Combatting Rendering Latency

Marc Olano, Jon Cohen, Mark Mine, Gary Bishop
Department of Computer Science, UNC Chapel Hill
{olano,cohenj,mine,gb}@cs.unc.edu

ABSTRACT

Latency or lag in an interactive graphics system is the delay between user input and displayed output. We have found latency and the apparent bobbing and swimming of objects that it produces to be a serious problem for head-mounted display (HMD) and augmented reality applications. At UNC, we have been investigating a number of ways to reduce latency; we present two of these. Slats is an experimental rendering system for our Pixel-Planes 5 graphics machine guaranteeing a constant single NTSC field of latency. This guaranteed response is especially important for predictive tracking. Just-in-time pixels is an attempt to compensate for rendering latency by rendering the pixels in a scanned display based on their position in the scan.

1 INTRODUCTION

1.1 What is latency?

Performance of current graphics systems is commonly measured in terms of the number of triangles rendered per second or in terms of the number of complete frames rendered per second. While these measures are useful, they don't tell the whole story.

Latency, which measures the start to finish time of an operation such as drawing a single image, is an often neglected measure of graphics performance. For some current modes of interaction, like manipulating a 3D object with a joystick, this measure of responsiveness may not be important. But for emerging modes of "natural" interaction, latency is a critical measure.

1.2 Why is it there?

All graphics systems must have some latency simply because it takes some time to compute an image. In addition, a system that can produce a new image every frame may (and often will) have more than one frame of latency. This is caused by the pipelining used to increase graphics performance. The classic problem with pipelining is that it provides increased throughput at a cost in latency. The computations required for a single frame are divided into stages and their execution is overlapped. This can expand the effective time available to work on that single frame since several frames are being computed at once. However, the latency is as long as the full time spent computing the frame in all of its stages.

1.3 Why is it bad?

Latency is a problem for head-mounted display (HMD) applications. The higher the total latency, the more the world seems to lag behind the user's head motions. The effect of this lag is a high viscosity world.

The effect of latency is even more noticeable with see-through HMDs. Such displays superimpose computer generated objects on the user's view of the physical world. The lag becomes obvious in this situation because the real world moves without lag, while the virtual objects shift in position during the lag time, catching up to their proper positions when the user stops moving. This "swimming" of the virtual objects not only detracts from the desired illusion of the objects' physical presence, but also hinders any effort to use this technology for real applications.

Most see-through HMD applications require a world without these "swimming" effects. If we hope to have applications present 3D instructions to guide the performance of "complex 3D tasks" [9], such as repairs to a photocopy machine or even a jet engine, the instructions must stay fixed to the machine in question. Current research into the use of see-through HMDs by obstetricians to visualize 3D ultrasound data indicates the need for lower latency visualization systems [3]. The use of see-through HMDs for assisting surgical procedures is unthinkable until we make significant advances in the area of low latency graphics systems.

2 COMBATTING LATENCY

2.1 Matching

A possible solution to this lag problem is to use video techniques to cause the user's view of the real world to lag in synchronization with the virtual world. However, this only works while the latency is relatively small.

2.2 Prediction

Another solution to the latency problem is to predict where the user's head will be when the image is finally displayed [10, 1, 2]. This technique, called predictive tracking, involves using both recent tracking data and accurate knowledge of the system's total latency to make a best guess at the position and orientation of the user's head when the image is displayed inside the HMD. Azuma states that for prediction to work effectively, the lag must be small and consistent. In fact he uses the single field-time latency rendering system (Slats), which we will discuss shortly, to achieve accurate prediction.

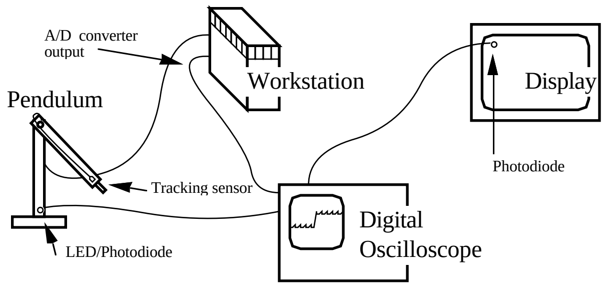

Figure 1: Apparatus for external measurement of tracking and display latency.

2.3 Rendering latency: compensation and reduction

2.3.1 Range of solutions

There are a wide spectrum of approaches that can be used to reduce lag in image generation or compensate for it. One way to compensate for image generation latency is to offset the display of the computed image based upon the latest available tracking data.

This technique is used, for example, by the Visual Display Research Tool (VDRT), a flight simulator developed at the Naval Training Systems Center in Orlando, Florida [5, 6]. VDRT is a helmet-mounted laser projection system which projects images onto a retro-reflective dome (instead of using the conventional mosaic of high resolution displays found in most flight simulators). In the VDRT system, images are first computed based upon the predicted position of the user's head at the time of image display. Immediately prior to image readout, the most recently available tracking data is used to compute the errors in the predicted head position used to generate the image. These errors are then used to offset the raster of the laser projector in pitch and yaw so that the image is projected at the angle for which it was computed. Rate signals are also calculated and are used to develop a time dependent correction signal which helps keep the projected image at the correct spatial orientation as the projector moves during the display field period.

Similarly, Regan and Pose are building the prototype for a hardware architecture called the address recalculation pipeline[15]. This system achieves a very small latency for head rotations by rendering a scene on the six faces of a cube. As a pixel is needed for display, appropriate memory locations from the rendered cube faces are read. A head rotation simply alters which memory is accessed, and thus contributes nothing to the latency. Head translation is handled by object-space subdivision and image composition. Objects are prioritized and re-rendered as necessary to accommodate translations of the user's head. The image may not always be correct if the rendering hardware cannot keep up, but the most important objects, which include the closest ones, should be rendered in time to keep their positions accurate.

Since pipelining can be a huge source of lag, latency can be reduced by reducing pipelining or basing it on smaller units of time like polygons or pixels instead of frames. Most commercial graphics systems are at least polygon pipelined. Whatever level the pipelining, a system that computes images frame by frame is by necessity saddled with at least a frame time of latency. Other methods overcome this by divorcing the image generation from the display update rate.

Frameless rendering[4] can be used to reduce latency in this way. In this technique pixels are updated continuously in a random pattern. This removes the dependence on frames and fields.

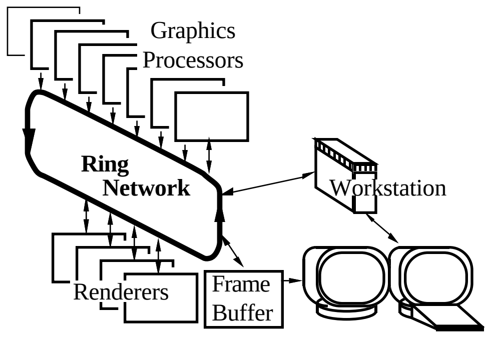

Figure 2: Pixel-Planes 5 system architecture

Pixels may be transformed at whatever rate is most convenient. This reduces latency at the cost of image clarity since only a portion of the pixels are updated. The transform rate can remain locked to the tracker update rate or separated on a pixel-by-pixel basis as with the just-in-time pixels method, discussed next.

2.3.2 Just-in-time pixels (JITP)

We will present a technique called just-in-time pixels, which deals with the placement of pixels on a scan-line display as a problem of temporal aliasing [14]. Although the display may take many milliseconds to refresh, the image we see on the display typically represents only a single instant in time. When we see an object in motion on the display, it appears distorted because we see the higher scan lines before we see the lower ones, making it seem as if the lower part of the object lags behind the upper part. Avoidance of this distortion entails generating every pixel the way it should appear at the exact time of its display. This can lead to a reduction in latency since neither the head position data, nor the output pixels are limited to increments of an entire frame time. This idea is of limited usefulness on current LCD HMDs with their sluggish response. However, it works quite well on the miniature CRT HMDs currently available and is also applicable to non-interactive video applications.

2.3.3 Slats

As a more conventional attack on latency, we have designed a rendering pipeline called Slats as a testbed for exploring fixed and low latency rendering [7]. Unlike just-in-time pixels, Slats still uses the single transform per frame paradigm. The rendering latency of Slats is exactly one field time (16.7 ms). This is perfect for predictive tracking which requires low and *predictable* latency. We measure this rendering latency from the time Slats begins transforming the data set into screen coordinates to the time the display devices begin to scan the pixel colors from the frame buffers onto the screens.

3 MEASURING LATENCY

We have made both external and internal measurements of the latency of the Pixel-Planes 5 PPHIGS graphics library [13, 7]. These have shown the image generation latency to be between 54 and 57 ms for minimal data sets. The internal measurement methods are quite specific to the PPHIGS library. However, the external measurements can be taken for any graphics system.

The external latency measurement apparatus records three timing signals on a digital oscilloscope (see figure 1). A pendulum and led/photodiode pair provide the reference time for a real-world event — the low point of the pendulum's arc. A tracker on the pendulum is fed into the graphics system. The graphics system

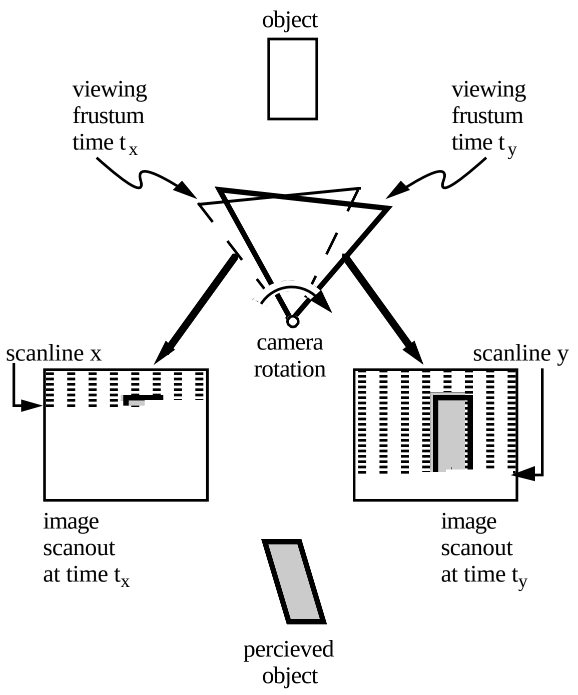
object
viewing frustum time t_x
viewing frustum time t_y
camera rotation
scanline x
scanline y
image scanout at time t_x
image scanout at time t_y
percieved object

Figure 3: Image generation in conventional computer graphics animation. Scanline x is displayed at time t_x, scanline y is displayed at time t_y.

starts a new frame when it detects the pendulum's low point from the tracking data. An D/A converter is used to tell the oscilloscope when the new frame has started. Frames alternate dark and light and a photodiode attached to the screen is used to tell when the image changes. The tracking latency was the time between the signal from the pendulum's photodiode and the rendering start signal out of the D/A converter. The rendering latency was the time between the signal out of the D/A converter and the signal from the photodiode attached to the screen. These time stamps were averaged over a number of frames.

The internal measurements found the same range of rendering latencies. The test was set up to be as fair as possible given the Pixel-Planes 5 architecture (figure 2, explained in more detail later). The test involved one full screen triangle for each graphics processor. This ensured that every graphics processor would have work to do and would have rendering instructions to send to every renderer. The first several frames were discarded to make sure the pipeline was full. Finally, latency determined from time stamps on the graphics processors was averaged over a number of frames.

4 JUST-IN-TIME PIXELS

4.1 The idea

When using a raster display device, the pixels that make up an image are not displayed all at once but are spread out over time. In a conventional graphics system generating NTSC video, for example, the pixels at the bottom of the screen are displayed almost 17 ms after those at the top. Matters are further aggravated when using NTSC video by the fact that not all of the lines of an NTSC image are displayed in one raster scan but are in fact interlaced across two fields. In the first field only the odd lines in an image are displayed, and in the second field only the even.

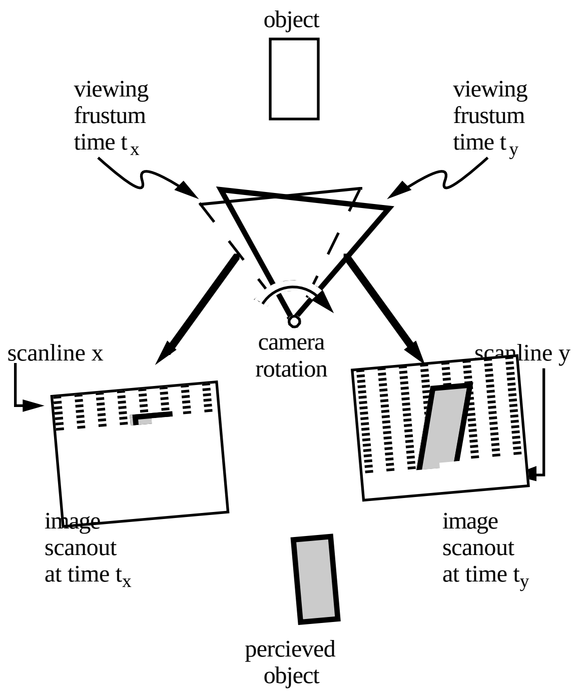
object
viewing frustum time t_x
viewing frustum time t_y
camera rotation
scanline x
scanline y
image scanout at time t_x
image scanout at time t_y
percieved object

Figure 4: Image generation using just-in-time pixels

Thus, unless animation is performed on fields (i.e. generating a separate image for each field), the last pixel in an image is displayed more than 33 ms after the first. The problem with this sequential readout of image data, is that it is not reflected in the manner in which the image is computed.

Typically, in conventional computer graphics animation, only a single viewing transform is used in generating the image data for an entire frame. Each frame represents a point sample in time which is inconsistent with the way in which it is displayed. As a result, as shown in figure 3 and plate 1, the image does not truly reflect the position of objects (relative to the view point of the camera) at the time of display of each pixel.

A quick "back of the envelope" calculation can demonstrate the magnitude of the errors that result if this display system delay is ignored. Assuming, for example, a camera rotation of 200 degrees/second (a reasonable value when compared with peak velocities of 370 degrees/second during typical head motion - see [12]) we find:

Assume:

1) 200 degrees/sec camera rotation
2) camera generating a 60 degree Field of View (FOV) image
3) NTSC video
   60 fields/sec NTSC video
   ~600 pixels/FOV horizontal resolution

We obtain:

$$200\frac{\text{degrees}}{\text{sec}}\times\frac{1}{60}\frac{\text{sec}}{\text{fields}}=3.3\frac{\text{degrees}}{\text{field}}\ \text{camera rotation}$$

Thus in a 60 degree FOV image when using NTSC video:

$$3.3\ \text{degrees}\times\frac{1}{60}\frac{\text{FOV}}{\text{degrees}}\times600\frac{\text{pixels}}{\text{FOV}}=33\ \text{pixels error}$$

Thus with camera rotation of approximately 200 degrees/second, registration errors of more than 30 pixels (for NTSC video) can occur in one field time. The term registration is being used here to describe the correspondence between the displayed image and the placement of objects in the computer generated world.

Note that even though the above discussion concentrates on camera rotation, the argument is valid for any relative motion between the camera and virtual objects. Thus, even if the camera's view point is unchanging, objects moving relative to the camera will exhibit the same registration errors as above. The amount of error is dependent upon the velocity of the object relative to the camera's view direction. If object motion is combined with rotation the resulting errors are correspondingly worse.

The ideal way to generate an image, therefore, would be to recalculate for each pixel the position and orientation of the camera and the position and orientation of the scene's objects, based upon the time of display of that pixel. The resulting color and intensity generated for the pixel will be consistent with the pixel's time of display. Though objects moving relative to the camera would appear distorted when the frame is shown statically, the distorted JITP objects will actually appear undistorted when viewed on the raster display. As shown in figure 4 and plate 2, each pixel in an ideal just-in-time pixels renderer represents a sample of the virtual world that is consistent with the time of the pixel's display.

Computation of both the viewing matrix and object positions for each pixel is quite expensive. Acceptable approximations to just-in-time pixels can be obtained, however, with considerably less computation. One option is to use a single transformation per scan line. This relies on the changes being small during the short (approximately 65 µs) time for the line. Calculations show this to be a reasonable assumption, allowing on the order of 0.13 pixels error.

Another approximation is to use only two transformations per field, one for the first pixel and one for the last pixel. Object positions are linearly interpolated between these two.

4.3 JITP applied to latency

A partial test implementation has been constructed that renders images using the just-in-time pixels paradigm. This system is intended to be used in a see-through HMD to help reduce image generation latency. In a real-time JITP system, instead of computing pixel values based upon the predicted position and velocity of the virtual camera, each pixel is computed based upon the position and orientation of the user's head at the time of display of that pixel. Generation of a just-in-time pixel in real time, therefore, requires knowledge of when a pixel is going to be displayed and where the user is going to be looking at the time. This implies the continuous and parallel execution of the following two central functions:

1) Synchronization of image generation and image scanout
2) Determination of the position and orientation of the user's head at the time of display of each pixel

By synchronizing image generation and image scanout, the JITP renderer can make use of the details of how the pixels in an image are scanned out to determine when a particular pixel is to be displayed. By knowing what scanline the pixel is on, for example, and how fast the scanlines in an image are displayed, the JITP renderer can easily calculate the time of display of that pixel.

Determination of where the user is looking can be accomplished through use of a conventional head tracking system (magnetic or optical for example). Determination of where the user is looking *at the time of display* of a pixel requires the use of a predictive tracking scheme. This is due to the presence of delays between the sampling of the position and orientation of the user's head and the corresponding display of a pixel. Included in the end-to-end delays is the time to collect tracking data, image generation time and the delays due to image scanout.

In the current implementation, the calculations for each scanline are pushed as late as possible. Ideally data for each scanline is transferred to the frame buffer just before it is read out by the raster scan. This technique, known as beam racing, was first used in early flight simulators. By pushing the graphics calculation as late as possible, beam racing allows image generation delays to be combined with display system delays. The result is lower overall end-to-end delay which simplifies the task of predicting the future position and orientation of the user's head. Prediction also benefits from the fact that the delayed computation makes it possible to use the latest available tracking data in the generation of the predicted user view point.

5 SLATS

5.1 Brief Pixel-Planes 5 description

To understand how Slats works requires some knowledge of Pixel-Planes 5 [11]. Using Pixel-Planes 5 gave us total control over the graphics software, which was all developed in-house. Because our goal was to achieve lower latency by modifying the rendering pipeline, such low-level control was necessary.

Referring to figure 2, Pixel-Planes 5 uses parallelism at both the transformation and rasterization stages of the rendering process. Primitives are typically generated on a host workstation and sent via a ring network to a set of graphics processors (GPs), where they are stored in local display lists. The graphics processors traverse these display lists, transforming the primitives from object coordinates to screen coordinates and generating appropriate rendering commands. The graphics processors then send these commands over the ring to the renderers, which perform rasterization and shading. Each of which handles a 128x128 region of the screen. Finally, the renderers send the resulting pixel values to a frame buffer, which is synchronized with a video display for output.

5.2 PPHIGS pipeline

PPHIGS is the standard rendering library for Pixel-Planes 5. It is controlled by a software layer called Rendering Control [8]. The rendering process is broken into three main stages. In the transform stage, the GPs transform the primitives. In the render stage, the renderers scan convert and shade the primitives. If there are more regions on the screen than there are renderers, the first renderer to finish starts on the next screen region. Finally, in the copy stage, the pixel data is copied into the frame buffer. This is illustrated in figure 5.

In this timing diagram and the ones that follow, each line shows use of an independent hardware resource. So the GPs, renderers, and frame buffer can all be used simultaneously. However one stage on the GPs must be finished before the next can begin. Arrows show, for one frame of interest, the dependencies between the different resources. All other timings can (and probably will) change depending on the contents of the scene.

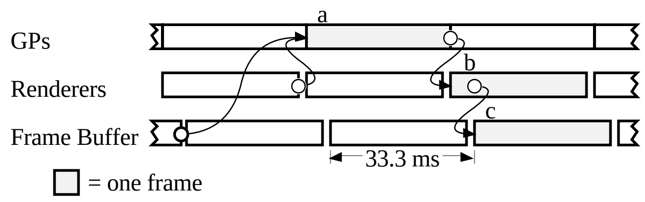

Figure 5: Basic PPHIGS timing for a frame passing through the pipeline. a, b, and c are the transform, render, and copy stages respectively for a single frame. The arrow between the middle of b and the start of c indicates that c can begin as soon as the first region is finished in b.

For stereo operation, PPHIGS handles first the left eye and then the right eye. However, both are considered part of a single unit. When the application software says to draw a frame, images for both eyes are drawn. This is illustrated in figure 6.

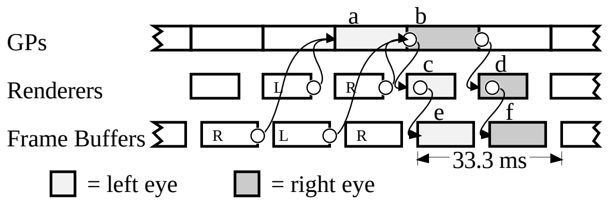

Figure 6: PPHIGS timing for a stereo pair passing through the pipeline. a, c, and e are the transform, render, and copy stages of the left eye. b, d, and f are the right eye.

As was mentioned earlier, the timings, other than those explicitly shown, can vary quite a bit. The lowest latency possible with PPHIGS occurs when the transform and render stages are small and the copy time is the limiting factor. In this case, the synchronization between the stages forces three fields of latency between the time the transformation begins and the time both eyes are complete and the images are displayed. This is illustrated in figure 7.

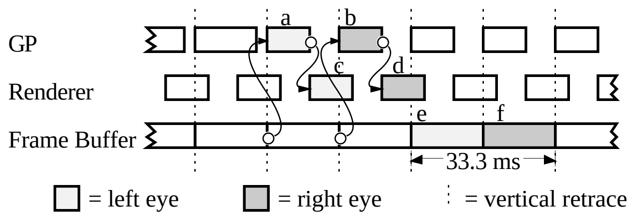

Figure 7: PPHIGS timing for a stereo pair with minimal latency. Render stage to copy stage dependencies are not shown for clarity.

5.3 Slats pipeline

Slats achieves its guaranteed latency by insisting that all the work for one field be finished during the field immediately before it. Since it is built with latency sensitive HMD applications in mind, it always generates stereo images. The pipelining in Slats is at a polygon level. As soon as a set of polygons are transformed (in clumps of 30 for ring network efficiency), they are sent to the renderers. Each renderer handles four screen regions so the entire screen for both eyes can be covered by the available renderers.

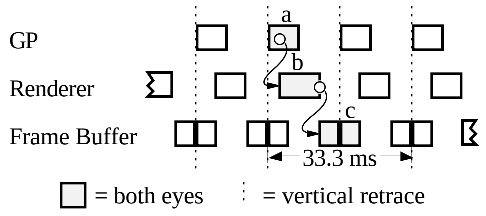

Figure 8: Slats timing for a stereo pair. a, b, and c are the transform, render, and copy stages respectively. Stage b starts after the first batch of triangles are transformed in a. The first half of c must finish before the vertical retrace.

Since a field is two regions high, the copy stage happens in two parts. The copy of the second half of the screen, which only takes 3.9 ms, doesn't occur until after the field is already being displayed. The copying of the first half of the screen must be done before the vertical retrace since those pixels are immediately displayed. This is illustrated in figure 8.

In many ways, Slats falls short of a general graphics library like PPHIGS. For the sake of simplicity, it uses only a single GP instead of the many (up to 50) available to PPHIGS. This severely limits the number of triangles that Slats can handle. The use of four regions per renderer makes polygon level pipelining easier, but also limits the shading model to simple Gouraud color interpolation.

All of the triangles must be transformed and rendered before the first copy begins, a period of about 12.8 ms. If there are too many primitives to make this deadline, Slats fails to generate a correct image. In the current implementation, this translates to about 100 triangles (or 12,000 triangles per second). Even if we optimized the code—and PPHIGS achieved about a factor of three performance increase after the triangle code was optimized to fit in the GP instruction cache—the communication bandwidth out of one GP and the speed of the renderers limits the maximum performance to about 250 triangles. We estimate that using multiple GPs and more renderers we might be able to push this to a few thousand, but currently don't have plans to follow this path.

These limitations are not flaws, Slats excels at what it is built for: experiments requiring low latency, fixed latency, or both. Azuma's work on predictive tracking [1] used Slats for just this reason.

Because it considers both eyes simultaneously, it can share more of the work than PPHIGS, which handles them sequentially but grouped. In fact, both eyes can be copied at the same time. Because it only renders the lines of the image visible in each field — the even lines are rendered while the odd field is visible, and the odd lines are rendered while the even lines are visible — it has half the rendering and half the copying.

As a comparison of the performance of both, figure 9 shows the pixel error for the setup used in our video. There is 33 ms of latency for the optical ceiling tracker[2], making a total of 90 ms for PPHIGS and 50 ms for Slats. Other trackers may have lower latency, but this will only increase the importance of image generation latency since the error is linear with respect to latency[1]. The error was calculated off-line with captured tracker data from a typical demo with a naive user under the optical ceiling tracker. The pixel error shown is computed by taking a point in the center of the field of view for each frame and

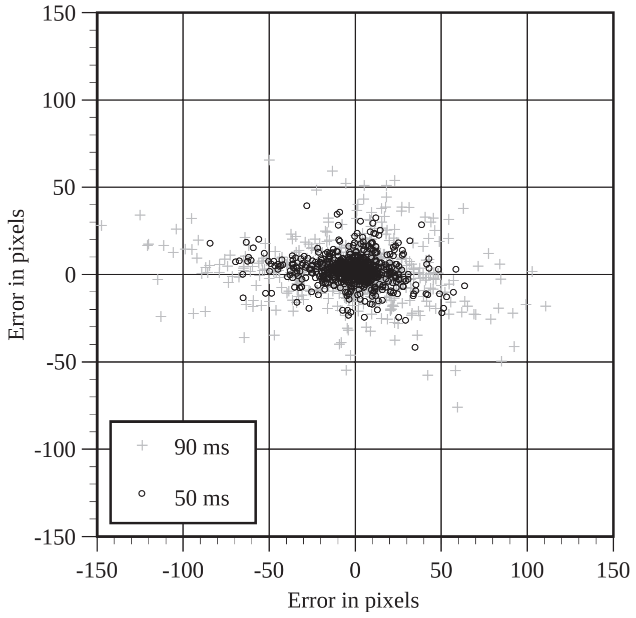

Figure 9: Pixels of error between a pixel at the center of the screen and the location where it should have been displayed by the time the frame was visible. For 90 ms, corresponding to PPHIGS + 33 ms tracker latency, and 50 ms, corresponding to Slats + 33 ms tracker latency.

determining how far from the center it would be when the frame is displayed.

6 CONCLUSION

We have presented two methods for reducing image generation latency. Both, necessarily, at a cost in polygon performance. As HMD applications become more prevalent, people will require minimal latency, much as they do high polygon rendering performance today.

7 ACKNOWLEDGMENTS

We would like to give special thanks to Ron Azuma. Ron is responsible figure 9, and was a huge help in the creation of the Slats video. We would also like to thank Tony Apodaca and Pixar for access to RenderMan, which was used to create the old well JITP simulation.

This project was funded in part by the National Science Foundation, NSF Grant Number MIP-9306208 and NSF Cooperative Agreement Number ASC 8920219, and by the Advanced Research Projects Agency, ARPA ISTO Order Number A410 and ARPA Contract DABT63-93-C-C048.

8 REFERENCES

1. Azuma, Ronald and Gary Bishop. Improving Static and Dynamic Registration in an Optical See-through HMD. Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994). In *Computer Graphics* Proceedings, Annual Conference Series, 1994. ACM SIGGRAPH, New York, 1994, pp. 197–204.

2. Azuma, Ronald. Predictive Tracking for Augmented Reality. UNC Chapel Hill Department of Computer Science PhD Dissertation, 1995.

3. Bajura, Michael, Henry Fuchs and Ryutarou Ohbuchi. Merging Virtual Objects with the Real World: Seeing Ultrasound Imagery within the Patient. Proceedings of SIGGRAPH '92 (Chicago, Illinois, July 26-31, 1992). In *Computer Graphics*, 26, 2 (July 1992), ACM SIGGRAPH, New York, 1992, pp. 203-210.

4. Bishop, Gary, Henry Fuchs, Leonard McMillan and Ellen Scher Zagier. Frameless Rendering: Double Buffering Considered Harmful. Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994). In *Computer Graphics* Proceedings, Annual Conference Series, 1994. ACM SIGGRAPH, New York, 1994, pp. 175–176.

5. Breglia, Denis, Michael Spooner and Dan Lobb. Helmet Mounted Laser Projector. Proceedings of the Image Generation/Display Conference II (Scottsdale, Arizona June 10–12, 1981). pp. 241–258.

6. Burbidge, Dick, Paul Murray. Hardware Improvements To The Helmet Mounted Projector On the Visual Display Research Tool (VDRT) At The Naval Training Systems Center. Proceedings of the SPIE conference on Head-Mounted Displays, 1989.

7. Cohen, Jon and Marc Olano. Low Latency Rendering on Pixel-Planes 5. UNC Chapel Hill Department of Computer Science technical report TR94-028, 1994.

8. David Ellsworth. Pixel-Planes 5 Rendering Control. UNC Chapel Hill Department of Computer Science Software Documentation, 1989.

9. Feiner, Steven, Blair MacIntyre and Dorée Seligmann. Knowledge-based Augmented Reality. *Communications of the ACM*, 36, 7, July 1993, pp. 52-62.

10. Friedmann, Martin, Thad Starner and Alex Pentland. Device Synchronization Using an Optimal Filter. Proceedings of 1992 Symposium on Interactive 3d Graphics (Cambridge, Massachusetts, March 29–April 1, 1992). Special issue of *Computer Graphics*, ACM SIGGRAPH, New York, 1992 pp. 57-62.

11. Fuchs, Henry, John Poulton, John Eyles, et al. Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories. Proceedings of SIGGRAPH '89 (Boston, MA, July 31–August 4, 1989). In *Computer Graphics*, 23, 3 (July 1989), ACM SIGGRAPH, New York, 1989, pp. 79–88.

12. List, Uwe Nonlinear Prediction of Head Movements for Helmet-Mounted Displays. Technical Paper AFHRL-TP-83-45, December 1983.

13. Mine, Mark. Characterization of End-to-End Delays in Head-Mounted Display Systems. UNC Chapel Hill Department of Computer Science technical report TR93-001, 1993.

14. Mine, Mark and Gary Bishop. Just-In-Time Pixels. UNC Chapel Hill Department of Computer Science technical report TR93-005, 1993.

15. Regan, Matthew and Ronald Pose. Priority Rendering with a Virtual Reality Address Recalculation Pipeline. Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994). In *Computer Graphics* Proceedings, Annual Conference Series, 1994. ACM SIGGRAPH, New York, 1994, pp. 155–162.

16. Ward, Mark, Ronald Azuma, Robert Bennett, Stefan Gottschalk and Henry Fuchs. A Demonstrated Optical Tracker with Scalable Work Area for Head Mounted Display Systems. Proceedings of 1992 Symposium on Interactive 3d Graphics (Cambridge, Massachusetts, March 29–April 1, 1992). Special issue of *Computer Graphics*, ACM SIGGRAPH, New York, 1992, pp. 43–52.

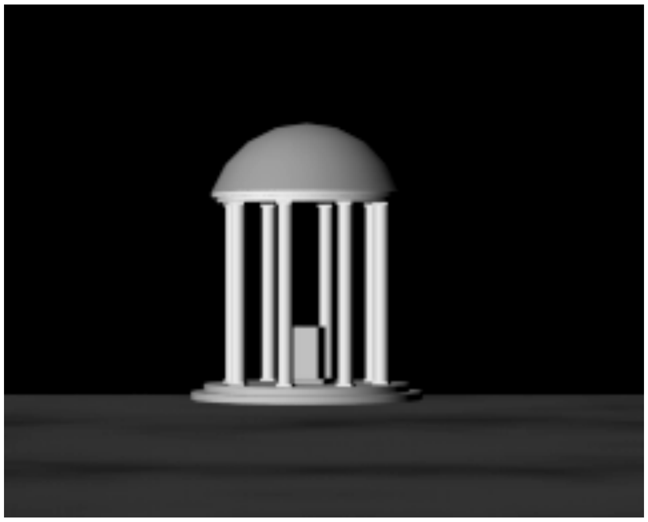

Plate 1: Image from an animation of UNC's Old Well, moving from left to right, rendered with conventional methods. On a scan line display, this appears slanted.

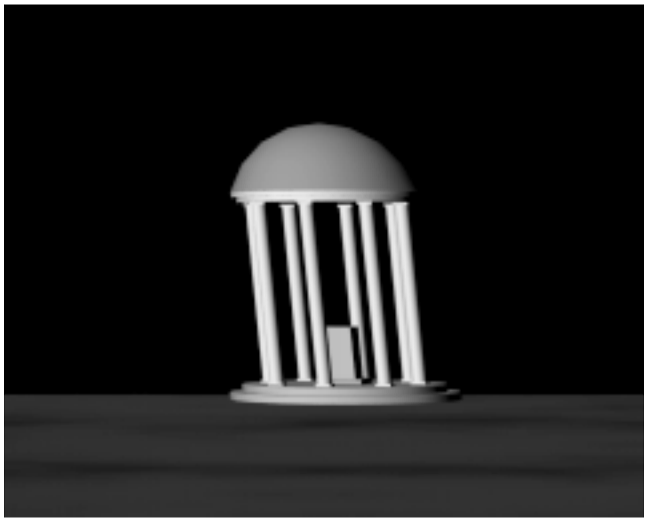

Plate 2: Image from an animation of UNC's Old Well, moving from left to right, rendered with the just-in-time pixels method. On a scan line display, this appears to be straight.