hurt it, it just won't turn on, and if it does it will not be bright. Running a 3.3V laser at 5V is almost guaranteed to fry it.

The next thing that needs to be considered is current. Your power supply will have to be able to supply enough current for all the lasers, and it is recommended that your power supply can supply more current then required. Current is measured in amps or milliamps (abbreviated A or mA respectively). Assume (very roughly) 500mA for each laser. If you have four lasers, you roughly need a 2A power supply. All you need to do is get the power supply running, and then extract the black wire (for ground), and either the red (5V), or orange (3.3V). Including a kill switch in series with the power supply is a good idea, and so is a visible indicator light in parallel with the lasers. Most people use some kind of modeling clay, putty, or similar to be able to adjust the laser in the mount but then let it dry and become permanent.

Step 7: Hooking up your lasers

Wiring up lasers is very different from wiring up LEDs. Lasers are almost never wired up in series, and there is no reason to do this for a project of this kind. We recommend wiring them in parallel, similar to the diagram below [Fig. 6].

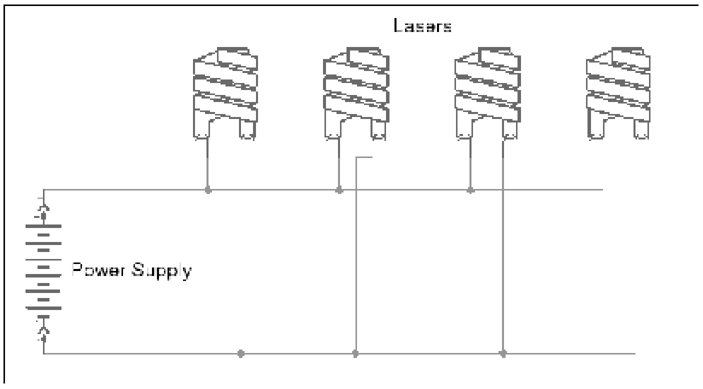


Figure 6: Parallel wiring diagram for lasers

Step 8: Attaching the line lens and focusing

The instructions in this part may differ from lasers other then those supplied by AixiZ, but the general idea should be the same. There should be three parts to your lasers, the Line Lens, Line Lens Mount, and the laser itself (Fig. 7).

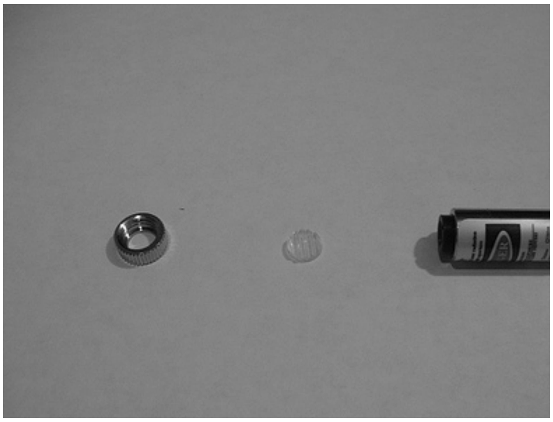


Figure 7: Parts of a laser setup (line lens, mount, laser)

Start by taking the line lens and placing it in the mount. The ridged side should be pointing towards the laser diode. If it is backwards this will become apparent when focused, the line will be very wavy and choppy. Next unscrew the black part from the laser completely, and screw it into the Line Lens mount until its snug (Fig. 8).

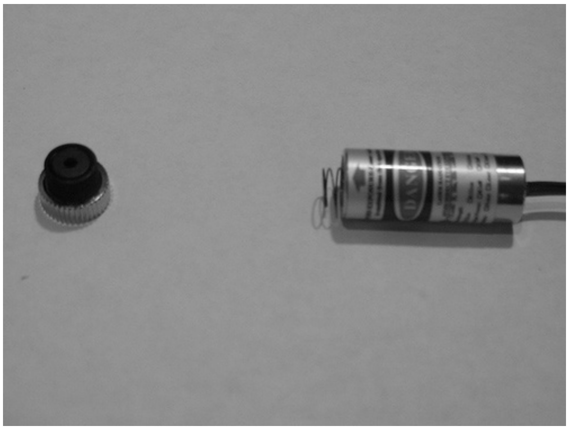


Figure 8: Laser alignment procedure – screwing in the Line Lens mount

Take the line lens mount/black piece, and screw it into the laser about half way. Put on IR goggles, grab a camera (the laser beam is very hard to see without one), power up the laser, and point it at a sheet of paper or other flat surface. Twist the line lens mount (so its screwing in and out of the laser, not the line lens mount) until the line appears very crisp. It should look something like figure 9 when done.

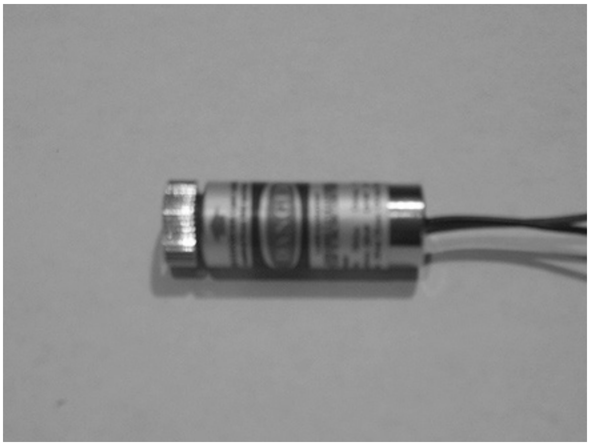


Figure 9: Assembled laser with line lens and mount

Step 9: Mounting and aligning lasers

There is no recommended or specific way that lasers should be mounted, its very setup-specific. People have done everything from duct tape to custom designed and fabricated mounts that allow for millimeter accuracy when adjusting. Mounting options very much depend on the setup, and are not within the scope of this tutorial.

Aligning the lasers is a different issue. The Line-Lens mount can be twisted about 180 degrees in either direction without loosing focus, so that makes it very easy to line up. Once they are attached, fold a piece of paper in half so there is a very flat edge protruding from your surface to compare with. Power your lasers on, and go through each of them holding a digital camera up so the beam is visible, and twist the line lens until its flat. You also want to get the laser plane as close to the touch surface as possible, so it doesn't sense a blob before the object comes in contact with the surface. This can be achieved in most cases by lifting up the back of the laser with small pieces of paper under it, or you may search for a more permanent solution.

Appendix C: Building a Rear-DI Setup

Infrared light is shined at the screen from below the touch surface. A diffuser is placed on the bottom of the touch surface so that when an object touches the surface it reflects more light than the diffuser or objects in the background. This extra light is sensed by an infrared camera and (using tracking software) is filtered into 'blobs' which are then converted into (x,y) coordinates Below is a picture illustrating the effect of diffused illumination.

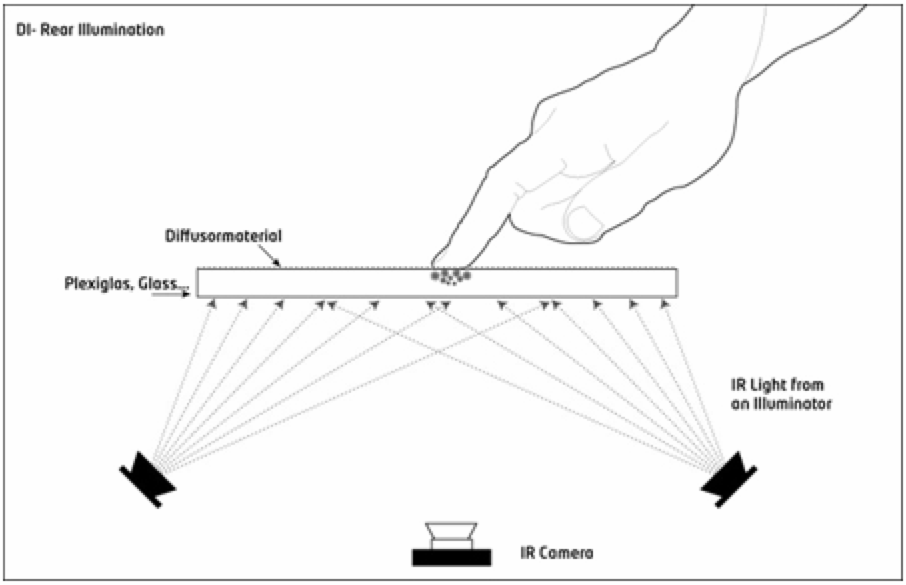


Figure 1: DI – Rear Illumination – schematic

Step 1: Building the Box

The first step in making a Diffused Illumination (DI) Multi-touch table is to build the enclosure. DI setups require a mostly closed box so that no infrared light escapes. Were light to escape, there would be no uniform infrared brightness within the box, effectively rendering the effect useless. Below is a basic 3D model created using Google SketchUp of the ORION Multi-touch Table.

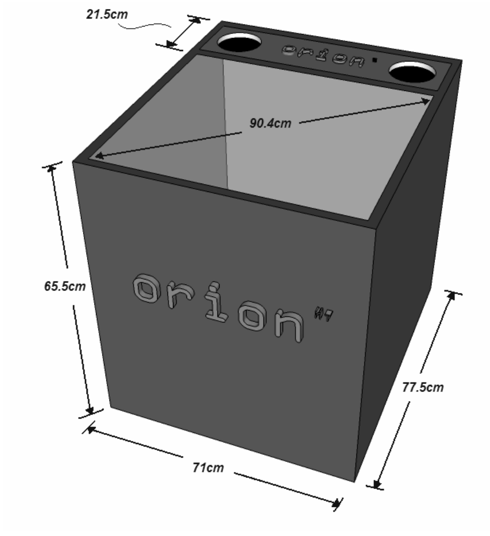


Figure 2: Sample box 3D model (above)

The box [Fig. 2] is built out of 12mm craft wood/medium-density fiberboard (MDF). There is a section on the top of the box so that a small mirror could be mounted to increase projection throw. This section is also designed to house some small speakers and possibly an external keyboard. The glass sheet pictured here [Fig. 3] was later replaced with a sheet of 4mm thick glass, 71cmx56cm frosted (sandblasted) on one side for the diffuser/projection surface.

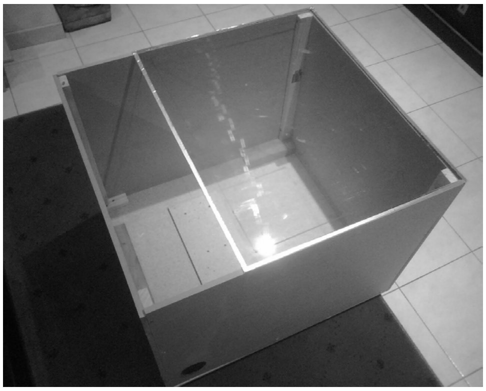


Figure 3: Semi-built box based on 3D model above.

Step 2: Projector

For the projection a SHARP DLP projector Model: PG-M15S is being used.

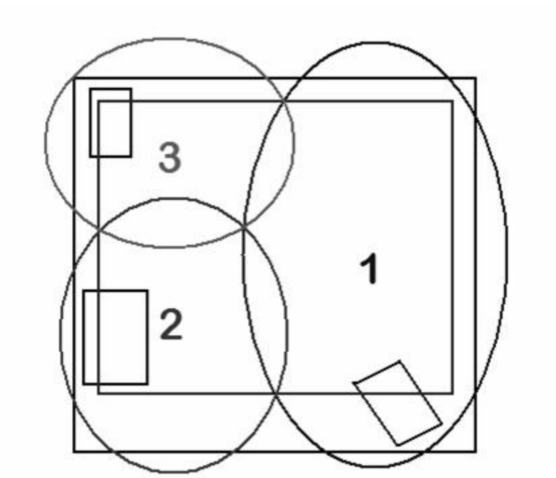
Step 3: Webcam

The camera used in this setup is an Xbox Vision 360 USB camera that can be used as a webcam on a regular PC. The resolution on the tracking software is 320x240 @ 60fps.. This camera was fairly easy to modify (removal of the IR filter), and was a cheap and easy option at the time for a reasonably good multi-touch camera.

Step 4: Infrared Illuminators

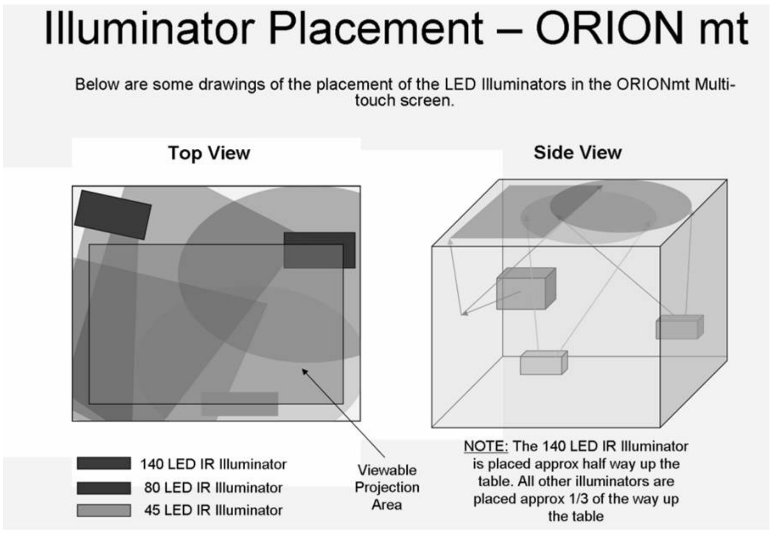
In the ORION mt there are three illuminators. A diagram illustrating the layout of the illuminators (1x 140 IR LEDs, 1x 80 IR LEDs and 1x 45 LEDs) is below.. The border represents the physical screen projection size in relation to the illuminators and the box.

The 140 IR LED Illuminator (Number 1 below) was originally bought by a friend from eBay.



Because of the narrow beam of this IR illuminator, there was a large hotspot on the glass. To get around this, the 140 IR Illuminator is angled so that the light is emitted and reflected off the side wall of the box.

The placement of the illuminators can be seen in this picture:


Illuminator Placement – ORION mt
Below are some drawings of the placement of the LED Illuminators in the ORIONmt Multi-touch screen.
Top View
Side View
140 LED IR Illuminator
80 LED IR Illuminator
45 LED IR Illuminator
Viewable Projection Area
NOTE: The 140 LED IR Illuminator is placed approx half way up the table. All other illuminators are placed approx 1/3 of the way up the table

In the illuminator placement diagram above, the overall result is that each illuminator's viewing range is overlapped to ensure that the entire viewable projection area is flooded with IR light. [Fig. 5].

Below are some specifications of the 140 IR LED illuminator:

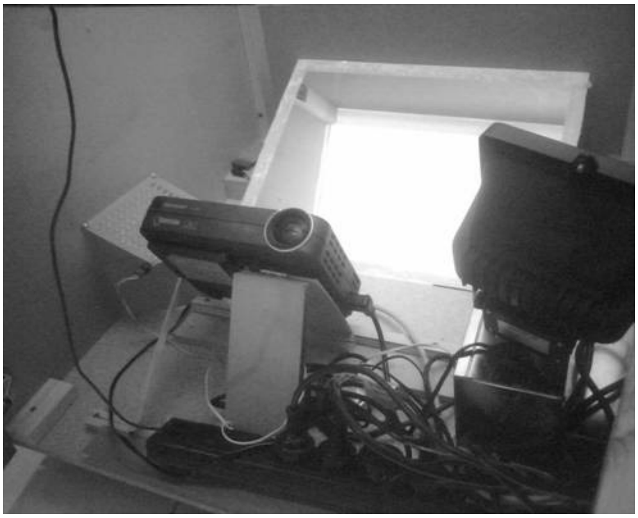


Figure 5: Image of DI setup – projector, illuminators, mirror

- Built-in light sensor (taped over)
- Illuminating range: Detecting range is 80m, viewing range is 60m (outdoor)
- Definition Consumed power: 18W
- 850nm
- Structure: All weather aluminum and reinforced glass
- Power: DC 12V 1000mA

The 80IR Led illuminator (Number 2), and the 45IR Led illuminator (Number 3), were designed and built using the LED calculator software available on the internet: http://led.linear1.org/led.wiz

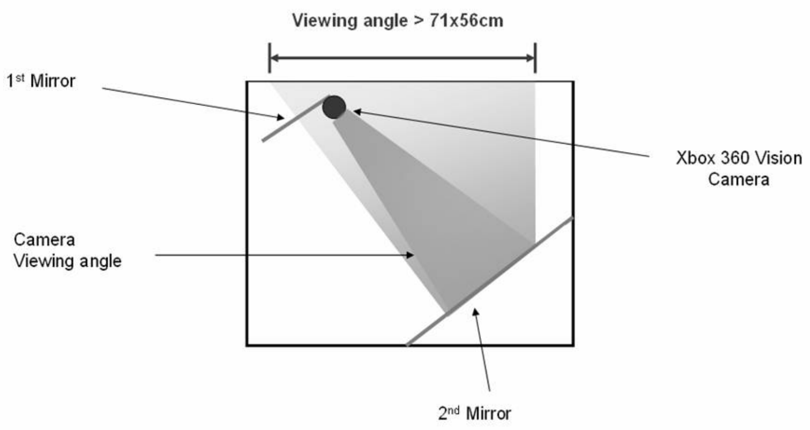
Originally only the 140IR LED illuminator was used in the setup. The downside to this was that during the day or close to a IR light source (overhead lights or the ambient sunlight), the IR generated was not enough to receive clear blobs. This meant that more IR light was required and therefore more IR illuminators

(Numbers 2 & 3) were used.

Step 5: Webcam & Projector Placement

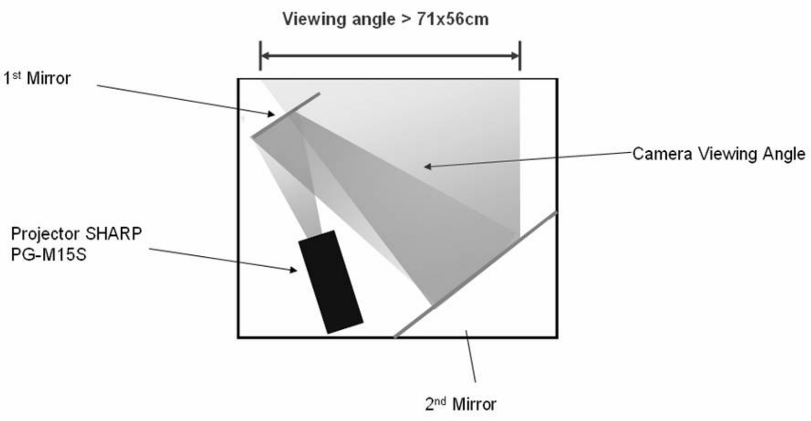
Camera Placement – ORION mt

Below I have added drawings of the ORIONmt Multi-touch screen camera and mirror placement.



Projector Placement – ORION mt

Below I have added drawings of the ORIONmt Multi-touch screen projector and mirror placement.



Camera, mirror, and projector placements are important issues to watch for to ensure that the webcam can view the entire touch surface and the projector can project on the entire surface. Originally, there was a camera angle issue in the OrionMT where the camera could not capture the entire screen when placed in the

middle of the floor. To overcome this, the cameras was moved to reflect of the 2nd mirror. That way, the entire screen can be captured (with overscan).

Step 6: Touch Surface

You need a thick enough surface to not flex when pressed (to avoid short-term hotspots becoming blobs) and some kind of surface for projection and diffusion. People have used vellum, tracing paper, fabric interfacing, and even cheap plastic tablecloths or shower curtains (in my case, what worked best) on top of the surface, in addition to frosted glass or acrylic. Before purchasing a frosted material and/or diffuser it's wise to test the surface's suitability both as a projection surface and as an IR diffuser.

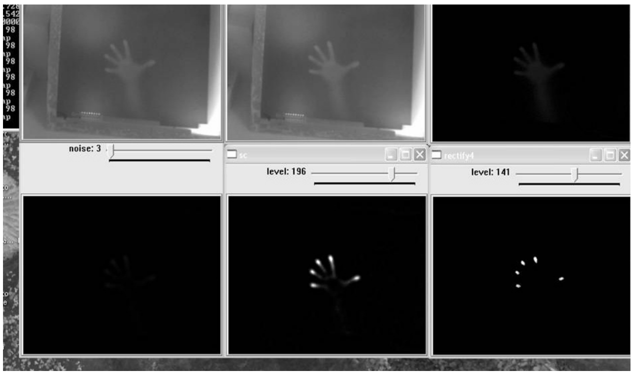


Figure 6: Touchlib (blob tracker) screenshot – configuration mode

Step 7: The Finished Product

The next step was to test the setup using the tracking software. If you look closely at the screenshot above you can just make out the 3 glowing sections where the 3 illuminators were placed from the raw camera feed (First 2 windows dsvlcapture0 & mono1). [Fig. 6].

Appendix D: Building an Interactive Floor

This chapter explains how to build an interactive floor. This floor differs mainly in the fact that multiple people can walk over and interact with the same floor space as opposed to single person interaction of regular interactive floors. A typical setup would only cover a part of the floor, ie. a lift lobby, or right at the entrance door, because covering an entire lobby - although possible - is unpractical.

Note that this should not be attempted as your first multitouch project. The reason for this is that when you build the interactive floor, you will then have the insight needed to identify problems before you go to far into the project.

Depending on the setup of the room and people installing equipment in the roof, this project will take 4-8 hours. The money spent is extremely dependent on the size of the room and optional parts required. No fixed figure can be given.

Before beginning with the project, search for possibilities to go into the roof for the setup. This will enable you to hide the setup from its users and only have a small hole where the projector projects through. Also, the light conditions in the room is one of the first things to consider. Right next to a window that has sun most of the day will not work. One can, however, control the light in the room by covering the windows with IR blocking film.

You will notice no IR illuminators are needed. The reason for that is that 95% of lobby areas will have a lighting system already installed. They already emit IR in an evenly spread way. If the lobby does not have a lighting system, you can easily add an illuminator next to the camera.

The shininess of the floor is usually not a problem. It actually helps tracking in some situations. In short - the less shiny the floor, the better the result. But if the floor is semi-shiny – or reflective – don't break your head about it. White floors are prefered as they give the best projection.

Warning: Because the setup relies on having a projector and camera right above the floor you must ensure that all parts in the roof – or hanging from the roof – is 100% secure. If not, the parts could easily drop from the roof onto a person injuring him/her or severely damaging parts.

Different Techniques:

There are basically two ways that you can build the setup. The first being the ability to go into the roof for the installation, the second not being able to. We will cover the In-Roof technique first, then the Below-Roof technique. [Fig. 1]

In-Roof Technique

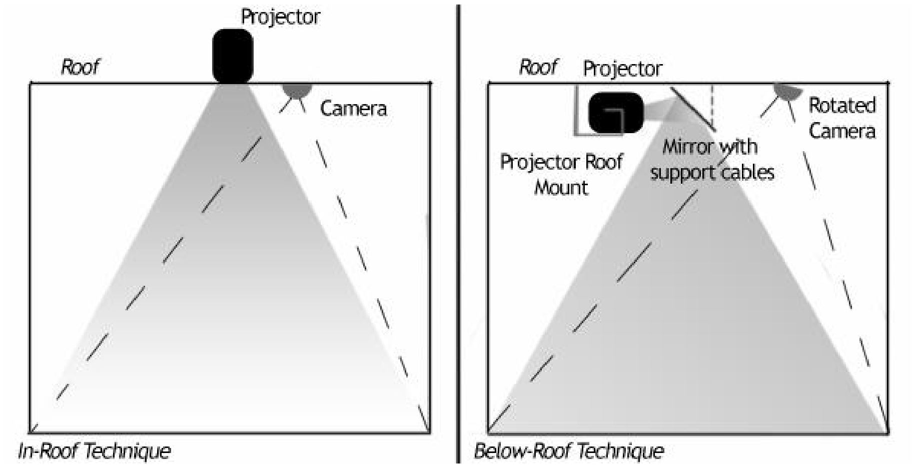


Figure 1: Roof comparison schematics

The tools required for every interactive floor differs. This list is only a guide to what tools you would generally need.

- Projector: Try keeping the projector as light as possible. Also use a wide-angle lens for the projector – or a short-throw projector – to ensure the entire floor gets covered.
- Camera: Use a camera with a wide-angle lens to ensure the entire projected image gets covered.
- Hammer and nails or screwdriver and screws: Depending on how you plan to install the camera into the roof.

Step by Step Construction:

Warning: Use a stable ladder. Have an extra person with you to help as a projector falling onto you will result in injury.

Step 1: Mount the projector

First, you need to make a hole in the roof for the projector lens to fit through. Detailed intructions on this cannot be given because of the vast number of roof types. The most important point for this step is that the projector should be secured 100% in the middle of the roof. Also ensure that the required area is filled by the projection.

Step 2: Install the camera

A CCTV-like camera will be most preferred due to the fact that they come standard with roof mounts. If yours does have a roof mount, use nails or screws to secure it to the roof. If your camera does not have a roof mount you will need to buy an extra camera mount and install it as per the mount's instructions.

Step 3: Ensure the safety of the setup

It is advised to put the setup through a test before proceeding. Have an extra person close to help with this part.

1. Wiggle the camera a bit to see if it is secure. If not, secure it more, otherwise continue.
2. Wiggle the projector a bit to see if it is secure. If not, secure it more, otherwise continue.
3. Usually you should leave the setup overnight and check the next morning to inspect for any safety issues.

Step 4: Testing

At this point you should have the installation done and ready for testing. How you plug the projector and camera into the computer is up to you. You will most probably need extension cables for the projector and camera.

Install the software of choice and test the setup, adjust as needed.

The tools required for every interactive floor differs. This list is only a guide to what tools you would generally need.

- Projector: Try keeping the projector as light as possible. Also use a wide-angle lens for the projector – or a short-throw projector – to ensure the entire floor gets covered.
- Camera: Use a camera with a wide-angle lens to ensure the entire projected image gets covered.
- Hammer and nails or screwdriver and screws: Depending on how you plan to install the camera into the roof.
- Projector mount: As light as possible, but strong as well.
- Mirror: Any will do.
- Mirror mount: Usually a metal arm that can be attached to any surface.

Step by Step Construction:

Warning: Use a stable ladder. Have an extra person with you to help as a projector falling onto you will result in injury.

Step 1: Mount the devices

Mount these in order (projector, mirror mount and mirror), as per the manufacturer's instructions.

Step 2: Install the camera

A CCTV-like camera will be most preferred due to the fact that they come standard with roof mounts. If yours does have a roof mount, use nails or screws to secure it to the roof. If your camera does not have a roof mount you will need to buy an extra camera mount and install it as per the mount's instructions.

Step 3: Ensure the safety of the setup

It is advised to put the setup through a test before proceeding. Have an extra person close to help with this part.

Wiggle the camera, mirror, and projector a bit to see if they are secure. If not, secure it more, otherwise continue.

Step 4: Testing

At this point you should have the installation done and ready for testing. How you plug the projector and camera into the computer is up to you. You will most probably need extension cables for the projector and camera.

Software Used

In order to calibrate the system, you can use touchlib or Community Core Vision (CCV). Any tracking software that supports the filters needed for diffused illumination will also work. You need to set your software to only detect large blobs. People moving on the floor will then be picked up optimally. A midrange rectify and midrange blur is suggested.

When setting up the software, ensure that you setup and test it in varying light conditions. This will greatly affect the setup. The recommended conditions are during the day, around 12:00, not overcast (if it can be avoided). This will give the greatest accuracy in how the floor will work on a day-to-day basis.

The following are the steps on how to calibrate the floor using CCV or Touchlib.

1. Start CCV or Touchlib
2. Get a flashlight
3. Shine the flashlight on the floor and calibrate the filter of CCV or Touchlib to only see the flashlight. In other words, set rectify to a rather high amount.
4. Turn of the flashlight, recapture background (press 'b') in CCV
5. Press 'c' to bring up the calibration, then hit 'c' again to startthe calibration.
6. Aim the flashlight at the green cross with the red circle.
7. Turn the flashlight on and turn it off again as quick as you can. It should be as a quick flash. CCV or touchlib should have picked it up as if it was a touch to a normal screen.

8. Repeat steps 6+7 until calibration is complete for all the crosses.

9. You should now have a quite accurate calibration.

Things to keep in mind while calibrating:

Try to stand clear of the camera's view area, this would interfere with the calibration.

Try to get the flashlight as close as possible to the camera. In other words, try to keep the light from the flashlight as close to a circle as you can. If you stand at a very low angle compared to the camera you'll see the flashlight making an elliptical light shape and thus making the calibration less accurate and more difficult to get right. If you can't get very close to the camera, calibration is still perfectly possible, it will just take some extra time.

Conclusion

This setup is fairly simple. Literally every setup differs so you need to play around with most of the steps to find what works best for your setup. The basic concepts covered here will always apply.

It is advised to have some multitouch experience before building this. That would help you identify problematic areas on the floor before you start, and will save time and money.

APPENDIX E: References

Please note that references are organized by section to which they are applicable. Reference citation numbers in the text reset per section, as indicated below.

Actionscript 3 (Flash)

[1] Adobe Flash, Wikipedia entry. http://en.wikipedia.org/wiki/Adobe_Flash

[2] “Flash for surface computing” by Manvesh Vyas http://www.adobe.com/devnet/edu/articles/manvesh-vyas.html

[3] flosc: Flash Open Sound Control http://www.benchun.net/flosc/

[4] Migrating from ActionScript 2.0 to ActionScript 3.0: Key concepts and changes by Dan Carr http://www.adobe.com/devnet/flash/articles/first_as3_application.html

[5] AS3: Dictionary Object by Grant Skinner http://www.gskinner.com/blog/archives/2006/07/as3_dictionary.html

Python

[1] Python Programming Language. http://www.python.org

[2] PyMT. A Python module for writing multi-touch applications. http://pymt.txzone.net

[3] Zelle, J. M., “Python as a First Language,” Proceedings of the 13th Annual Midwest Computer Conference, March 1999. Also available at: http://mcsp.wartburg.edu/zelle/python/python-first.html

[4] touchPy. http://code.google.com/p/touchpy/

[5] PointIR YouTube demonstration. PyCon 2008. http://www.youtube.com/watch?v=bEf3nGjOgpU

[6] libAVG. A high-level media development platform. http://www.libavg.de/

[7] pyTUIO. A python module for the TUIO protocol. http://code.google.com/p/pytuio/

[8] Kaltenbrunner, M., Bovermann, T., Bencina, R., Costanza, E.: “TUIO - A

Protocol for Table Based Tangible User Interfaces". Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation (GW 2005), Vannes, France, 2005. see also: http://www.tuio.org/

[9] Teiche, Alex. http://xelapondsstuff.wordpress.com/

[10] Hansen, Thomas. http://cs.uiowa.edu/~tehansen

[11] Pyglet. http://www.pyglet.org

[12] OpenGL http://www.opengl.org/

[13] Wikipedia: OpenGL http://en.wikipedia.org/wiki/OpenGL

[14] OpenGL Architecture Review Board, Dave Shreiner, Mason Woo, Jackie Neider, Tom Davis. OpenGL Programming Guide: The Official Guide to Learning OpenGL. ("The Red Book"). (Amazon link: http://www.amazon.com/exec/obidos/ASIN/0321481003/)

[15] http://nehe.gamedev.net/

[16] Wikipedia. GUI Widget. http://en.wikipedia.org/wiki/GUI_widget

[17] PyMT API Documentation. http://pymt.txzone.net/docs/api/

[18] Wikipedia. Transformation Matrix . http://en.wikipedia.org/wiki/Transformation_matrix

[19] Affine transformations. http://www.leptonica.com/affine.html

[20] Wikipedia. Angle http://en.wikipedia.org/wiki/Angle

[21] Wikipedia. Euclidean Vector. http://en.wikipedia.org/wiki/Euclidean_vector

Gesture recognition

[1] Grafiti : Alessandro De Nardi

[2] Real-time gesture recognition by means of hybrid recognizers : Corradini Andrea

[3]The Biological Foundations of Gestures: J. Nespoulous, P. Perron, A. R. Lecours.

[4] A Hidden Markov Model-Based Isolated and Meaningful Hand Gesture Recognition : Mahmoud Elmezain, Ayoub Al-Hamadi, J¨org Appenrodt, and Bernd Michaelis

Miscellaneous

[1] Han, Jerfferson Y. "Low Cost Multi-Touch Sensing through Frustrated Total Internal Reflection." Symposium on User Interface Software and Technology: Proceedings of the 18th annual ACM symposium on User interface software and technology. Seattle,WA, USA, 2005. 115-118.

[2] Gettys, Edward W, Frederick J Keller, and Malcolm J Skove. Classical and Modern Physics. New York: McGraw-Hill, 1989.

[3] Real-time gesture recognition by means of hybrid recognizers : Corradini Andrea

[4] The Biological Foundations of Gestures: J. Nespoulous, P. Perron, A. R. Lecours.

[5] A Hidden Markov Model-Based Isolated and Meaningful Hand Gesture Recognition: Mahmoud Elmezain, Ayoub Al-Hamadi, J¨org Appenrodt, and Bernd Michaelis

With contributions from 18 professionals and academics with over 50 figures and 100 pages, the “Multi-touch Technologies” book provides a wealth of practical and theoretical information and a fresh perspective in the field of multi-touch. Being the first publication in its domain, authors have ensured that this revison contains new information, insights, latest research and practice in an accessible manner.

This book covers areas including but not limited to:

- Different methods and tools to build a multitouch system
- How multi-touch systems work
- Challenges in user interaction, design and experience
- Open source applications, frameworks and libraries
- Companies and organizations working in this field