Leveraging Graphics Hardware for Vision Based Human Computer Interaction

Sven Olsen

ECE 432

September 27, 2005

Abstract

We present a system which allows users to draw on arbitrary display surfaces. The system is implemented using consumer electronics; data is gathered with a webcam, and most video analysis is done on a desktop graphics card. Two variants of the core system are discussed. In the first, users draw on a computer monitor using a laser pointer. In the second, a light pen is used to draw on a display surface created by back-projecting onto a plexiglass plate. The laser pointer system presents a more challenging data filtering problem. These challenges may be addressed through additional processing, though the results from the back-projection system remain superior.

Contents

List of Figures

Chapter 1

Introduction

Video cameras and computer vision algorithms have often been used to facilitate human computer interaction. Our system is distinguished by its use of commodity graphics hardware and by its relative simplicity.

1.1 Motivation

As the hectic pace of development in consumer electronics moves ever forward, high-resolution video cameras and powerful parallel processors are becoming commonplace. The current generation of video cards are capable of general purpose parallel computing at 50 GFlops or more; and a $40 webcam can capture video at a resolution of 640x480 and a frame rate of 30 fps.

The goals of this project are as follows:

- Assemble a vision based interaction system out of cheap and commonly available consumer electronics.
- The system should be both simple and robust.
- Maximize the resolution of the system while minimizing latency. Ideally, users should be able to draw detailed figures on a display surface in real time using an optical marker.
- Exploit graphics hardware to the maximum extent possible in order to better achieve the above.

1.2 Related Work

This project combines elements from three related subfields: computer vision, general purpose GPU computation, and human computer interaction.

1.2.1 Electronic Whiteboards

The specific goal of building a chalkboard-like computer interaction system has been a theme of work in human computer interaction systems for some time. One of the earliest systems was LiveBoard, [4], developed by Xerox research, which used a custom built pen and sensitive wall display system to allow interactive drawing on a large surface. Currently, the Mimio whiteboard capture system, a hardware system for turning normal projection surfaces into interactive whiteboards is being marketed in the $1000 range, and uses ultrasound tracking technology to pinpoint the location of a stylus on the board.

Another recent system which allows for fairly precise interaction with a projected surface is the Diamond Touch technology developed at the Mitsubishi Electric Research Labs, which uses an electric current conducted through user's bodies to detect touched point positions on a front projected screen.

Low cost technologies for converting static whiteboards into digital form have been explored. In Zang '02 [17], images of a whiteboard taken with a digital camera are processed to produce a high quality record of the board's contents.

1.2.2 Vision Based HCI

Researchers in human computer interaction have often made use of cameras pointed at display surfaces. Olsen and Nielsen [11] and Cheng and Pulo [2] present interactive systems in which laser pointers are used to interact with projected screens. Data gained from laser pointers is typically too crude to be useful for detailed drawing, however, it has been used to good effect for the manipulation of GUI elements, or for capturing raw data which is then used to drive a gesturing interface. Flachsbart et al. [6] and Apperley et al. [1] present systems in which videos of people moving around the projected displayed are analyzed in the course of the interaction.

Another project similar to our own is the "Visual Panel", presented by Wu [15], in which the motions of a finger on a normal piece of paper are tracked and analyzed with sufficient resolution and speed to allow figures drawn on the paper to appear on a computer screen.

1.2.3 GPUs in Computer Vision

Over the past two years, the increased programmability of graphics hardware has lead to a range of applications in computer vision. Montemayor et al. [13] presented a preliminary GPU-based particle filtering system. Fung [7] performed a number of vision tasks on GPUs, including a Cany Edge detection and simple hand tracking, and all of his code is available for download as part of the open source OpenVIDIA computer vision library. The OpenVIDIA functions have been used to build augmented reality systems, in which dynamic images were superimposed on rectangular surfaces in the user's environment.

Chapter 2

Implementation

2.1 Hardware

2.1.1 Video Capture

We experimented with three different video capture devices:

- A Cannon GL2 digital camcorder
- A Logitech Quickcam pro 400 webcam
- An Apple iSight Firewire webcam

All three capture sources could be instructed to capture at a resolution of 640x480. While the camcorder returns high quality images, the digital out transforms the video to compressed DV format before sending it to the computer, which introduces a large amount of latency into the system. The Logitech webcam had less latency, but had low image quality and operated at a slightly slower framerate (20fps). The iSight had very little latency, good image quality, and a reasonable framerate (30fps) - therefore we choose to use it in the final version of the system. This was by no means an exhaustive survey of consumer video capture devises, indeed, both the iSight and the Quickcam pro are several years old, and more recently introduced usb2 webcameras may well outperform both of them.

2.1.2 Light Pen and Back Projection

We constructed a back projection surface by attaching a piece of frosted acetate to a plexiglass plate. We have built a simple "light pen", consisting of a high intensity red light emitting

Figure 2.1: The back projection system, which consists of a plexiglass plate, projector, and webcam. Also highlighted are the drawing devices, the light pen and laser pointer.

diode, secured to the end of a marker in such a way that it activates when pressure is applied to the tip of the pen. Viewed from behind the back projection surface, the active pen shows up as a bright circle of diffuse light. This setup is attractive for several reasons:

- Using back projection allows us to avoid occlusion problems entirely
- The interaction of the LED and the frosted acetate creates a strong diffuse optical marker, so very little filtering is needed.
- The pressure activated pen is easy to control precisely, and well suited to our goal of creating an interactive drawing system.

The downside of this setup is that the use of a projector strains our goal of constructing something out of "common consumer electronics". That said, the cost of the other special

Figure 2.2: The active back projection system (see associated videos).

Figure 2.3: A pressure sensative light pen, constructed out of an LED and two AA batteries.

materials are negligible; the pen, plexiglass, and acetate have a net value of about $12, and while projectors are still fairly expensive, smaller, cheaper projectors are being intensively developed by companies such as Mitsubishi and Toshiba.

2.1.3 Laser Pointer and Monitor

A back projection system is not the only way that we can allow a user to draw on a display surface. We may also allow users to draw on their computer monitor. Pointing a webcam at the screen is easy enough, but, with this arrangement, occlusion tends to preclude the use of the light pen. In place of the pen, a laser pointer may be shined on the monitor to impose a user guided optical marker on the screen. Unfortunately, because the laser is highly directional, the amount of laser light observed by the webcam depends on their relative angles. In contrast to the light pen, the laser pointer tends to be difficult to control accurately. However, unlike the back projection setup, this variant of the system is very easy to construct.

2.2 Software

The software components of our system are implemented in a mixture of C++ and Cg. We interact with the graphics hardware through OpenGL, and we use DirectShow for interfacing with the video capture device.

2.2.1 Video Capture

Video frames are captured from the webcam using the Microsoft DirectShow library. A filter graph is constructed which connects the video capture source to `IID_ISampleGrabber` and `CLSID_NullRenderer` filters. The graph runs in its own thread, which introduces the potential for resource conflicts, as both the main thread and the capture thread need to access the captured video frame. One way to avoid such conflicts is to copy the buffer into a neutral memory area guaranteed never to be accessed by both threads simultaneously. If both threads copy the data into/outof the shared area before doing any processing on it, neither will need to wait while the other performs calculations on the data. It is preferable for the neutral memory to reside on the video card, rather than the main system, so that when the main thread copies the data into it's own memory space, it can take advantage of the higher memory bandwidth available on the graphics card[1]. The only way we have

[1]While the L1 cache of the Pentium 4 does have significantly higher memory bandwidth than the NVIDIA Geforce GT series (44 GB/sec vs. 36 GB/sec), a 640x480 24bit image is far too large to be stored in the L1 cache (and probably also too large to be practical in the L2 cache as well). Thus, if the neutral memory space were to be kept on the CPU side, it would need to be stored in system memory, which only has a memory bandwidth of 6 GB/sec [5].

found to achieve this is by using the pBuffer OpenGL extensions to specify a neutral patch of texture memory which is read as a texture by the main thread, but bound as a rendering context by the capture thread. As the interaction of the Microsoft multithreading libraries and the OpenGL pBuffer extensions are not well documented, it's unclear whether or not a more efficient implementation is possible, or even if the own method is guarantied to be thread safe. However, thus far the approach has worked well in practice.

2.2.2 Calibration

In order for our system to respond appropriately when it detects a point in the camera buffer, it must develop a mapping between the camera frames and the screen buffer. A mapping between points on the screen as viewed by the camera and the representation of those same points in screen coordinates is a mapping between two different views of points on a planar surface, and can thus be represented by a 3x3 homography matrix. Finding such a matrix given a set of known feature correspondences is a common task in both computer vision and image based rendering, we follow the example of Raskar et al. [14], and use a nonlinear optimization method, very similar to that introduced by Zang [16]. A range of alternate calibration algorithms are discussed in Hartley and Zisserman [10].

Gathering Features

In order to generate correspondences between the screen coordinates and points in the camera image, a series of boxes are displayed on the screen. A subtraction image is generated by comparing each frame coming from the video camera with the previous frame, and occlusion testing used to quickly determine whether or not there has been a substantial change between frames (See Figure 2.4). Assuming the background noise is relatively small, a large change will indicate that the current thresholded subtraction image contains the newly shown box. When such a change is detected the thresholded subtraction image is copied back to system memory, and the centroid of the points that survived the threshold operation are calculated. This centroid is then considered to correspond to the screen coordinates at which the current box is being displayed. The screen coordinates for the boxes are generated pseudorandomly (using a 2D Halton sequence, as in Keller [12]), and if a changed point marker goes unnoticed for long enough, the program will throw it out, and move on to the next point in the sequence. This way, a complete set of correspondences can be found, even if portions of the screen are not visible to the camera. Though any number of set correspondences with more than 4

elements could be used, we have been using 7-10, as they can be captured quickly (within about 10 seconds), and seem to produce good results.

Find the Homography Matrix

We need to find a matrix $\mathbf{H}$

$$\mathbf{H} = \begin{bmatrix} h_0 & h_1 & h_2 \\ h_3 & h_4 & h_5 \\ h_6 & h_7 & 1 \end{bmatrix}.$$

Such that any point on our projected screen ($\mathbf{u}$) will be related to a point on the projected surface ($\mathbf{x}$) by

$$\mathbf{u} \tilde{=} \mathbf{H}\mathbf{x}$$

This is done by solving the following minimization:

$$\min \sum ||\mathbf{H}x_i - u_i||$$

for some known feature points. Sadly, this is not a linear least squares problem, as $\mathbf{u}$ and $\mathbf{x}$ are in homogenous coordinates. The equation expands to:

$$\min \sum || \begin{bmatrix} \frac{h_0 x_0 + h_1 x_1 + h_2}{h_6 x_0 + h_7 x_1 + 1} - u_0 \\ \frac{h_3 x_0 + h_4 x_1 + h_5}{h_6 x_0 + h_7 x_1 + 1} - u_1 \end{bmatrix} || \tag{2.1}$$

This is a relatively simple 8 variable unconstrained nonlinear optimization problem. In [16], a Levenberg-Marquardt solver is used to solve this problem. We have taken a different route. At the end of the calibration phase our program spawns a temporary process which uses the free student copy of AMPL and the SNOPT nonlinear solver to solve Equation 2.1. This introduces a considerable amount of overhead, but as the problem is only solved once, during the initialization of our system, and as it is not a particularly computationally expensive operation, there would be little to gain in implementing and debugging a C++ solver.

Because of it's nonlinearity, the problem may contain local minima, so we should be cautious about our initial choice of $\mathbf{H}$. Zang '00 uses an approximate analytic solution to initialize $\mathbf{H}$, but we have found that in our own case, starting with the identify matrix provides acceptable performance.

2.2.3 Fragment Shaders

Programmable fragment shaders are one of the components of graphics cards best suited to general-purpose programming. Fragment processing occurs after the texture coordinates and screen position for a potential pixel render have been calculated. The role of the fragment shader in the rendering pipeline is to take this information and use it to determine whether or not the pixel should be rendered (typically by doing depth testing) and what color the rendered pixel should have. Programmers can write fragment programs that do anything they like with the information available to them, the only limitation is that the output of the program must be limited to a decision whether or not the pixel should be rendered, and if it is, the color that the pixel should have (several different color representations are allowed, the most information rich being a 4 element floating point vector). Fragment shader programs may read from any location in texture memory. Until recently, it was not possible use branching or looping instructions inside a fragment program, but the most recent NVIDIA cards have added this feature, and as it is a requirement of Shader Model 3.0, it is scheduled to be included in the coming generation of ATI cards as well. Fragment shaders are commonly used to implement convolution filters, iterative linear solvers, or fast matrix multiplications.

In our system, custom fragment shaders are used to filter the incoming video frames in order to find optical markers, to generate the thresholded subtraction images used by various other parts of the program, and to display the processed data on the screen.

The program defines three rendering contexts, the video capture context, a main processing context, and the display context. The first two may be used as texture maps as well as rendering targets, which makes them well suited to general purpose computation. The memory in the display context, however, is effectively write only.

The main thread acts as follows :

- First the data from the video capture context is copied into the main processing context.
- Then a sequence of fragment programs are applied on the captured data in order to reduce it to a small number of pixels which we believe represent the point being indicated by the user.
- Finally, the filtered data is added to a buffer containing the accumulated points from all prior filtering results. This portion of the texture is transformed by the homography matrix in order to render an image to the screen.

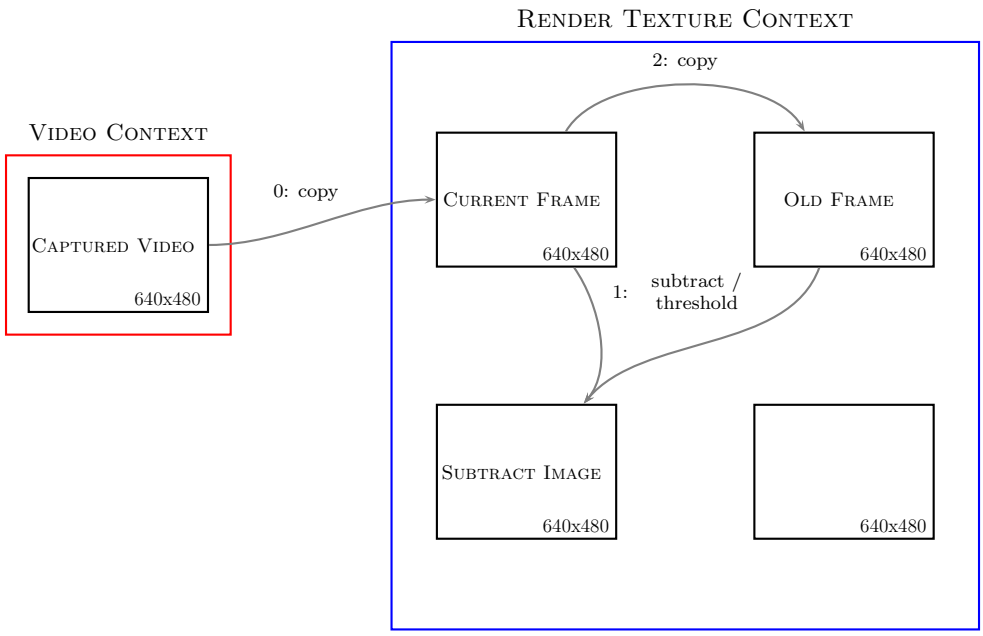


Figure 2.4: The fragment programs used in the calibration step. A similar series of operations are used to create the initial data for both the laser and light pen systems, though the copy operation is postponed until after the filtering steps have been completed. In the case of the light pen system, the incoming data is tends to be clean enough that no subtraction step is needed.

Because of the overhead of switching between different rendering contexts, it is best to keep as much data as possible in different areas of the same render texture context, and copy data from one area of the texture to another in order (copying data over itself, however, must be avoided, as it would lead to resource conflicts).

Subtraction and Thresholding

Regardless of whether the laser pointer or light pen setup is being used, the data from the video frame is immediately transformed to create a rough approximation of point indicated by the user. In any either case, that point should show up as a red blob. Depending on the angle of reflection, the mark left by a laser pointer on a monitor may be fairly faint, and difficult to separate from the background noise. However, as we expect that the laser pointer will be the only large moving red object in the scene, we can threshold use the difference

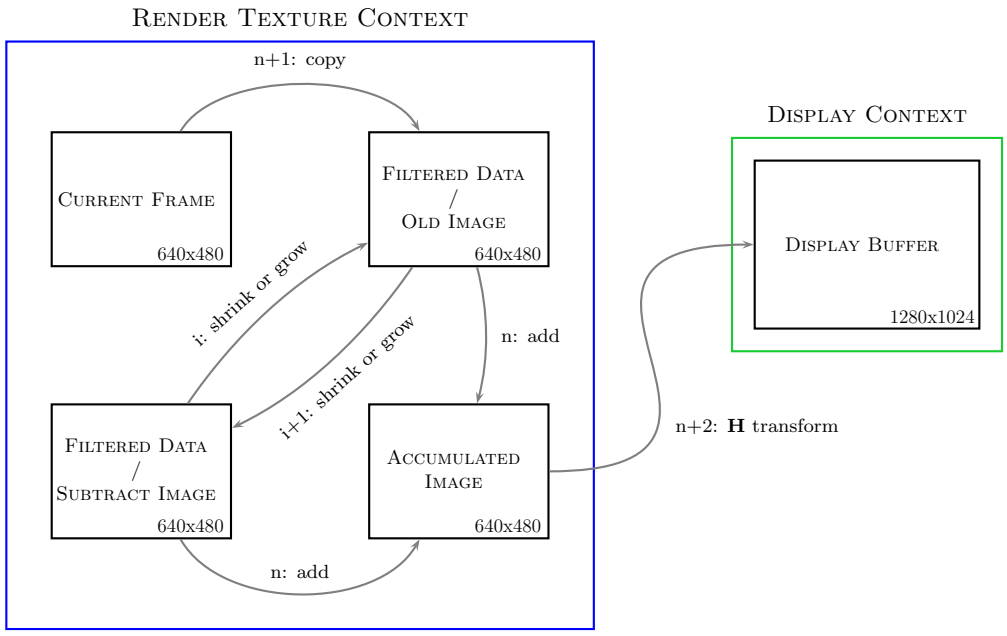
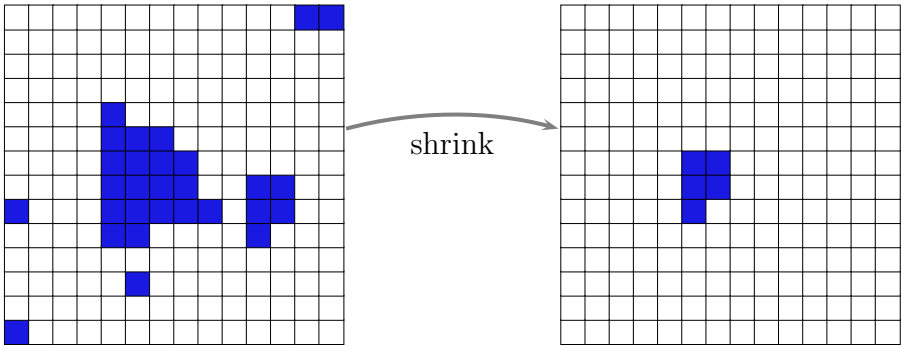


Figure 2.5: The fragment programs used to filter the laser pointer data, and display it on the screen. The shrink or grow operations are performed until the occlusion tests indicate that number of pixels lies within a given range. The programs used in the light pen case are nearly identical, except that the initial data does not come from a subtracted image, and data growing is never used.

of the current frame and the last frame, picking out the larger red values, and thus be left with a reasonable approximation of the laser point. The light pen, on the other hand, leaves a much brighter circular profile, and therefore a higher threshold can be applied directly to the captured frame, with little risk of losing the signal.

Refining the Data

This is were things get interesting. In the case of the light pen system, the threshold results typically leave us with a large circle centered on the touch position. If the data were on the CPU, we could find the expected position iteratively, but calculating the centroid of the points. However, moving the data to the CPU would be expensive, and a similar operation can be performed on the GPU itself by exploiting the occlusion testing features. Each pixel which passed the thresholding phase is treated as an active element of a cellular autometa, and an update rule applied that destroys any cell which is not surrounded by active cells.




```
float3 shrink(   float2 pos : TEXCOORD0,
                 uniform samplerRECT tex
                 ) : COLOR {

    float3 v_00=texRECT( tex,pos+float2(0,0) );
    if(v_00.r==0) discard;
    else {
        float v_01=texRECT( tex,pos+float2(0,1) ).r;
        float v_10=texRECT( tex,pos+float2(1,0) ).r;
        float v_m0=texRECT( tex,pos+float2(-1,0) ).r;
        float v_0m=texRECT( tex,pos+float2(0,-1) ).r;

        if(      v_01==0 ||
        v_10==0 ||       v_m0==0 ||
                 v_0m==0 )
                 discard;
    }

    return v_00;
}
```

Figure 2.6: One of the fragment programs used for data filtering, examined in detail.

Using occlusion testing, we can keep track of the number of pixels that survive after each update, and keep applying the data shrinking rule until the number of surving pixels falls below a target maximum. In some ways, this is actually preferable to finding the centroid, as it tends to eliminate any noise in the origional thresholded pixels. [2]

In the case of the laser pointer, the situation is more challenging, as, depending on our

[2]These operations would certainly benefit from Z-Culling optimizations, as explained in [9], and if I ever get around to carefully profiling the code, I will certainly implement them.

luck with reflections, we may have ended up with either a very large or very small blob of red points which passed the thresholding step. Thus the program may decide to grow out the detected signal, rather than shrinking it, in the case that our occlusion testing indicates that a small but nontrivial number of points have passed through the threshold.

Chapter 3

Results

As supplements to this report, we have created a number of videos, which may be downloaded from the project webpage at `http://cs.northwestern.edu/~sco590/vision/project.html`

As demonstrated by the videos, the assembled back projection system is useable, though there is clearly room for improvement.

- Calibration is simple and effective.
- The total latency of the the system is such that electronic drawing with the light pen is possible, but only at a slow rate.
- The resolution and picture quality are low, but again, good enough to allow for careful interactive drawing.

The laser pointer is less useful for drawing, as it typically moves too fast for the tracking algorithms to keep up with it. A laser pointer reflected off a monitor is also harder to track than the diffuse light sources created by reflections from the frosted acetate used as the back projection surface.

3.1 Comparisons with previous systems

This project occupies and unusual niche – an attempt to build a high performance electronic whiteboard out of cheap consumer electronics. As such, it is difficult to directly compare it with similar systems. For example, the resolution which we have achieved appears to be better than that in the "Visual Panel" system presented in [15]; but this is to be expected, as

we are using more advanced hardware and solving a much simpler vision problem. Expensive electronic whiteboard systems such as [4] are built with special purpose hardware, and so, by rights they *should* perform better than our own system. Our calibration algorithms run more slowly than those in [14], but are also more robust, as they do not require the entire display surface to be viewed by the camera. The fragment shader techniques which we have used are not revolutionary, but they do represent novel variations the emerging themes of general purpose graphics processing.

3.2 Conclusions

The system which we have created is functional but crude. The accuracy could be helped by improving the filtering algorithms, and sufficiently effective filtering could lead to results with a resolution exceeding that of the source frames captured from the webcam. However, implementing such algorithms without increasing the latency in the system would be challenging. A particle filter would seem to be a particularly attractive approach, as it parallelizes well, and can implemented entirely on the graphics card. (The particle filter implementation presented in August of 2004 [13], was a preliminary system and had not been tested in a practical system such as this.)

The latency in the system is very close to that implied by camera and DirectShow overhead. Careful engineering could improve the latter, and better hardware the former. The present system, used without modification, is suitable for interaction tasks which demand less high resolution, low latency data than interactive drawing; such as manipulating components of a GUI, or gesture based control.

Appendix A

Personal Statements

A.1 My Views on Computer Vision

I started my undergraduate career studying philosophy and physics. It was only near the end of my degree that I switched my focus to applied science. I had become disheartened by the glacial pace of my chosen disciplines, and moving to computer science allowed me to become involved in a more active field. Of all the subdisciplines of computer science, computer vision is one of those that best satisfies my desire for a lively research environment. There are interesting and immediate applications, and a steady stream of novel algorithms are being developed to address them. The underlying computational technology is rapidly evolving, and for the next decade at least, researchers will be scrambling to keep up with it.

As a computer graphics specialist, I am particularly interested in the role that the rise of cheap parallel computing is likely to play in computer vision. Many vision problems can benefit from the features offered by graphics cards: limited SIMD instruction sets that act on large volumes of data. Developing algorithms designed to exploit such hardware remains an under-explored research area, and one sure to see increased attention as this type of parallel device becomes ever more common.

A.2 Course Feedback

I enjoyed this course - the lectures introduced a range of important algorithms and techniques, and the presentation was thorough. Before taking the class, I had already encountered many of these algorithms in passing, and the course gave me the opportunity to strengthen my knowledge of computer vision.

In some of the lectures I was relatively unfamiliar with the underlying math, and as a result became a bit lost. The homework only partially covered the scope of the presented material, and thus I never had cause to reexamine the mathematics that had escaped me. If the homework had covered more of the material presented in the lectures, I would have finished the course with a more complete understanding of the material.

That said - as taught the workload for this class is already fairly heavy. There is a tradeoff here: more homework ensures that students leave the course with a broader understanding of computer vision, but less homework opens up space for projects, which then have a chance of provoking interesting research. Though I like the emphasis on projects, the next time the course is taught, I would suggest adding at least one small homework assignment related to Bayesian analysis.

A.3 Acknowledgments

I would like to thank Holger Winnemöller for the conversation that lead to this project, and for his help with the Direct Show libraries. I would also like to thank Jack Tumblin for his advise on designing back projection systems. Finally, I would like thank Bob Adolf, both for his help researching consumer video cameras and for taking a night off to help me build the light pen and back projection system.

Bibliography

[1] Mark Apperley, Laurie McLeod, Masood Masoodian, Lance Paine, Malcolm Phillips, Bill Rogers, and Kirsten Thomson. Use of video shadow for small group interaction awareness on a large interactive display surface. In *CRPITS '18: Proceedings of the Fourth Australian user interface conference on User interfaces 2003*, pages 81–90, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[2] K. Cheng and K. Pulo. Direct interaction with large-scale display systems using infrared laser tracking devices. In *Proc. Australian Symposium on Information Visualisation, (invis.au'03)*, volume 24, pages 102–111. ACS, 2003.

[3] P.H. Dietz and D.L. Leigh. Diamondtouch: A multi-user touch technology. In *ACM Symposium on User Interface Software and Technology (UIST)*, pages 219–226, November 2001.

[4] Scott Elrod, Richard Bruce, Rich Gold, David Goldberg, Frank Halasz, William Janssen, David Lee, Kim McCall, Elin Pedersen, Ken Pier, John Tang, and Brent Welch. Liveboard: a large interactive display supporting group meetings, presentations, and remote collaboration. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 599–607, New York, NY, USA, 1992. ACM Press.

[5] K. Fatahalian, I. Buck, M. Houston, and P. Hanrahan. Gpubench: Evaluating gpu performance for numerical and scientific applications. In *GP^2, ACM Workshop on General Purpose Computing on Graphics Processors*, 2004.

[6] Joshua Flachsbart, David Franklin, and Kristian Hammond. Improving human computer interaction in a classroom environment using computer vision. In *IUI '00: Proceedings of the 5th international conference on Intelligent user interfaces*, pages 86–93, New York, NY, USA, 2000. ACM Press.

[7] James Fung. Computer vision on the gpu. In Matt Pharr, editor, *GPU Gems 2*, pages 649–665. Addison Wesley, 2005.

[8] Nolan Goodnight, Cliff Woolley, Gregory Lewin, David Luebke, and Greg Humphreys. A multigrid solver for boundary value problems using programmable graphics hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 102–111, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[9] Mark Harris and Ian Buck. Gpu flow-control idioms. In Matt Pharr, editor, *GPU Gems 2*, pages 547–555. Addison Wesley, 2005.

[10] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[11] D.R. Olsen Jr and T. Nielsen. Laser pointer interaction. In *ACM CHI'2001 Conference Proceedings: Human Factors in Computing Systems*, pages 11–22, 2001.

[12] Alexander Keller. Instant radiosity. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[13] Antonio S. Montemayor, Juan José Pantrigo, Ángel Sánchez, and Felipe Fernández. Particle filter on gpus for real-time tracking. In *SIGGRAPH Posters*, 2004.

[14] R. Raskar, J. van Baar, and J.X. Chai. A low-cost projector mosaic with fast registration. *Asian Conference on Computer Vision (ACCV)*, January 2002.

[15] Ying Wu. *Vision and Learning For Intelligent Human-Computer Interaction*. PhD thesis, University of Illinois at Urbana-Champaign, 2001.

[16] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11), 2000.

[17] Zhengyou Zhang and Li wei He. Whiteboard scanning: Get a high-res scan with an inexpensive camera, 2002.