# Mediated Reality using Computer Graphics Hardware for Computer Vision

James Fung, Felix Tang and Steve Mann
University of Toronto, Dept. of Electrical and Computer Engineering
10 King's College Road, Toronto, Canada,
{fungja, mann}@eecg.toronto.edu, tangf@eyetap.org

## Abstract

*Wearable, camera based, head–tracking systems use spatial image registration algorithms to align images taken as the wearer gazes around their environment. This allows for computer–generated information to appear to the user as though it was anchored in the real world. Often, these algorithms require creation of a multiscale Gaussian pyramid or repetitive re–projection of the images. Such operations, however, can be computationally expensive, and such head–tracking algorithms are desired to run in real–time on a body borne computer. In this paper, we present a method of using the 3D computer graphics hardware that is available in a typical wearable computer to accelerate the repetitive image projections required in many computer vision algorithms. We apply this "graphics for vision" technique to a wearable camera based head–tracking algorithm, implemented on a wearable computer with 3D graphics hardware. We perform an analysis of the acceleration achieved by applying graphics hardware to computer vision to create a Mediated Reality.*

## 1 Introduction

Motion estimation and image registration algorithms often use repetitive or iterative schemes to determine motion between images. At each iteration or repetition, such algorithms often require the image to be warped, and this warped image is then used as the input to the next stage of processing. Additionally, multi-scale Gaussian pyramids may need to be created, and image filtering and down sampling may be required. Since the image warp will typically not result in pixels being mapped exactly to other pixel locations, some methods of interpolation are required. The process of image warping can often be computationally intensive, and accurate filtering and interpolation techniques add to the computational complexity of the image warp. Many vision algorithms, however, are desired to run in real–time, and the time spent re–warping and filtering the images can make up a large portion of the calculations required on each frame.

Many current graphics cards incorporate a great deal of hardware specifically designed to achieve extremely fast real–time rendering of texture mapped polygons. Additionally, high–end graphics cards incorporate hardware designed for filtering and pixel interpolation to create accurate texture maps. Many modern graphics cards are capable of hardware bilinear filtering and anisotropic filtering (though the specification of anisotropic filtering varies greatly between different graphics cards). The process of displaying a texture mapped polygon is essentially the same as applying a projective coordinate transformation to an image. This suggests that it is possible to utilize the hardware of modern graphics cards to apply a projective coordinate transformation in hardware, rather than doing so in software. In particular, graphics cards are tuned to create perspective projections of planar surfaces. When an image is texture mapped onto this planar surface, the graphics hardware will project the image onto the surface, which is a type of image warping (under projection).

In order to achieve fast, real–time computer vision algorithms, specialized hardware has, in the past, been used. In [4], a general purpose array of FPGAs was used to achieve face recognition at camera frame rates. While similar hardware could be developed for image warping, such hardware is already incorporated into modern graphics cards. It is thus possible to use existing, low cost and easily available graphics cards to realize hardware acceleration. In this way, computer graphics hardware, which is most commonly used to project computer generated information into an image (image synthesis), is rather being used for the purpose of accelerating a computer vision algorithm (image analysis).

Proceedings of the 6th International Symposium on Wearable Computers (ISWC'02)
0-7695-1816-8/02 $17.00 © 2002 IEEE

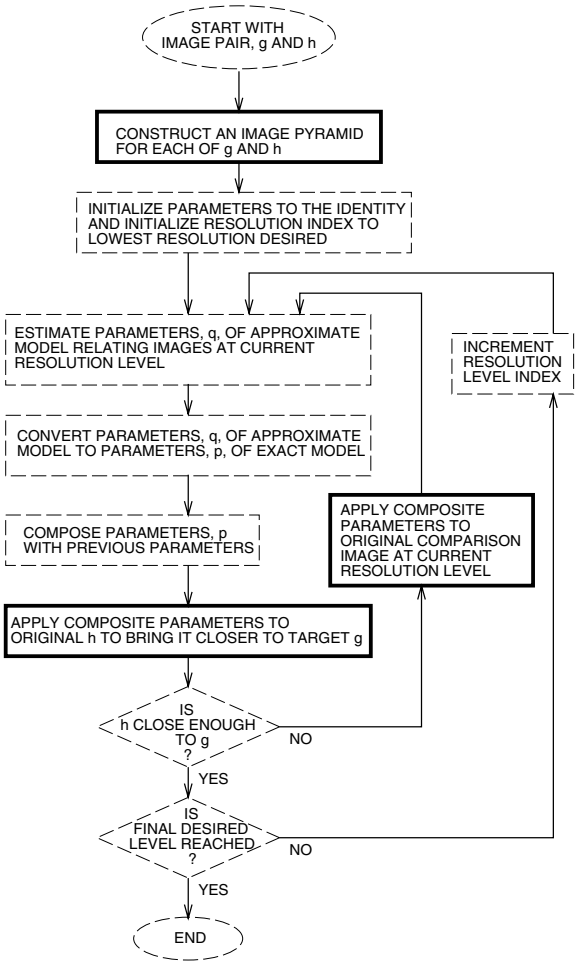


Figure 1: The VideoOrbits algorithm. The steps shown in solid dark lines have been accelerated using graphics hardware available on common 3D accelerated computer graphics cards.

## 2 Experimental Setup

In order to determine the advantages of applying hardware video acceleration for image warping, a Mesa3D program was written to access available video hardware. Mesa3D is a library which works identically to OpenGL [8]. Mesa3D is capable of performing perspective transformations of a plane [3] according to a projection matrix. Such an ability is particularly suited towards the VideoOrbits algorithm.

### 2.1 VideoOrbits

The VideoOrbits algorithm [7] considers transformations of planar patches as seen by a camera free to pan, tilt, and rotate about its optical axis. VideoOrbits is well suited for applying OpenGL video acceleration because VideoOrbits is a repetitive multiscale algorithm. Figure 1 shows the VideoOrbits algorithm. The steps of the algorithm which can be accelerated with graphics hardware are outlined in bold. From the figure, it can be seen that at each repetition, VideoOrbits attempts to estimate eight parameters of a projective coordinate transformation to spatially register two images. Then, VideoOrbits projects one image accordingly. This projected image is then used as an input to the next repetition of the algorithm. The projective coordinate transformation used in VideoOrbits maps straight lines to straight lines at all times, and thus represents a subset of all possible image warps. Thus we refer to image warping in VideoOrbits as projection rather than the more general term of image warping. In OpenGL, this is achieved by the viewing of a plane from different camera angles. In fact, the projective parameters calculated by VideoOrbits can be used as a camera transformation in OpenGL.

Figure 2 demonstrates VideoOrbits image processing. The top row of images shows the original images taken by a camera looking about a static scene. The second row of images shows each of the original images projected to spatially align with the leftmost image. These images are demonstrate the operations carried out in the highlighted algorithmic steps of figure 1. Additionally, the images have been comparametrically processed [6] to set their exposures equivalently. Note from the shape of the images that after the VideoOrbits projection, the images appear as projected quadrilaterals. In OpenGL, each of the images is a texture mapped plane, viewed at the appropriate angle. After each of the images has been properly projected, they can each be composited into a single image composite. The image in the third row shows the result of compositing the mulitple images of different exposure. This final image is thus of large spatial extent. Furthermore, comparametric statistical methods have also been applied to create an image of greater dynamic range than any of the original images. The statistical image enhancement methods applied to these images required floating pointing accuracy, which is effortlessly accomodated by OpenGL since OpenGL and graphics hardware can work natively with floating point representations of textures.

VideoOrbits has applications for performing camera based head tracking to create a wearable, tetherless mediated reality [5]. Essentially, the motion of a user's head as they look around a scene is very similar to the motion of a camera panning, rotating, and tilting about its optical axis. This is to say that motion of the camera is induced by motion of the head. VideoOrbits is well particularly suited to describe this prevalent motion in a camera based head tracking system. This is especially useful in Mediated Reality applications to make computer–generated information appear as though

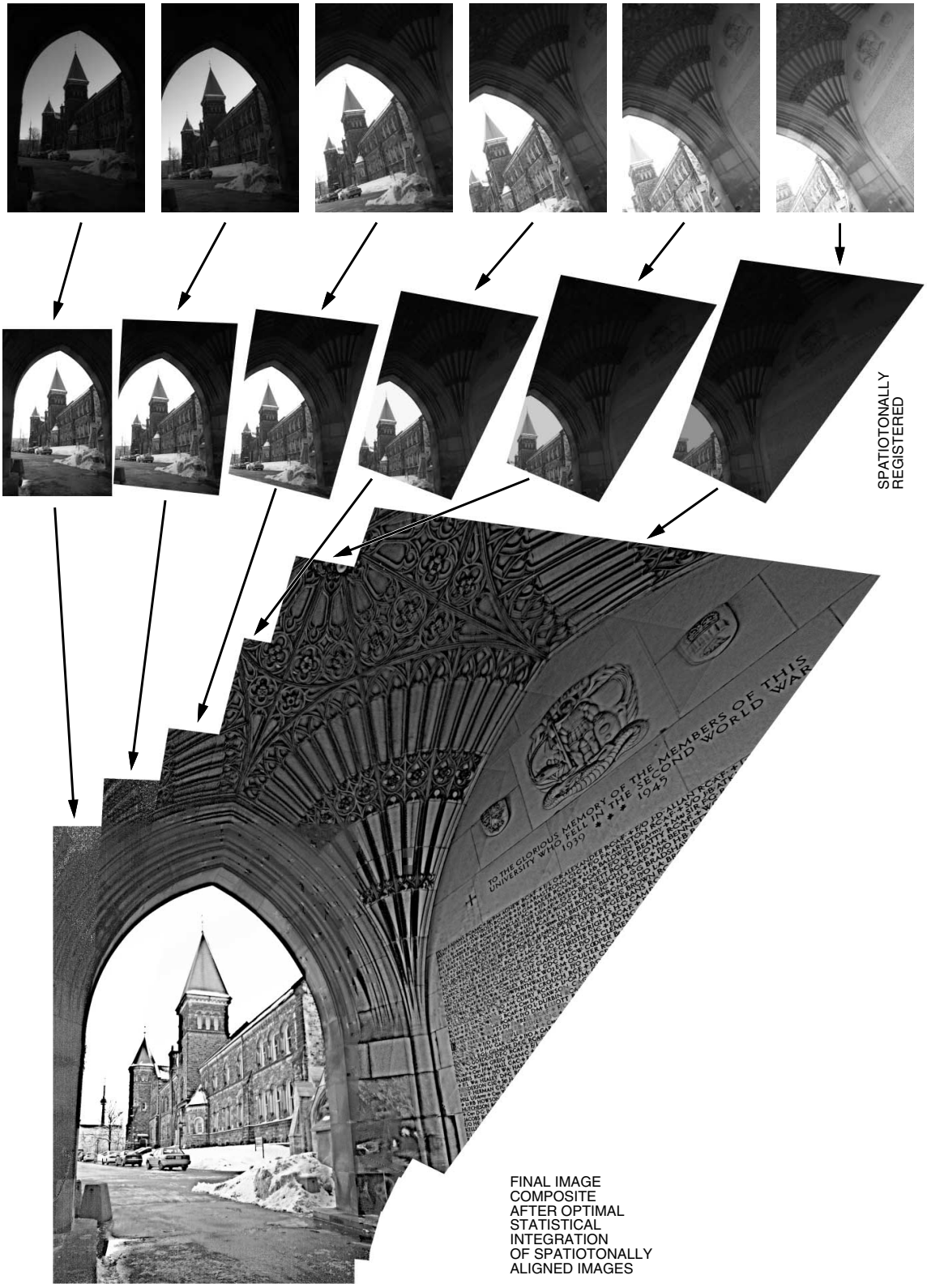


Figure 2: VideoOrbits example. Multiple images of the same subject matter are composited into a single high–dynamic range composite.

it was affixed to the real world scene as viewed through an HMD or EyeTap [5] devices.

VideoOrbits camera based headtracking has been implemented on a high–end server computer and runs at 11 frames per second. However, even faster processing is desirable for more accurate head–tracking, and often, wearable computers do not have as much computational power as their desktop counterparts. Currently the algorithm can lose tracking when large motions occur (such as the user moving their head quickly). Faster tracking causes large head motion to be captured by more frames, thus in each pair of frames, the motion appears smaller. This in turn results in greater accuracy for the VideoOrbits algorithm. Furthermore, by accelerating the image projection, more repetitions of the algorithm can be performed on each subsequent image, which in turn yields greater accuracy as well.

To investigate the effect of graphics hardware acceleration, the OpenGL program was designed to accept projective parameters from the VideoOrbits algorithm, allowing the OpenGL program to be used with VideoOrbits. In this investigation, the speedup achieved with graphics hardware was examined. This was done by comparing the speed of the software image projection algorithm to the OpenGL image projection program.

## 2.2 Mapping Projective Coordinate Transformations

In OpenGL, the most straightforward way of applying the projective coordinate transformation of VideoOrbits is to consider it to be a transformation to be applied to the projection matrix used in OpenGL.

The operation of applying a projective coordinate transformation to an image is homomorphic [1] to the process of projecting a texture mapped polygon under perspective projection in OpenGL. Thus, hardware acceleration of VideoOrbits projective transformations can be achieved by defining an homomorphism between the projective space of VideoOrbits and the projective space and homogeneous coordinate system of OpenGL. An homomorphism $\phi$ is defined by a mapping of VideoOrbits projective transformations $G$ to OpenGL projection matrices $M$:

$$\phi : G \rightarrow M \tag{1}$$

In the VideoOrbits algorithm, the projective coordinate transformation (PCT) is written as:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \frac{\mathbf{A}\mathbf{x} + \mathbf{b}}{\mathbf{c}^T\mathbf{x} + 1} \tag{2}$$

$$= \frac{\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}}{\begin{bmatrix} c_1 & c_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + 1} \tag{3}$$

Thus, it defines an eight parameter space. The transformation can be re-written as a $R_{3\times 3}$ matrix:

$$\begin{bmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ c_1 & c_2 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = G\mathbf{x} \tag{4}$$

where it can be seen that the set of projective coordinate transformation forms a group acting upon a set $S$ of image coordinates.

Thus, what is desired is some homomorphism $\phi$ mapping the projective coordinate transformation of VideoOrbits to a $R_{4\times 4}$ projection matrix in OpenGL.

The desired homomorphism is given by:

$$\phi(G) = \phi\left(\begin{bmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ c_1 & c_2 & 1 \end{bmatrix}\right) \tag{5}$$

$$= \begin{bmatrix} 1/a_{22} & -a_{21} & b_2 & 0 \\ -a_{12} & 1/a_{11} & b_1 & 0 \\ c_2 & c_1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

where it is now necessary to restrict $a_{11}, a_{22} \neq 0$. This however, is not a problem since $a_{11}$ and $a_{22}$ represent zoom in the $x$ and $y$ directions, and a zoom of 0 has little physical sense. This mapping takes into account the different coordinate systems and conventions used by each program. Equation 6 is used as a camera transformation matrix. Thus, it describes the transformation the camera undergoes, such that the plane will appear as required under OpenGL perspective projection.

To perform the image projection in OpenGL, then, the image is first loaded into the OpenGL program as a texture map. The four corners of the image are mapped onto the four vertices of a plane located at $z = 1$.

Then, the 'camera' in OpenGL is positioned to face this plane. This is achieved by using the `gluLookAt()` utility function.

Once the camera is facing the plane, a bounding frustum is defined to create a perspective projection. The function call `glFrustum()` creates a frustum which is used to map a the region which falls inside the frustum into a normalized device coordinate system (NDCS). The resulting framebuffer holds an image which should be close to (if not identical) to the VideoOrbits projection.

To conform exactly with the VideoOrbits programs, the OpenGL program must create a viewing window

[1] homomorphic refers precisely to algebraic homomorphisms, as discussed in [2]

which properly bounds the transformed image. In order to calculate where the bounds of this viewing window should lie, a normalized frustum projection matrix is created. Then, the desired projection matrix is applied to move the camera appropriately. The four corners of the plane can now be calculated by applying the new projection matrix to each of the four corners. These corners, given in the NDCS for the current frustum, give the maximum bounds for the projected plane. A new frustum, then, can be determined from these NDCS coordinates. The new frustum viewing angle remains the same, but the near plane geometry is determined by the desired bounding box. Using this frustum to create the image gives the properly projected and correctly bound image.

### 2.3 Direct Rendering in GNU/Linux

In GNU/Linux, the hardware is allowed to render directly into the framebuffer via the Direct Rendering Interface [1] (DRI). DRI allows the video chipset to bypass the function calls required by the graphical windowing system (X-Windows) to display graphics. DRI is required to achieve hardware video acceleration in GNU/Linux.

Once the plane has been projected, the results must be read back from the framebuffer into main memory. This process is known as "readback" [8], and is implemented by the OpenGL function call `glReadPixels()`. Because graphics cards are not necessarily designed for fast reads from the frame buffer to memory, the readback time must be included in the timing comparisons of hardware vs. software image projection as it is an important aspect of the hardware rendering scheme being used.

Since DRI renders directly to the framebuffer, the image will appear on the screen when rendered. Thus, the OpenGL hardware rendering program requires time to set up the rendering area, and create an X-Window. In an iterative/repetitive motion estimation algorithm, this initial overhead time would be incurred only once, and this setup time would not apply to the further estimations.

In order to investigate the effect of hardware on multiple iterations, the display setup time was not timed. The timing information for both the Mesa3D hardware and software programs timed the following:

1. Initialization of the texture mapping the texture onto a plane
2. Calculation of the bounding frustum
3. Positioning of the camera to apply the projective coordinate transformation on the plane and rendering of the scene
4. Reading of the pixels from the framebuffer back into memory

The time required to complete these steps is hereafter referred to as the projection times. To get a proper timing for the projection, it must be timed at some point after the image is known to have been rendered. The function call to `glReadPixels()` is used to read back the image from the framebuffer. It ensures that the image is rendered and this was verified by saving and displaying the buffer which was read back.

## 3 Results

To determine the speed–up attained by using hardware acceleration, a program using the hardware acceleration was compared with the software algorithm. A set of projective coordinate transformations was generated according to the equations of [9]. The programs were run on a wearable computer with a 700 MHz Pentium–III processor, with 64 MB of RAM. The wearable computer had an Intel i810 graphics chipset. Additionally, the programs were run an a more powerful desktop computer which had a Nvidia GeForce2 GPU to investiage the speedup achieved by more powerful graphics processing.

Figure 3 shows the results of timing a single projection using three methods:

1. a program using Mesa3D and available computer graphics hardware and DRI (ran on both the i810 and the GeForce2)
2. a program using Mesa3D, using software algorithms
3. a program running the equivalent VideoOrbits algorithm

Thus, an additional program is discussed here, which uses the software implementation of Mesa3D (actually the Mesa3D program is the same as the DRI program, with the direct rendering turned off). The software Mesa3D was examined because it is considered to be well optimized code for computer graphics applications. Thus, computer vision algorithms can also benefit from the speed and optimizations used in computer graphics software. So on machines which may not benefit from 3D graphics acceleration, Mesa will still implement an optimized software projection, and additionally this was examined.

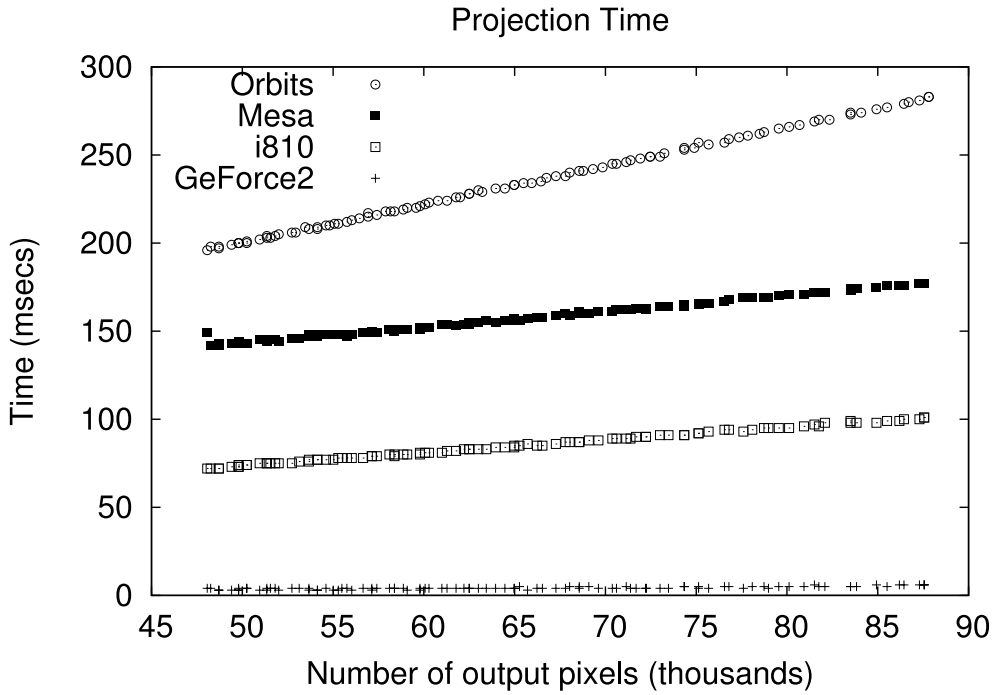


Figure 3: Projection times for VideoOrbits software, Mesa3D (using software rendering) and Mesa3D with DRI hardware rendering enabled.

For the plot of figure 3, the input image size was set, and different projection parameters were given to the three different programs, and the time taken to project the image was recorded. The projections used were independent rotations about each of the principle axis, with a maximum rotation of 15 degrees about any axis. From the data, the average speedup between VideoOrbits using DRI vs. using the CPU was $2.75\times$. The average speedup between VideoOrbits and the Mesa software rendering was $1.48\times$ and the average speedup between the Mesa software rendering and the DRI implementation was $1.83\times$. This speedup verifies that our DRI implementation did indeed use the available graphics hardware. For the Nvidia GeForce2 GPU, it was noted that the first iteration had an extra 10 msec overhead (for a 320x240 size texture), possibly due to a texture cache miss on the initial projection. Subsequent iterations took between 5-8 msecs.

Figure 4 shows the effect of i810 hardware acceleration on input images of different sizes, on the wearable computer. For this figure, the projection parameters were held constant, and the input image size was varied. In all cases, the hardware accelerated program projected the image faster than VideoOrbits. The smallest image size was $76 \times 58$ and the largest input image size was $435 \times 331$. The slope of a linear best fit line through the plot is $2.99$. Thus, for this range of input image sizes, the hardware speedup was $2.99\times$.

Figure 5 shows the effect of different projection parameters on the speed of the projection, on the wearable computer with the i810. For this plot, the input image was held constant, but the projection parameters were varied. The larger projection times shown correspond with increasing numbers of output pixels of the resulting image. Thus, this plot is measuring the effects of increased amounts of pixel interpolation, since large ouput images required more pixel interpolation since there were more output pixels. The slope of this graph was $4.07$. Slope here may be interpreted as how well each of the programs using DRI and the CPU dealt with more interpolation being required. Thus, the hardware was able to handle increased amounts of interpolation $4.07\times$ faster than the VideoOrbits software.

## 4 Conclusion

We have discussed the use of OpenGL to accelerate the VideoOrbits camera based head–tracking algorithm. VideoOrbits estimates the eight parameters of a projective coordinate transformation to spatially register two frames of video from a camera free to pan, tilt, and rotate about its optical axis. Because this coordinate transformation equivalently describes the projection of a rigid

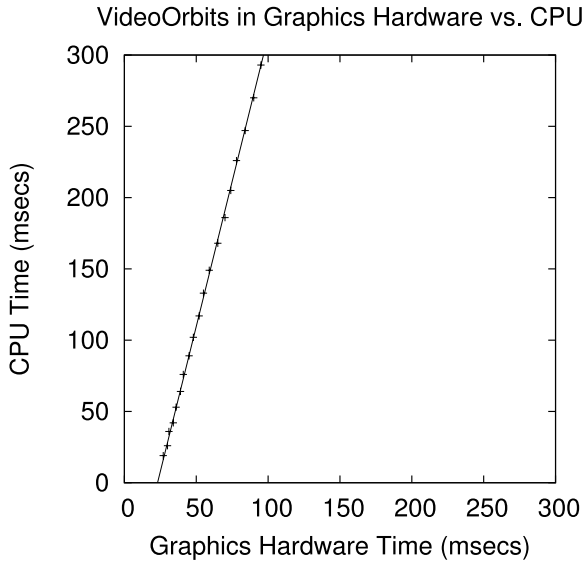


Figure 4: Projection times for VideoOrbits using graphics hardware vs. the CPU given varying image sizes. All images had identical projection parameters.

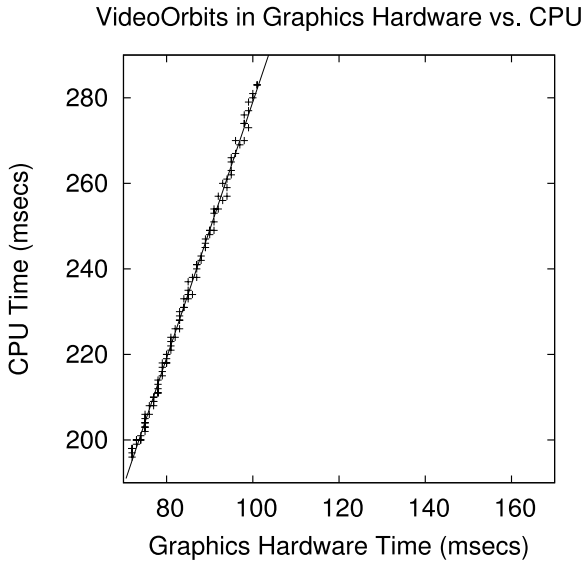


Figure 5: Projection times for VideoOrbits programs, one using graphics hardware and the other the CPU. The algorithm was given a fixed input image and the projection parameters were varied (resulting in larger output images).

planar patch as seen by a camera at an arbitrary angle, VideoOrbits is suitable for hardware acceleration using commonly available 3D computer graphics hardware. In each step of the parameter estimation, VideoOrbits projects the input frames. This projection is equivalent to the viewing of a texture mapped polygon in OpenGL. Thus, the rigid planar patch was treated as a polygon, and the image mapped onto the polygon as a texture. The resulting image, viewed under an appropriately defined perspective projection, was then read back from the graphics hardware, and was used by the VideoOrbits algorithm again. This hardware accelerated VideoOrbits algorithm was implemented on a wearable computer, and utilized the on–board 3D graphics chipset. From the timing measurements, it is clear that in all cases the hardware projection outperformed the software projection, generally speeding up image projection by a factor of $2.75\times$. The hardware benefit is greater with increasing amounts of pixel interpolation. This shows that the hardware can be effectively used to speed up repetitive image registration algorithms.

## Acknowledgements

We would like to acknowledge Chris Aimone and Corey Manders for their help.

## References

[1] Direct rendering interface. *http://dri.sourceforge.net*.

[2] M. Artin. *Algebra*. Prentice Hall, 1995.

[3] Foley, vanDam, Feiner, and Hughes. *Computer Graphics, PRINCIPLES AND PRACTICE*. THE SYSTEMS PROGRAMMING SERIES. Addison-Wesley, second edition, 1990.

[4] R. Herpers, G. Verghese, K. Derpanis, R. McReady, J. MacLean, A. Levin, D. Topalovic, L. Wood, A. Jepson, and J. Tsotsos. Detection and tracking of faces in real environments. In *Proceedings of the International Workshop on Recognition, Analysis and Tracking of Faces and Gestures in Real-Time Systems*, Corfu, Greece, 1999.

[5] S. Mann and J. Fung. Videoorbits on eye tap devices for deliberately diminished reality or altering the visual perception of rigid planar patches of a real world scene. In *Proceedings of International Symposium on Mixed Reality (ISMR2001)*, pages 48–55, March 14-15 2001.

[6] S. Mann, C. Manders, and J. Fung. Painting with looks: Photographic images from video using quantimetric processing. In *ACM Multimedia 2002 (to appear in)*, Juan Les Pins, France, December 1-6, 2002.

[7] S. Mann and R. W. Picard. Video orbits of the projective group; a simple approach to featureless estimation of parameters. TR 338, Massachusetts Institute of Technology, Cambridge, Massachusetts, See http://hi.eecg.toronto.edu/tip.html 1995. Also appears in IEEE Trans. Image Proc., Sept 1997, Vol. 6 No. 9, p. 1281–1295.

[8] J. Neider, T. Davis, and M. Woo. *OpenGL Programming Guide: The official guide to learning OpenGL*. Addison-Wesley, third edition, 1993.

[9] R. Y. Tsai and T. S. Huang. Estimating Three-Dimensional Motion Parameters of a Rigid Planar Patch I. *IEEE Trans. Accoust., Speech, and Sig. Proc.*, ASSP(29):1147–1152, December 1981.