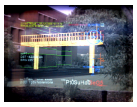
Tinmith evolution 5

Augmented Reality With Wearable Computers Running Linux

Wayne Piekarski, Bruce Thomas
School of Computer and Information Science
University of South Australia, Mawson Lakes, SA, Australia



Abstract

This paper describes the concept of augmented reality, the process of drawing virtual images over the real world using a head mounted display. Using wearable computers, it is possible to take augmented reality software outdoors and visualise data that only exists in a computer. The paper discusses all concepts in detail, and the hardware used to build the system, explaining the various components and costs. Examples of AR applications developed by the authors are explored, showing some ideas as to what the technology could be used for.

The Tinmith system is a complete software architecture designed to develop AR and other software that deals with trackers, input devices, and graphics. The design of the code is explained, including how it was developed. Tinmith is based on a completely free software system comprising the Linux kernel, GNU tools and libraries, the GNU C/C++ compiler, XFree86 graphics server, GGI graphics interface, OpenGL 3D renderer, PostgreSQL database, and Freetype font renderer.

1 Introduction

In recent times, numerous advances have been made in many areas of electronics and computing, allowing us to explore many previously unknown areas and open up new opportunities for research.

Technology such as powerful portable computers, head-mounted displays (HMD), global positioning systems (GPS), and software have made it possible to develop an augmented reality (AR) system which can be operated in a non-computer friendly, outdoor environment.

Imagine a user wearing a head-mounted display, allowing them to watch TV or their computer on a virtual screen projected into their eyes, similar to virtual reality (VR) technology you may have seen before. However, using an optical combiner inside the HMD, it is possible to see both the real world, and overlaid computer imagery at the same time. This ability to view physical and virtual worlds at the same time is called augmented reality, and is the focus of this paper. AR is an exciting new field and the authors feel that it has enormous potential for both commercial and recreational use.

Since this is a field with new technology that many people have not been previously exposed to, this paper will provide an introduction explaining the technology and components in some detail. Some of the AR applications developed by the authors are then discussed, showing how the technology can be used in the real world. Most of all, since this is a conference about Linux and software, the rest of the paper then discusses how the system works from a software point of view. We discuss the free tools we used (such as Linux, GNU, and X) to build the system, and the overall architecture of the software.

The software developed for this system is called Tinmith, which was started in 1998 as an undergraduate engineering project, and is now being used further as part of the author's PhD thesis. The system is always under continuous development as it is a test bed for many new research ideas. The main goal was to provide a complete architecture to develop AR and other applications that deal with trackers, input devices, and graphics, something that is currently not very well developed.

Most of all, the paper has been written to give information about constructing your own hardware. Hacking isn't just about software, its about using a soldering iron to modify your hardware, and making mistakes. The components and their costs are discussed, however, it is not a cheap field to be in as some of the hardware tends to be expensive and hard to find. So read on to find out more.

2 Hardware

The most important aspect of this work is the hardware, as it is what enables us to do our work. Some of the hardware is what would be termed 'exotic', in that most people do not know they even exist, let alone own them and dream of attaching them to a PC. Without these components, this research would not even be possible, so it is important to explain what each of them do. [AZUM97a]

The first thing to realise is that hardware enables software to do things, you can have the best software in the world but if you don't have a computer to run it on then it is useless. So in order to do AR while walking around outdoors, we need certain parts before the software can be executed.

So, to perform our research, we have built up a wearable computer [MANN96, BASS97] based around a standard PC laptop, mounted on a backpack. When walking around outdoors, traditional components such as big displays and keyboards are all useless, and so these have been replaced with more exotic components such as a GPS receiver, 3-axis magnetic compass, head mounted displays, and custom built input devices.

2.1 Integration

Everything needs to be carried around by the user, making them completely independent and able to move autonomously. In order to connect everything together and make it portable for outdoor use, each of the components are firmly attached to a hiking frame, with the cables tied up and secured. The figure below shows each of the devices mentioned in this section, plus the batteries, straps, and cables used to hold everything together.

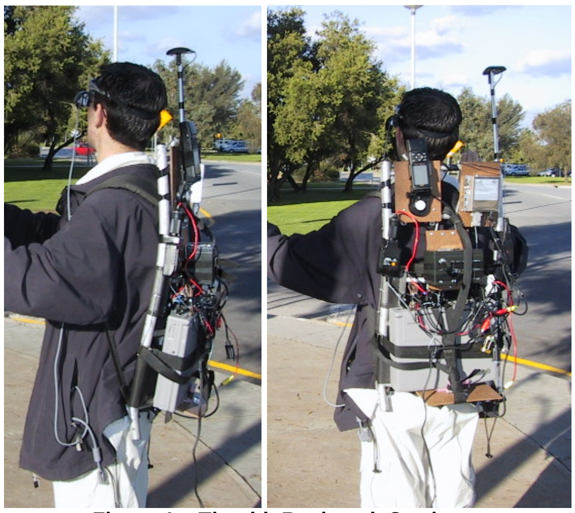


Figure 1 – Tinmith Backpack Outdoors

The first thing that most people observe when seeing the backpack is that it is heavy and bulky, and we agree with them. But it does work however, and the main objective of the project is to perform research into augmented reality – the wearable computer aspect is not so important. As a result, flexibility with hardware and ease of use are the most important driving factors. Given sufficient manufacturing and financial resources it would be straightforward to produce a smaller footprint device containing all the components fully integrated, but this is not the area of interest. Some companies are starting to produce hardware for this application and so it is only really a matter of time.

2.2 Portable Computer

The core of the system that brings everything together is the laptop computer, which is used to process the information from the sensors, and then provide feedback to the user looking through the HMD. For our system, we initially used a Toshiba 320CDS (P-200), although now we use a Gateway Solo (PII-450) which is considerably more powerful. Laptops are very useful for a wearable because they are already portable, highly integrated, are power efficient, and you don't have to solder anything. It is also possible to buy small, embedded OEM biscuit PC boards, which usually have more I/O capabilities, but require you to build the case and connectors yourself.

Any kind of laptop can be used for a wearable, depending on your personal taste, needs, and financial resources. Some important factors are CPU, memory, video, I/O, and Linux compatibility - and most Pentium-based laptops work quite nicely. The most critical ability is to have a laptop with serial, USB, and PCMCIA connectors to allow the connection of various kinds of devices.

2.3 Head Mounted Displays

The most obvious part of the system is the head mounted display. We use two kinds of Sony Glasstron devices: the PLM-S700E, which runs at 800x600, and the PLM-100 display, which runs at NTSC resolution with a VGA converter.

A HMD is a relatively simple device, (although the most expensive) containing an LCD display similar to that used inside a small TV, and uses mirrors and lenses to steer the light into the user's eyes. When wearing a HMD, the wearer is given the impression of looking at a large screen floating several metres away.

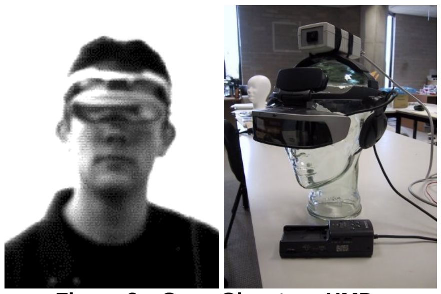


Figure 2 – Sony Glasstron HMDs

A traditional HMD for virtual reality is opaque, in that the user wears it as part of a helmet, and the rest of the world is completely blacked out so you are immersed into the VR environment. When the power goes, out everything is dark and you must take the display off to see anything. The key difference with the HMDs we are using is that they are partially transparent.

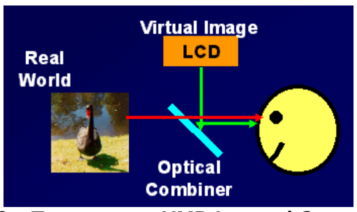


Figure 3 – Transparent HMD Internal Construction

Instead of using standard optics like in opaque HMDs, these use a half-silvered, optical combiner mirror. The image from the LCD display is still shown to the user as before, except now the mirror also allows light from the real world. The end result is a ghosted image where you can see the physical world but also computer generated imagery, as shown in the figure below.

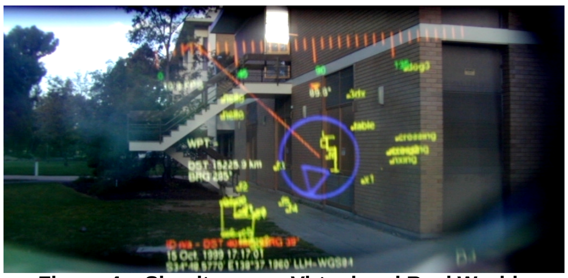


Figure 4 – Simultaneous Virtual and Real Worlds

There are many kinds of HMDs on the market, although most of them tend to be expensive (thousands of dollars) as there is not a large market for these yet. Also, HMDs vary in image quality and field of view depending on the price. Some displays have very dull looking images, do not work well in sunlight, and have very low resolution. The term field of view (FOV) is defined as how much of the user's vision the display can draw over, the larger the view then the more complex the internal optics are.

Recently, a new generation of laser displays has emerged, which paint an image on to the retina in the back of the eye directly. These displays have a much higher quality image, and will soon be small and cheap enough to be actually used. These are still being developed but are something to look out for.

2.4 Position Tracking

In order to allow the computer to draw images that match the real world, it needs to know where you are. When trying to overlay three dimensional models over the real world, having an accurate position is very important, otherwise the images will not overlay correctly and the user will have trouble understanding the image.

Since we are operating outdoors, we use US Global Positioning System (GPS) satellites, along with a Garmin 12XL receiver unit. The GPS receiver calculates its position once per second by working out its distance from a constellation of GPS satellites in space. Due to atmospheric noise and intentional signal degradation by the US Department of Defence, (selective availability) the accuracy of GPS can vary to around 30 metres, which is not acceptable. By using a differential GPS receiver, it is possible to receive correction signals from base stations that allow us to get position updates that are at around 5 metres accuracy, although this can vary.

Position trackers rely on some kind of infrastructure being in place before they can be used. Systems exist that use magnetic and ultrasonic trackers, although these have limited range and require fixed transmitters. Other systems use video cameras to capture images to work out location, but image recognition is immature and requires prior knowledge of the surroundings. The advantage of GPS receivers is that their reference points are in orbit, meaning they are visible almost all the time, and work anywhere in the world. The only catch with GPS is that it does not work indoors.

Figure 5 – Garmin 12XL GPS

Most GPS manufacturers support the NMEA-0183 standard, which allows us to transmit the GPS position information over an RS-232 serial cable to a PC for processing. This format is very easy to process and contains the current world location in latitude/longitude/height (LLH) values.

Since the GPS receiver is only being used as a way of calculating position, having a unit with an inbuilt display or map is a waste of money and not useful, as it is mounted onto the backpack. When purchasing a GPS unit, a good quality receiver is the criticial part, as some units cut corners here. A 5 metre (using differential) GPS unit can be bought for about $500, while more accurate units (from 1 to 50 cm) range from $8,500 to $50,000.

2.5 Orientation Tracking

Just knowing where the user is located in the real world is still not enough for the computer to know everything about where you are and what you are looking at. The heading, pitch, and roll of the user's head is also important, as people very rarely look exactly north. As a result, we use another sensor, a TCM2-80 from Precision Navigation.

This unit uses a 3-axis magnetometer to calculate heading relative to north, and pitch and roll. A fluid filled sensor is used to provide extra calibration information to the onboard microcontroller for processing, and the data is then sent to a PC via an easy to process serial protocol, at a rate of 16 Hz.

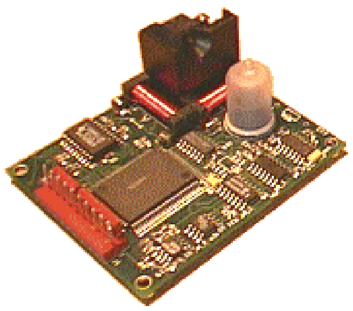


Figure 6 – TCM2 Magnetic Orientation Sensor

The device is mounted onto the HMD, (which is worn by the user) so that it can sense the movements of the user. As they rotate their head, the display updates itself to match.

There are not many devices of this calibre on the market (due to a small demand) and vary in price depending on accuracy and range. The price for the TCM2 range of trackers is around the $1000 mark, which is cheap in comparison to other units. The problem with the TCM2 compass is that it tends to jitter and suffers from magnetic distortion. Large metal objects such as cars and street signs tend to cause the compass to output incorrect information. Other techniques like using gyroscopes and accelerometers are immune to interference, but tend to have problems with drift and alignment. One solution is to use hybrid techniques, [AZUM99] that combine magnetic sensors and gyroscopes together to take advantage of their individual strengths. The Intersense-300 (for $8,500) uses this technique to produce output that is much more stable and accurate, and is something we would like to acquire in the future.

2.6 Miscellaneous Hardware

To provide networking, a Lucent WaveLAN card is used. These operate at 11 Mbit/s and have reasonable range around the building. They tend to suffer from interference from large machinery and thick buildings however, so we are experimenting with antennae placement in the building.

One problem with current PCs is the lack of serial ports – each of the tracker devices sends its data via RS-232 cables, and most laptops only come with one port. As a result, we have had to use PCMCIA based adaptor cards to provide extra ports. These come at a cost however, a four port Quatech card costs about $800, and the connector contains a lot of very fine pins that can break easily when being used outdoors.

We are patiently waiting for USB technology to mature, then we can use RS-232 to USB converters and a hub to connect up as many serial devices as we like – this solution is a lot more expandable and cost effective than using PCMCIA cards. Currently, the converters are not properly supported under Linux, and also are not integrated enough to allow us to have ten of them without requiring lots of USB hubs and power supplies to match.

2.7 Input Devices

A portable backpack computer is useless if the user cannot interact with it. The HMD allows the user to receive information from the computer, but not to input information. Traditional desktop devices like keyboards and mice are not practical outdoors as they are too bulky and require a fixed surface. Instead, devices such as small forearm keyboards, touch pad mice, and track balls are required as an initial starting point at least.

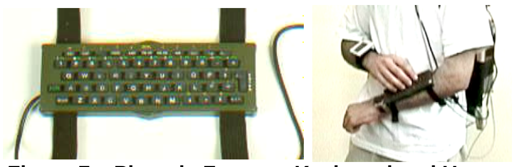


Figure 7 – Phoenix Forearm Keyboard and Usage

One thing that should be stressed is that we are in a different environment from a desktop, with much more freedom of movement, so why should we be restricted to primitive two-dimensional input devices from 20 or more years ago? Rather than try and fit a 3D immersive environment around 2D devices, why not use devices that are designed for the environment? As a result, other more exotic input means like speech recognition and 3D input devices are also currently being looked into.

At this point in time, we are experimenting with using USB cameras to capture video, then perform image recognition of special marker patterns to allow the computer to work out the position and orientation of the user’s hands. Hand tracking is traditionally performed indoors using expensive magnetic trackers, but these are too bulky and unworkable outside. By using hand gesturing, voice recognition, and home made data gloves, we hope to develop a useable user interface for AR. This user interface will allow users to manipulate and create new objects in a 3D outdoor AR environment, as shown in the mock up figure below.

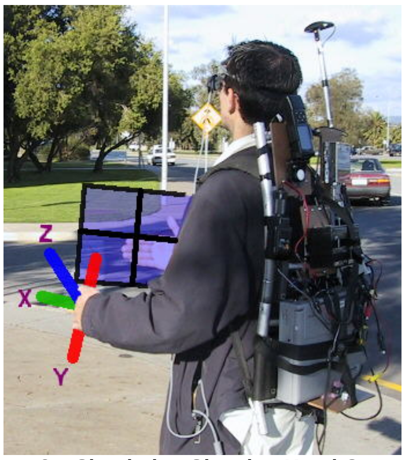


Figure 8 – Simulation Showing Hand Gestures Manipulating 3D Objects

3 Outdoor Augmented Reality

Given the previously integrated hardware components, plus a suitable software system, it is possible to take the equipment outdoors to use. [AZUM97b, FEIN97] We have implemented a number of AR applications using the Tinmith system, and these will be explained in the following sections.

3.1 Outdoor Problems

As with most research, it usually takes place indoors under nicely controlled conditions. Factors such as lighting, metal, power, and portability can be easily controlled, as everything is static.

Moving outdoors is a real challenge because everything must be portable, power must be carried, each unit uses a different supply voltage, and fragile connectors easily break. Things that work indoors, like the latest and greatest processors, hard disks, and 3D cards, cannot be made portable. Many tracking techniques stop working in large areas. As a result, sacrifices must be made in order to work outside.

Probably the biggest challenge with working outdoors is the cabling and integration – most PC equipment is very fragile and not designed to be moved and put under stress, and as a result, things tend to break a lot. Wiring needs to be tied to the backpack, but at the same time needs to be able to be removed once indoors when things are changed. If the HMD does not work, you cannot use the laptop screen because it is fixed down to the backpack. The tracker devices fail to send serial data due to fragile PCMCIA connectors. Sometimes things don't work, and then a faulty connector will begin to work again for no real reason.

As a result, working with this equipment can be frustrating at times, and going outside usually involves quite a bit of preparation, including going down three flights of stairs in the CS building from our lab. In most cases, things are tested indoors as much as possible before venturing into the hostile outdoor environment.

3.2 Wire Frame Augmented Reality

Once the hardware issues are resolved, the real research is in augmented reality however. The main output of the Tinmith software system is 3D rendered output. This renderer is similar to that used for games such as Quake, except the renderer is a lot simpler and has been optimised for AR, and runs in a variety of different modes. The different versions of Tinmith each have renderers of different capabilities, and so the release number will be mentioned for each one.

In this example, we are using the wire-frame based renderer in Tinmith-III. A model of some UniSA campus buildings was carefully designed in AutoCAD over a period of about two weeks, (most designers use this program so there is no choice in this matter) using information from the GPS to align it with the real world. The figure below shows roughly what the model looks like from an aerial perspective, and the images presented here are based on this.

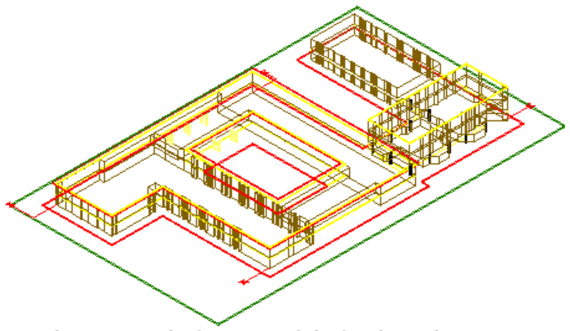


Figure 9 – Wireframe Model of Selected Campus Buildings

For the renderer, instead of the user controlling the position of the camera, the GPS and compass are used instead. So as you walk around the real world, the computer renders a scene to the display that attempts to match the real world.

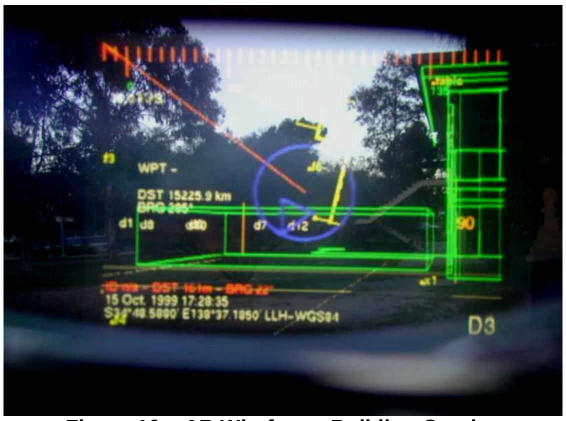


Figure 10 – AR Wireframe Building Overlay

The figure above shows one such example, showing the edge of the building on the right, and the computer has overlaid the wireframe in green over the top. In the centre of the display is a green box, which was used to visualise an extension to a lecture theatre at UniSA. The wireframe is useful for overlaying on real objects, but not very useful for visualising non-existent objects. As a result, a solid renderer is used to produce better displays and was introduced in Tinmith evo4. The rest of the screen components will be explained in a later subsection.

The wireframe model does not line up with the building due to tracker errors – the GPS and compass are not perfect and as a result the image tends to drift and jitter across the screen instead of being perfectly stable. [AZUM99] This can be partly fixed with more expensive equipment, but there is no such thing as perfect tracking, and there are still problems with the computer not rendering frames to the display fast enough. As the user moves their head, the tracker returns a value of the current position, which is sent to the computer with a small delay. The computer then redraws the display, then waits for the retrace in the HMD to redraw the screen. This all takes time (40 ms and up) and even delays near 1 ms can be noticed by the user.

3.3 Architectural Visualisation

The new solid renderer in Tinmith evo4 allowed more powerful visualisation – rather than just looking at enhancements of existing objects, it allowed us to see what things would look like before they are built. This is useful for people like architects and town planners, they can use the computer to see in real-time what changes to a city landscape would look like without having to use a bulldozer. To do this, the AutoCAD wire-frame model was extended to contain an air bridge connecting two of the campus buildings, drawn using solid polygons.

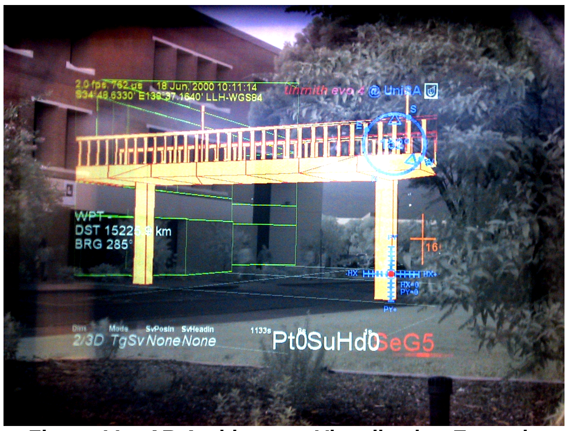


Figure 11 – AR Architecture Visualisation Example

The figure above shows the scene outdoors through the HMD. Notice the green wireframe outline does not line up to the building once again. Also note the limited area that the display can actually draw on, the field of view of most HMDs is quite small.

3.4 Outdoor Navigation System

The original Tinmith-II system was designed to be used as a navigation system by DSTO. To navigate outdoors in unfamiliar terrain, a map and compass must be used to avoid getting lost. This tends to rely on using local landmarks, plus counting paces and using a compass. This fails to work during the night when there is no light, or in featureless terrain. Also, if an obstacle like a forest or river is encountered while dead reckoning, you must plow through it rather than walk around it, otherwise you will lose your position. However, using a suitable navigation system mapped to a HMD, it is possible to see exactly where you are in the world, what objects are around, dangerous places to avoid, and steering instructions to get where you are going. The goal was to improve situational awareness, make someone's job easier, and give them time to think about more important problems.

The main interface for this navigation system is a 2D top down gods-eye view of the world, with fixed information overlaid on top, shown in the figure below.

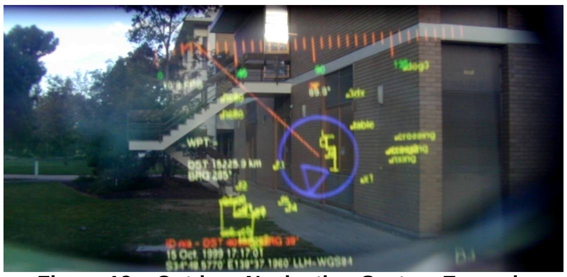


Figure 12 – Outdoor Navigation System Example

At the top of the display is the compass heading, which is represented as notches and a value every 45 degrees. As the user rotates their head, the compass updates itself and scrolls left or right to indicate the new heading. The various pieces of text information placed around the display are used to show position, GPS accuracy, and steering instructions to the nominated waypoint.

At the centre of the display is a blue circle indicating current position at the centre, with a blue triangle indicating the direction of where the user wants to go. Shown in yellow is the outline of the buildings in the area, which could be used to walk around the buildings. A red line with crosshair is controlled with a touch pad mouse to designate new waypoints on the map. The entire display is presented as a top down view, where the direction the wearer is facing is up. Every visual cue is rotated in real time as the user moves around.

3.5 Integration With DSTO DIS Simulations

As part of our collaboration with DSTO, we modified the system (to produce Tinmith-III) so it would be able to interact with other systems. At DSTO, they run a lot of simulation software, such as ModSAF [ARMY99] and MetaVR [META99] (very expensive commercial programs). The ModSAF program is used to generate entities for virtual battles, containing code to generate realistic intelligence for tanks, helicopters, and soldiers. These entities generate DIS packets which are broadcast onto a network for other software to use. The Distributed Information Standard (IEEE standard 1278) is an open standard using UDP, designed to allow simulation software to share information. The MetaVR program takes in these DIS entities and renders them on realistic 3D terrain.

DSTO uses these programs for training simulations, and have tank and helicopter cockpits that people can sit in (using real helicopter controls and seats, and three projectors for an immersive feel) and interact with the virtual entities. As part of a joint research project, [PIEK99c] it was thought it could be useful if the DIS systems could share information with Tinmith. Using a Lucent WaveLAN card in the laptop, the DIS packets were passed from the DSTO network to the wearable by a Linux based desktop machine. This interface allows the DIS simulations to see the wearable moving outdoors, and also allows the wearable to see all the DIS entities. The wearable computer can then participate in the exercise just like anything else.

The screen shots above show the displays seen by the users indoors. The top four are MetaVR output, and the lower one is the ModSAF entity generator. Each of them are projected onto the walls of the development room at DSTO, giving a command centre like feel. As the wearable person walks around outdoors, people indoors can keep track of their location and current surroundings from the 3D rendered displays. The flying vehicles are generated by the helicopter simulator, which is being flown by a human pilot indoors. The wearable user can see all the simulated entities on their HMD. The goal achieved was to improve the situational awareness of all the users of the system.

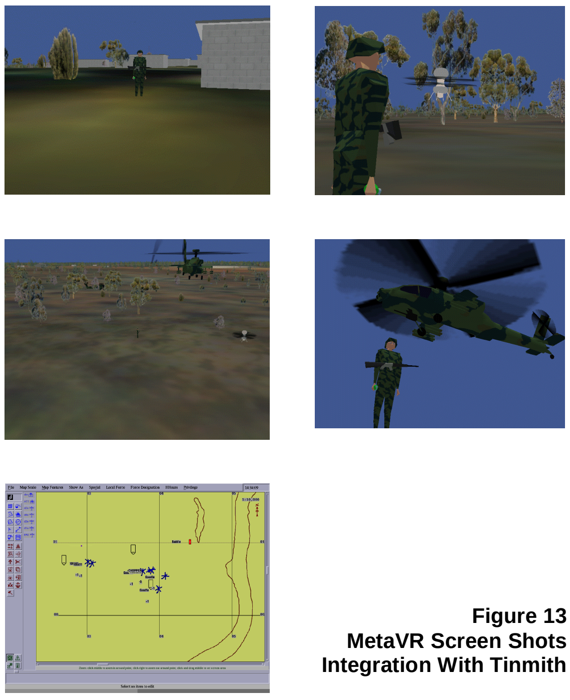


Figure 13
MetaVR Screen Shots
Integration With Tinmith

3.6 AR-Quake

A project just completed this year was performed by a group of computer science honours students. The display module used in Tinmith was turned off, and instead, the open source version of Quake was used to draw the displays. The program was modified to read UDP packets generated from Tinmith's sensor processing code, and the renderer was slaved to the GPS and compass positions.

By supplying the game with models of the UniSA campus, the user can run around outside and play a game of Quake, except it is now a physical game rather than just sitting down. If you want to move in the game, you must move in real life. By having multiple wearables and wireless networking, it is possible to play against other people in real life.

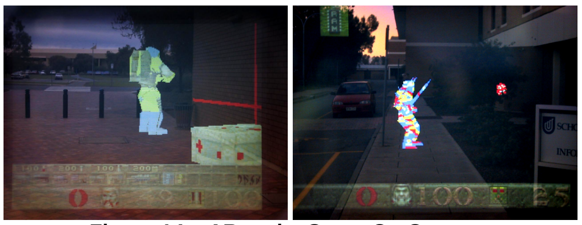


Figure 14 – ARquake Game On Campus

The figure above shows some example shots of a person playing the game. Some of the Quake monsters were modified to improve their visibility, and the controls for moving the player are non-existent, but apart from that, the game plays exactly the same as normal.

3.7 Summary

The previous subsections have covered some of the examples we have developed using the Tinmith system. These are designed to show what AR is capable of, but this is by no means the limit. AR and VR systems have traditionally lacked anything in the way of a user interface, and so the systems tend to be very much read only – you can't interact with them, and this limits what you can do. By making them more interactive, this should allow lots of new applications.

4 Software Support

When the Tinmith system was initially being designed, there were a number of requirements that had to be met for the development environment. This includes kernels, libraries, and development tools. Since a large investment of time was going to be made into something that could be reused in the future, making the right decision the first time was important.

4.1 Operating System

Some of the operating system requirements were:

- True pre-emptive multitasking
- 32-bit memory management and protection
- Interprocess communication
- TCP/IP network support
- Fast graphics rendering support
- Interface to diverse types of legacy hardware
- Low resource requirements
- High reliability

DOS is a good system if you want to talk low level with hardware, as it is only a program launcher and then gets out of the way, but there is no support for this any more, and does not support many of the requirements above.

Win32 has good support for new hardware being released, with all manufacturers writing drivers, and has a good working support base of APIs to do everything from multimedia to 3D. However, it does not allow tinkering with internals, no remote administration, wastes resources, has questionable reliability and protection, and just generally lacks the flexibility and control required to build a system like this.

The Linux and FreeBSD kernels, with GNU tools, and the X window system, tend to be very stable, and problems can be always be fixed. Resources are kept track of carefully, and used minimally, and it is almost impossible to crash the systems due to their design. The support for most hardware (both new and legacy) is excellent, with well-written drivers. The only catch is that driver support tends to lag by a few months for new hardware, and some rare hardware takes longer. Also, the APIs for things like USB, sound, multimedia, and 3D are also lagging behind Win32, although just recently this is starting to change. Many of these problems are caused by hardware manufacturers, (the developers of these drivers do a fantastic job) but at the end of the day, if you are building something to be used in the real world, you have to use what works. Our machines previously used Slackware, but now use RedHat 6 distributions. The system has been tested under FreeBSD as well, and will port to any Unix-like system that has all the libraries we use.

4.2 Free Software

The technical superiority of the system chosen was the main factor in selection - things like licensing, cost, or favouritism were not part of the process. If there were tools that could have done the job better, but with some kind of cost and no source code, we would have selected them instead. Having a kernel with open source is handy, but during the life of the Tinmith system, we have not yet made any changes to it. In reality, most people are not able to write their own video or network card driver, or fix a bug in the kernel, without a lot of experience and dedication. Since we are not making any changes to other's code, there are not too many differences between GPL, BSD, and other open source licenses.

4.3 Development Tools

The GNU tools form a major core for the development environment, using programs such as Emacs, and the GNU C/C++ compiler and debugger. These programs have proven to be of excellent quality, although there are some rare bugs that occur in the g++ compiler. The PostgreSQL database is used to store configuration information for the system instead of traditional text files. Other programs such as CVS and KDE are used to complete the code development environment, along with the rest of the standard programs included with most Linux distributions.

4.4 Graphics Support Libraries

Implementing everything from scratch can be quite difficult, and so where possible, other libraries were used to provide functionality. The most important area was graphics rendering, and so the main focus is on this.

The original prototype Tinmith-I system onwards used X protocol drawing calls, and double-buffering in pixmaps for smooth refreshes. The displays were quite primitive with only 2D objects, and so it was possible to render frames at a reasonable speed using this method. However, when rendering thousands of primitives per frame at very high refresh rates, X protocol breaks down. The amount of task switching and IPC between the X server and display module causes the system to waste most of its CPU time in the kernel instead of getting real work done. The one major flaw we can see with the X model for high performance graphics is that it requires IPC to do anything, although this is changing, with support for new architectures like DGA and GLX.

As a result of this, a more direct approach was needed. It would have been possible to use the shared memory extensions to send client rendered frames to the X server, but this would require writing a complete graphics library for handling lines, triangles, and so forth. This could take a lot of time to implement properly, and has been done before many times already. Unix users have tended to focus on the X window system as the only way to do graphics, but some times direct to hardware is the only way to go.

Libraries like SVGAlib allow the programmer to write to the display directly, but this library is very old and has limited support for newer video hardware. It does not seem to be maintained by anyone any more, and my impression is that it is basically dead. OpenGL hardware acceleration was not supported under Linux at the time (Only Mesa with slow software rendering, more on this later). The only available alternative was the Generic Graphics Interface (GGI) project. [GGIP00] The purpose of this library is to provide an abstraction layer for all graphics and input device hardware, and it works on the console, in X windows using both Xlib and shared memory, the new frame buffer drivers in the kernel, plus other special XFree86 extensions like DGA. We used the X shared memory target, as it was faster than the DGA (direct graphics aperture) interface, and it did not lock up the console. The GGI's ability to abstract away input devices on a variety of targets was a big plus, as handling this can be quite tricky in some cases.

So Tinmith-evo4 was created to use the GGI libraries, and with the speed restrictions removed, it was possible to add support for a full 3D polygon-based renderer. The performance increase was phenomenal, the number of system calls dropped to the point where strace() only showed IPC calls at the end of each frame being rendered. The GGI project is definitely a work in progress, the SVGAlib target does not support double buffering, and many of the other drivers are not yet complete. There were a few bugs present in the code for 16-bit mode which we had to hack out to make things work properly. Also, there is no decent font support built in, so a wrapper was written to the Freetype font renderer, allowing us to draw cached anti-aliased True Type fonts to the display at very high speeds. Speed is everything, and so we use large amounts of memory (which is cheap and abundant) to store things that we will use more than once, rather than create them repeatedly.

4.5 OpenGL Support

The GGI library was selected as it was the only available solution at the time that worked properly. Speed is everything, and so having proper 3D hardware handle the rendering is always preferable. Most of the processing in Tinmith happens quite quickly except for the rendering stage, which is the most complex when dealing with large models.

OpenGL was, and always will be, the real goal, but unfortunately, the Mesa software-only renderer was useless as it was too slow. Windows has always had drivers for OpenGL hardware, but these were not available to use with Linux. At the start of 2000, proper hardware accelerated 3D under XFree86 was almost non-existent, but recently a number of new drivers have been written to support this. When these drivers become stable, available with standard distributions, and are able to operate with laptop 3D chipsets like ATI Mobile Rage, then Tinmith will be modified to use this, as having hardware do 3D rendering is always preferable.

When the system was ported to GGI, a custom 3D renderer was written to handle the drawing of the models. However, the internal structure of this was optimised for OpenGL rather than the custom renderer, and the code was written so it would be easy to add the necessary OpenGL calls when it became feasible.

Due to the design of the GGI library, it should be possible to make GGI work with OpenGL, rather than having to throw all the previous code away. GGI supports dynamic loading and extensions, and some work is currently being done on writing a generic 3D API which fits into OpenGL – the progress of this project is unknown however.

4.6 Summary

Using free software tools and libraries has provided the ability to be very flexible and create some powerful software. The operating system is very efficient and can be stripped down to the minimum required software so that majority of the CPU time is dedicated to processing incoming data to render the HMD output. However, Linux currently lacks API support for things like writing games (which Tinmith is very similar to in requirements) and as a result, a number of wrapper libraries and hacks were made to make everything fit together. Having proper 2D and 3D with hardware support would make the development of the system a lot easier. Fortunately, a dedicated team of developers is making new progress in this area all the time, however, these things all take time and resources.

5 Software Architecture

5.1 Introduction

Kernels tend to provide only very limited services to hardware – open(), read(), write(), and some simple controls via ioctl(). However, only limited functionality is supplied and so the rest of it is implemented in user-land using libraries like GNU libC. Other systems like X provide interfaces to devices like mice and displays, to remove the complexity of having to deal with so many different kinds of hardware. However, just having low level libraries that can read desktop input devices and draw 2D pictures is not sufficient for AR/VR technology.

AR/VR is a very new area and is also immature, with lots of different kinds of trackers, input devices, and graphics hardware available. Each of them supports a different standard, and there are no standard APIs to interface to them.

In order to implement test AR applications for our research, a set of libraries needed to be developed to provide a set of basic functions that could be reused. The Tinmith system is an architecture that developers can use to develop AR (and other) applications that deal with trackers, input devices, networks, and graphics. Many of the interfaces to the kernel in libC are very primitive and so higher level interfaces are provided to hide away many of the details, making life easier for the developer. It also provides support at very high levels to process the protocols from various different tracking devices into a common format, for other code to read. This makes coding the rest of the system easier as the developer only has to worry about high level problems and not trivial ones like parsing tracker outputs.

5.2 Overall Modular Structure

The HMD snap shots shown as examples in this paper are really only the tip of the iceberg. The Tinmith system is not just a single process running on a machine, but in fact a collection of separate software processes that communicate with each other using TCP/IP. Each process tends to perform a separate function and can be plugged in together as certain needs arise. Proper Unix processes were originally used because threads were not fully stable at the time of initial design. Modules that have bugs can be kept separate so any memory problems can be isolated from the rest of the system and debugged individually. The architecture allows us to manage complexity by keeping things separate from each other, where possible. Some of the modules in the system are outlined in the following subsections.

5.2.1 Harvester Module

The most important module is the harvester, which gathers data in the system. This module polls all the serial ports available, and parses the incoming data from the variety of different hardware trackers plugged in. Tinmith defines special structures for storing position and orientation information, and the values are parsed and stored into these standard containers. These values are then made available on a networked object bus for other modules to receive.

5.2.2 Navigation Module

This module takes in position information, and uses it to make steering calculations for the user. The module contains a list of all waypoints in the system, as well as the currently selected one, and as each new position update is received from the harvester module, the bearing and distance are recalculated. These values are then made available for other modules to use.

5.2.3 Display Module

This module is the other core module, it takes in values from the harvester and navigation modules, and uses these to produce the display seen by the user. Any key presses received on the keyboard are made available to other modules that need to use this information for controls.

5.2.4 Sound and Watchdog Modules

This process continuously monitors the various variables in the system, and will sound audible (beeps or wave sound files) and/or spoken (synthesized voice) alarms when it is detected that these values have exceeded some parameter. For example, you could setup the watchdog module to sound an alarm or speak an announcement when you arrive within 100 metres of the destination waypoint.

5.2.5 Web Module

This process interfaces to CGI programs run by the Apache web server, allowing users to find out information about the wearable computer from a web browser. The module provides information about the wearable location and orientation, and refreshes continuously on the screen. This is a good example of how the system can be interfaced to systems that are of different design.

5.2.6 DIS and LSAP Module

The wearable computer contains an in-built object tracking module, which allows it to know the location of objects in the world, and follow their movements. This interface has been written to allow this tracker to share information (both ways) with the DIS (Distributed Interactive Simulation) and LSAP (Land Situational Awareness Picture System) based software used at DSTO. This ability once again demonstrates that the network architecture of Tinmith allows it to be extended and modified to fit the need relatively easily. Given something like a radar tracking device, it should be possible to modify the system to process targets and track moving real world objects. If the hardware is capable of it, the system should be able to handle it.

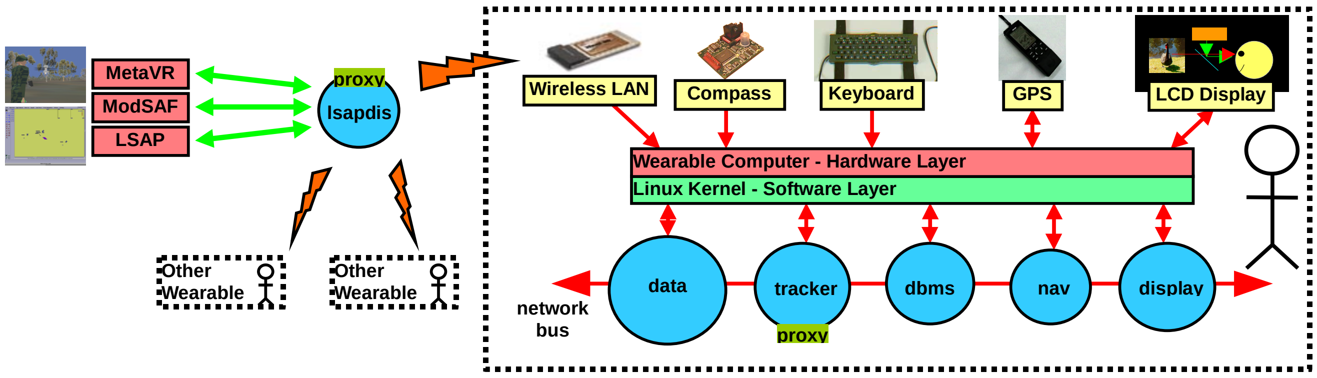
5.3 Module Communications

To interconnect modules, we used a client-server style architecture. The server is a data source for other modules, and it listens on a TCP/IP socket waiting for incoming requests from clients. A client that wishes to receive data will contact the server, and send a listen message to subscribe to it. Whenever the server updates the value of this data, it will send the new value out to all clients that have registered an interest in the message. A client receiving new data may use it to update the screen, or calculate new navigation parameters, for example. Note that many servers in the system are actually clients for other servers as well.

The entire system operates asynchronously, and is data driven; if there is no new data in the system, no action will be taken by any of the software modules. To illustrate this, consider the case of a new incoming position from the GPS. The harvester will process the new data, and then distribute it to all clients. The navigation module will receive this update, and recalculate navigation information. The display module will eventually receive an update from the harvester, and the new steering instructions from the navigation module, and use these to redraw the screen to reflect the user's new location.

5.4 Software Library

To implement the modular architecture, a software library to support this was designed, with goals being to be flexible, extendible, and layered. Layering was employed to provide increasing levels of abstraction for allowing modules to interact with the system at the appropriate level they require, while at the same time minimising code replication across the system, and localising possible errors. Rather than modules focusing on communicating with each other, the code



only does the tasks it needs to do, and then makes calls to library functions that actually make connections, subscribe, and process new incoming data. As a result, writing software modules to fit into the system is simple, and with many of the low level and repetitive details hidden away, also quite small.

The libraries provide functionality for distributed processing, asynchronous I/O, dynamic software configuration, and automatic code generation (among others):

5.4.1 Running modules in parallel over TCP/IP

Each of the modules are implemented as separate Unix processes. This allows modules to be distributed over multiple processors on one machine, or multiple machines due to the network support. The ability for the system to support this at a fundamental level improves the scalability for larger, resource intensive applications. For example, the outdoor navigation system was distributed over both the laptop and a second 486 wearable, which was included to increase the limited I/O capabilities of the laptop.

5.4.2 Asynchronous I/O event handling

The core of the library revolves around an event handler which monitors open file descriptors and waits for them to become available for reading or writing. When data arrives on the socket, the data is read, processed, and then handed to the calling code. As the complexities of doing I/O are abstracted away from the calling code, (to the point where even the type of transport is not specified) it is straightforward for the TCP implementation to be replaced by UDP by simply rewriting the library code. Slow serial links requiring writes to be buffered, and support for handling devices such as X servers, tty devices, and serial ports are integrated in.

One interesting feature of the I/O library is the ability to plug in simulated devices. During testing indoors, it was possible to plug in software simulations instead of real GPS and compass hardware, enabling us to test the modules easily. Running GDB through a HMD outdoors, with a keyboard on the ground, is not easy and very tiring on the eyes.

5.4.3 Dynamic configuration from the DBMS

Most software tends to use statically compiled controls, or possibly a configuration text file. Our system takes configuration to the next level by loading all system parameters such as the location of modules, port numbers, device names, and screen colours into a series of relational database tables. When the software initialises, it queries the database and loads the values required. By sending messages throughout the system when changes are made, it is possible for clients to reconfigure themselves by querying the database again. The software does not have to be restarted as would be required if the controls were static. The database proved to be very powerful because someone inside can change it remotely via the network if needed. A second feature is the strong type checking by the database engine (in our case PostgreSQL v6.5) rather than relying on parsing a text file, which could contain errors. This feature proved useful when performing testing outdoors, for example, tuning the various display options such as colours and font sizes.

5.4.4 Automatic message code generation

Each of the data values in the system (represented by messages) needs to be able to be sent in a portable fashion across the network. Since Tinmith is designed to be compiled on both big and little endian machines (Sun and SGI machines use different byte ordering than Intel), it is not good enough just to send the message in binary format. Also, as versions of the software change, it would be a good feature if older versions of the software could handle messages that have new fields added, but the old ones still remain the same. A custom program called STC is used to compile special message definition files into code and headers that is responsible for serialising and deserialising them for the network. The precompiler saves a lot of effort needed to write the code by hand.

5.5 Summary

The Tinmith software architecture was designed from the start to be flexible, and able to be extended in the future. The modular design enables new components to be plugged in easily, and many of the prototypes created with this system were not designed until after the architecture was complete. The architecture is data driven, and can benefit from distributing components as separate processes across multiple computers. A layered implementation eases application development, increases the overall reliability of the software, and gives a degree of device independence.

6 Problems and Notes

During the development of the work presented here, a number of issues and problems became apparent. Linux and its support environment is not perfect, and there are lots of places with room for improvement. However, improvement is always something the free software community has excelled at, a good example being the KDE and Gnome projects, and so given time these issues will surely be addressed.

6.1 USB Support

Proper USB support for devices like video cameras is currently only available in development kernels, and the code has been buggy at the best of times. Since the USB code is only very new, we had to put in a lot of effort to try and make Linux talk to the cameras. At times the machine would lock up, or refuse to communicate with the camera until reboot.

Also, support for many kinds of other cameras and serial port interfaces is non-existent, as the manufacturers have not released specifications or no one has written a driver yet. We had to shop carefully for USB devices to ensure they would work at all. This is something that will improve with time, but as of now the support is limited.

6.2 Graphics

Hardware accelerated 3D under XFree86 is just coming of age now. New extensions to XFree86 v3 and v4 have been made to support certain cards, and some manufacturers have supplied drivers for their hardware. However, many chipsets are not supported, or only with partial functionality. Also, tinkering with all these new drivers and patches takes a lot of precious time, having this supported as a standard feature in some type of future Linux distribution would be a lot easier to handle. For now, we are still using custom written renders, but using OpenGL will mean we can make the code simpler and faster at the same time, freeing up CPU resources for other uses.

6.3 Higher Level APIs

Linux and FreeBSD operating systems have wonderfully designed kernels, and good low level support in libC. However, there is not much higher level functionality beyond that. Authors who wish to write games for Linux must currently roll their own libraries for sound, video, input devices, and graphics, or they have to hunt around for development libraries which may or may not do the job properly. This can take a considerable amount of time and is probably a reason why some have not ported their games to Linux. There is currently an effort in the community to fix this, but these are far from complete and some projects may not be completed or have been abandoned.

7 Current Work

7.1 Tinmith Evolution 5

The Tinmith system (up to version four) worked quite well for a number of years, however there are also a number of limitations and problems with the design. While the design had some features that seemed like a good idea, some of them turned out to be not used but a performance penalty was still imposed in the design. Also, there were limitations in the design which were making it hard to implement new user interface techniques that we wanted to try out.

As a result, a new Tinmith design, evolution 5, written in C++, was put on to the drawing board in May, and work is currently progressing on that. The first major difference is the use of C++, version four used only standard C but portions of the code were dedicated to providing object oriented features. In reality, if you are using OO, you should use a proper OO language. The code contained a lot of void* pointers internally and this was starting to cause confusion and problems. Also, extra features like the STL provide functionality that does not have to be implemented yourself, and generally more efficient at the same time.

The other major problem was the software modules – being implemented as separate Unix processes required them to use IPC to communicate with each other. As a result, processing large amounts of data also required large numbers of system calls to move all the data around. Waiting for the kernel to task switch around four or more processes takes time, and this was causing lag in the renderer output, and as mentioned before, any lag that is preventable should be eliminated to improve the quality of the system. Serialisation, rather than parallelisation, is the key to efficiency here. Due to the way the libraries were written, it was not possible to combine all the modules together into one without making changes everywhere, and the effort was not worth it.

So as a result, the code was converted into C++, each part of the system was either modified or rewritten to fit the new design. The low level parts were rewritten into C++ code, and the high level code (which took the most time to write) like the renderer, parsers, navigation calculations, 3D modeller, and graphical displays – were all ported over relatively easily. The original code has now been completely converted and works as before, but now as a single process with no threads. Spreading the system over a network is still supported, but not a requirement like before. The streamlined design makes the system run faster and more efficiently, and the code is easier to understand overall.

The lesson learned from this is that C++ (used correctly of course) can write programs that are much easier to understand, especially when they become very complicated. Trying to solve an object oriented problem using traditional functions and structures is really a hack and produces code which can cause problems in the long run. The effort required to convert from C to C++ has already paid off, with new changes being easier to implement.

7.2 Software Availability

The Tinmith system is currently not available to the public at this time. This system is the vehicle I use to perform the research for my PhD thesis, and is currently undergoing major changes to support new user interface techniques I am developing. The architecture is usually having changes made, and is not complete to the point where someone else could write their own applications easily – there is minimal documentation except for my own designs and notes.

During the completion of my research, the Tinmith-evo5 architecture will be finalised, and used to develop a number of custom testing applications for user evaluation trials. At this time, the code should be useable by others and the university may allow the release the code to the general public. Also, I do not have enough time at this point to prepare the code for distribution, accept patches or code contributions from other authors, and at the same time complete my PhD thesis on schedule.

Keep an eye on the web page references included at the back, or contact the authors for more information if you would like to work with Tinmith.

8 Conclusion

The purpose of this paper was to give the user an idea as to what augmented reality is, and some of the things it can do. Tinmith is one of only a handful of augmented reality systems in the world today, and with the new changes for evo5 being made, will also be one of the most advanced.

Currently, most AR and VR systems tend to be read only, the user participates by looking at things. With Tinmith, the authors intend to allow users to interact with the environment using custom built 3D input devices to manipulate 3D widgets and objects. The goal at the end is to produce for 3D environments what people have been using on 2D desktops. There are a number of research problems that need to be solved before this is possible however, just using 2D techniques from the desktop is not good enough.

Augmented reality is an exciting field to be participating in, it is relatively new, and there are lots of ideas to be explored. Rather than just reimplementing the wheel, we are researching and designing the original wheel itself.

9 References

[ARMY99] US Army Simulation, Training, and Instrumentation Command (STRICOM), ModSAF – http://www-leav.army.mil/nsc/stow/saf/modsaf/index.htm

[AZUM97a] R. Azuma, Survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6(4). 1997.

[AZUM97b] R. Azuma, The challenge of making augmented reality work outdoors. In *Mixed Reality: Merging Real and Virtual Worlds*. Y. Ohta, H. Tamura (ed), Springer-Verlag, 1999. Chp 21 pp. 379-390.

[AZUM99] R. Azuma, B. Hoff, H. Neely, R. Sarfaty. A Motion-Stabilized Outdoor Augmented Reality System. In *Proc. of IEEE Virtual Reality '99*, Houston, TX, Mar 1999. pp 252-259.

[BASS97] L. Bass, C. Kasabach, R. Martin, D. Siewiorek, A. Smailagic, J. Stivoric. The design of a wearable computer. In *CHI 97 Looking to the Future*, pp 139-146. ACM SIGCHI, ACM. 1997.

[FEIN97] S. Feiner, B. MacIntyre, T. Hollerer, A. Webster. A touring machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment. In *1st Intl. Symposium on Wearable Computers*, Cambridge, Ma, Oct, 1997. pp 74-81.

[GGIP00] Generic Graphics Interface Project, http://www.ggi-project.org, 2000.

[JACO97] M. Jacobs, M. Livingston, A. State, Managing Latency in Complex Augmented Reality Systems. In *Proc. 1997 Symposium on Interactive 3D Graphics*, Providence, RI, Apr 1997. pp 49-54.

[MANN96] S. Mann, Smart Clothing: The Shift to Wearable Computing. In *Communications of the ACM*, Vol 39, No. 8, pp 23-24, Aug 1996.

[META99] MetaVR, Inc., *MetaVR Virtual Reality Scene Generator* – http://www.metavr.com, 1999.

[PIEK98a] W. Piekarski, D. Hepworth, Outdoor Augmented Reality Navigation System – Project Documentation. University of South Australia, 1998.

[PIEK99a] W. Piekarski, D. Hepworth, V. Demczuk, B. Thomas, B. Gunther. A Mobile Augmented Reality User Interface for Terrestrial Navigation. In *Proc. of the 22nd Australasian Computer Science Conference*, Auckland, NZ, Jan 1999. pp 122-133

[PIEK99b] W. Piekarski, B. Thomas, D. Hepworth, B. Gunther, V. Demczuk. An Architecture for Outdoor Wearable Computers to Support Augmented Reality and Multimedia Applications. In *Proc. of the 3rd International Conference on Knowledge-Based Intelligent Information Engineering Systems*, Adelaide, South Australia, Aug 2000. pp 70-73.

[PIEK99c] W. Piekarski, B. Gunther, B. Thomas. Integrating Virtual and Augmented Realities in an Outdoor Application. In *Proc. of the 2nd International Workshop on Augmented Reality,* San Francisco, Ca, Oct 1999. pp 45-54.

[THOM97] B. Thomas, S. Tyerman, K. Grimmer, Evaluation of Three Input Mechanisms for Wearable Computers. In *1st Intl. Symposium on Wearable Computers*, Cambridge, Ma, Oct, 1997. pp 2-9.

[THOM98] B. Thomas, V. Demczuk, W. Piekarski, D. Hepworth, B. Gunther, A Wearable Computer System With Augmented Reality to Support Terrestrial Navigation. In *2nd Intl. Symposium on Wearable Computers*, Pittsburg, Pa, Oct 1998. pp 168-171.

10 Further Information

10.1 Reading Sources

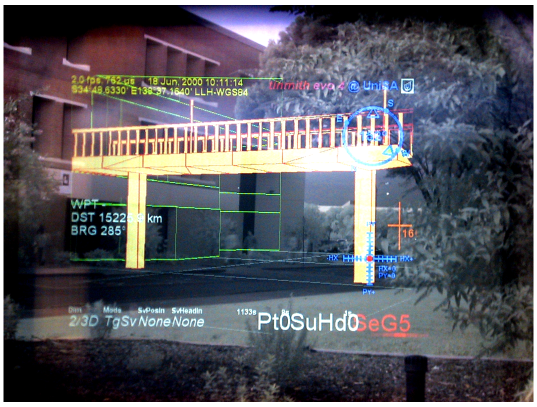
There is a lot of information about the topics discussed here on the Internet, use a search engine to look for it. A number of conferences proceedings from IEEE and ACM contain interesting material, such as:

- ACM CHI (Computer Human Interaction)
- ACM I3D (Interactive 3D Graphics)
- ACM SIGGRAPH (Graphics)
- ACM UIST (User Interface Software and Technology)
- IEEE ISAR (International Symposium on Augmented Reality)
- IEEE ISWC (International Symposium on Wearable Computers)

10.2 Acknowledgements

The authors would like to acknowledge the following people who have helped out with the project in the past:

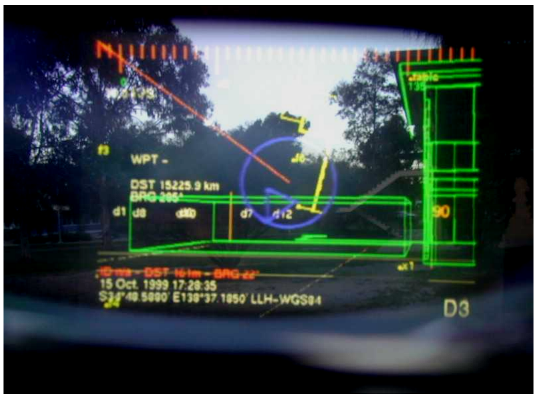
- QuakeVR Project - Ben Close, John Donoghue, John Squires, Philip De Bondi, Mick Morris
- DSTO - Victor Demczuk and Franco Principe
- Campus Models – Arron Piekarski



10.3 About The Wearable Computer Lab

The Wearable Computer Lab is part of the School of Computer and Information Science, Advanced Computing Research Centre, at the University of South Australia. Research areas in the lab include outdoor wearable computing, augmented reality, 3D user interfaces, computer graphics, and virtual reality.

Apart from just hacking code, we have also been known to use soldering irons and sticky tape on computer hardware occasionally. The original backpack in 1998 had the laptop attached to the backpack using packing tape, and when wearing the computer, an assistant was always present to apply more tape as needed.



10.3.1 The Authors

Wayne Piekarski is currently doing a PhD at UniSA, the thesis being titled, *Human Machine Interfacing In Augmented Reality Worlds.* He is the person behind the Tinmith system, designing and coding it to support his research. He is an avid Linux fan, starting off in 1994 with Slackware 2.0 and kernel 1.1.59, although has been known to use Windows and FreeBSD boxes as well.

Dr. Bruce Thomas is a researcher and lecturer at UniSA, performing work in a number of areas including AR, VR, wearable computers, and user interfaces. He is the supervisor for Wayne's PhD research.

10.3.2 More Information

For more information about our work, please visit the following URLs or email us. We always like to hear about what people think about our research. If you feel that there is something you would like to contribute, then please contact us.

Information about the wearable lab
http://wearables.unisa.edu.au

Information on the Tinmith project
http://tinmith.unisa.edu.au

UniSA

The author's home page and email address
http://www.cs.unisa.edu.au/~ciswp
wayne@cs.unisa.edu.au
thomas@cs.unisa.edu.au