Introduction:

Despite the significant advances in real-time computer graphics performance via improvements in hardware and software algorithmic design, there are, and will continue to be data sets expanding beyond existing capabilities. Current double-buffering techniques are inadequate for addressing these large data sets. If all pixels must be computed into a new buffer before updating the display, very slow update rates may occur. In applications where the response rate to user input is crucial, a slow response with old information is completely unacceptable.

Consider a hardware design realizable in the near future: LCDs with programmable transistors at each pixel which can be accessed and updated individually. The pixels could, of course, be updated in a typical rasterization sequence: begin scanning at top left pixel and display line by line. Such a frame oriented scheme is the current norm, but will not be necessary with future computer graphics hardware technology.

Future 'frameless' hardware devices can erase the whole notion of double buffering. There will not be a compelling justification for delaying the display of a pixel's current information until all other pixel data for a frame are computed. If the constraint of using discrete frame units is eliminated, the pixel-at-a-time updates become seamless. Further, for pixel or subpixel level rendering strategies, such as ray tracing and supersampling for antialiasing, pixels can be displayed independently with their most current information. There is no reason to wait for a) all the pixels to be updated, nor b) all samples to be computed, before displaying current state.

It has already been illustrated in the 'frameless rendering' test [1] that utilization of the aforementioned hardware in a non-scanline ordered format can result in beneficial effects: quicker response time with more current information and a reasonable tradeoff in image quality. The purpose of this project is to illustrate, within a frame-based system, that a similar technique applied during the antialiasing stage can also achieve similar beneficial results.

Previous 'Frameless Rendering' Results:

We have already illustrated that the utilization of 'frameless rendering' [1], can offer a more fluid animation, which translates to quicker response time in interactive applications, with only a slight degradation of image quality. We simulated the proof of concept on an HP workstation by updating a random percentage of pixels at each time increment and comparing this with a double buffered animation with equal pixel budget. The double buffered frame updates occur only after all pixels are computed. For example, a 5 hz double buffered case is compared with a frameless rendering 15 hz update. In the latter case 33% of the pixels are updated every 67 milliseconds (ms) resulting in more fluid motion. A randomization without replacement strategy is used to choose this set of pixels. This eliminates the tearing artifacts associated with single buffering. The frameless rendered animation sequence has new, current information at each frame. For an equivalent pixel budget, the double buffered animation exhibits jerky motion and can only update with new information every 200 ms. Further it began computation on information that is already 200 ms old by the time the frame is displayed.

'Frameless Antialiasing':

Antialiasing is very computationally expensive. Good antialiasing with 16 samples per pixel can multiply compute time by an equivalent factor of 16 or higher. In applications requiring tight coupling of user input to system response, the slowdown is intolerable. In a frameless environment, though, there is no reason to resort to an all-or-none extreme. If there is time for x number of samples to be computed at time t, then all x samples should be computed and displayed.

I have to illustrated the potential for utilization of frameless antialiasing. Consider a typical double buffered animation, with full sampling per pixel, running at a relatively low frame rate. A buffer is not swapped until all samples are ready. The frameless antialiasing case will simulate pre-display of the ready samples by displaying current information for in-between frames. At each in-between frame I update the next 1/n samples (n=number of 'frameless' frames to one double buffered frame). The tradeoff of image quality is small compared to the benefit of the resulting smooth motion realizable in the frameless antialiasing test. Further, not only are more frames displayed per unit of time, but the information is more current at each time step.

Consider this example: Let's assume a supersampling grid of 4x4: 16 samples per pixel. For each frame in the frameless antialiasing case, only 25% of the samples are updated. If these frames are updated at 60 frames/second, the corresponding double buffering case with equivalent supersampling budget must wait for all 16 samples and will be updated at only 15 frames/second. (See Figure 1).

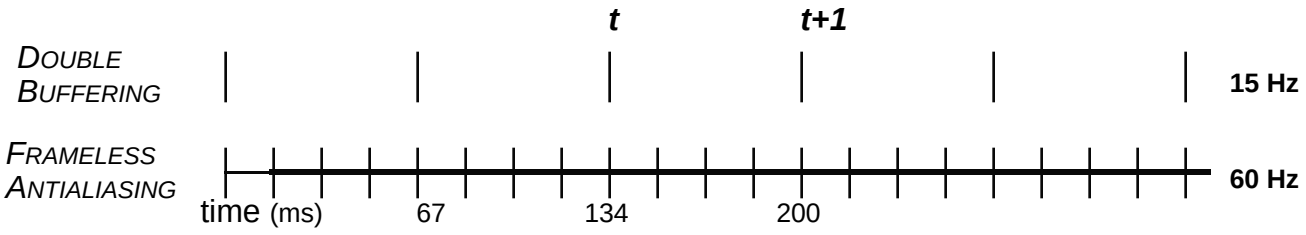

Figure 1: Update time increments for 'Frameless Antialiasing' vs. Double Buffering.

The double buffered computation begins at time t, but is not displayed until time $t+1$, so the image in the display at time t is $2t$ = 134 ms old before it is replaced. By contrast the 'frameless' animation is getting updates every 17 ms.

Antialiasing Strategy:

Antialiasing is a necessary step in the rendering pipeline in order to create realistic pictures. Although computer images are comprised of discrete pixels, this limitation should not be detectable in our images. The problem of aliasing or "jaggies", occurs because of high frequency information in the image - frequencies above twice the sampling rate. By sampling at a higher frequency (supersampling) and averaging those values to the spatial resolution of the final image, some of the aliasing artifacts are eliminated.

Supersampling techniques involve some slight variations on the following formula [3]:

1) Sample the continuous image at n (# samples per pixel) times the resolution of the final image to create your virtual image.

2) Apply a lowpass filter to the virtual image to eliminate the high

frequencies.

3) Reconstruct image from the virtual image by sampling at the resolution the final image will be displayed.

Using an existing rendering program, Rayshade [6], and rewriting portions of an existing display program, Flipbook (written by Leonard McMillan), and using a form of the above antialiasing scheme, I illustrated the concepts described in the introduction. Rayshade only permits antialiasing using sample choices of n = 1,2,3,4, or 5, corresponding to n*n samples per pixel. In order to accumulate samples to illustrate frameless antialiasing, I preferred to use samples on the full range of 1-16 per pixel. To achieve this, rather than rewriting Rayshade, I jittered the lookpoint according to a Poisson-disk distribution, and computed frames via the rendering program.

To compute the varied lookpoint values, the pixel size must be known relative to current field of view (FOV) and distance from eyepoint to lookpoint. (See Figure 2).

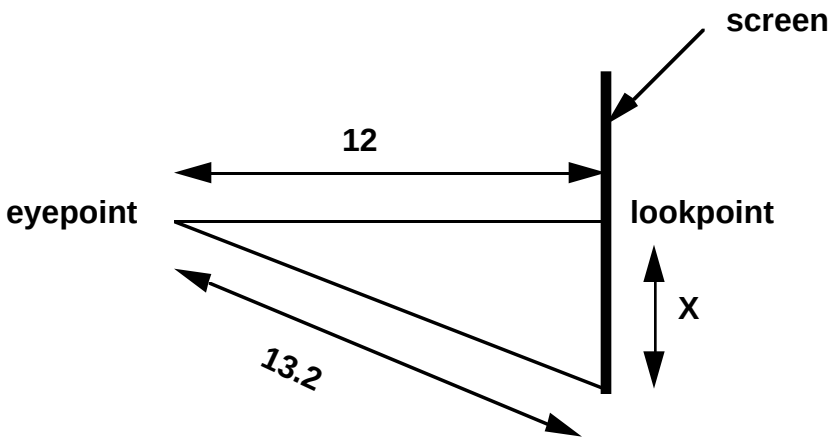

Figure 2: Computation used for sample points.

For FOV = 50 degrees:

sin 65 = cos 25 = 12/hypoteneuse (hyp) = .906; hyp = 13.2

sin 25 = cos 65 = X/12.2 = .423; X = 5.595; 2X = 11.19

Screen height is 2X = 11.19.
For resolution of 256x256:
pixel dimension is 11.19/256 x 11.19/256 <==> .044 x .044.

In Rayshade units, screen dimension is 11.19 x 11.19 units.
Each pixel covers .044 square units. (See Figure 3).

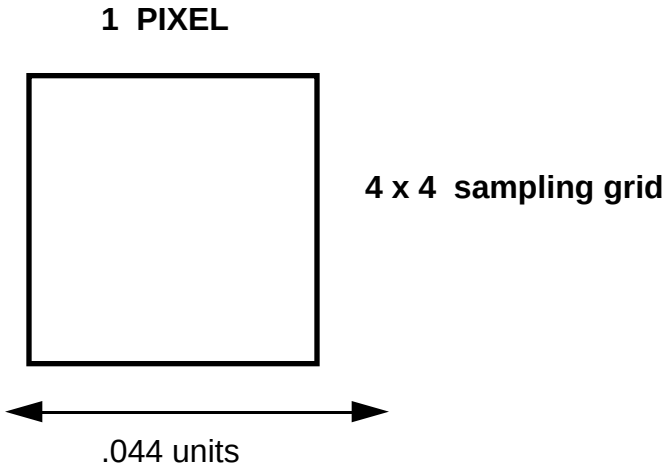

Figure 3. Sampling of one pixel for RayShade scene.

Distance for a uniform sampling: (See Figure 4)
Let x_delta, y_delta be distance between sample points.
x_delta = y_delta = pixel size/$\sqrt{\text{sample size}}$ = .044/4 = .011
delta = $\sqrt{x\_delta^2 + y\_delta^2}$ = .0155

Figure 4. Uniform sampling grid

Reasonable distance requirement for Poisson distribution:
final_delta = delta *.5 = .0078

I created a simple Poisson-like sampling per pixel. I varied the lookpoint only once so it should be noted that each pixel has the same distribution of samples with respect to its center. Only 16 samples/pixel are typically needed for good antialiasing. These 16 offsets were generated by a call to a randomization program which maintains the final_delta distance computed above.

Jittered antialiasing involves dividing a pixel into subregions [2], and choosing a sample point at a random position in each of these regions. This causes clumping of samples. Although the better Poisson-disk distribution is usually abandoned due to computational expense, using the same distribution for each pixel eliminates this factor. Although I used only 16 samples Poisson-distributed, I verified that the points were 'uniformly jittered' using a 100 sample point test. (See Figures 5 and 6).

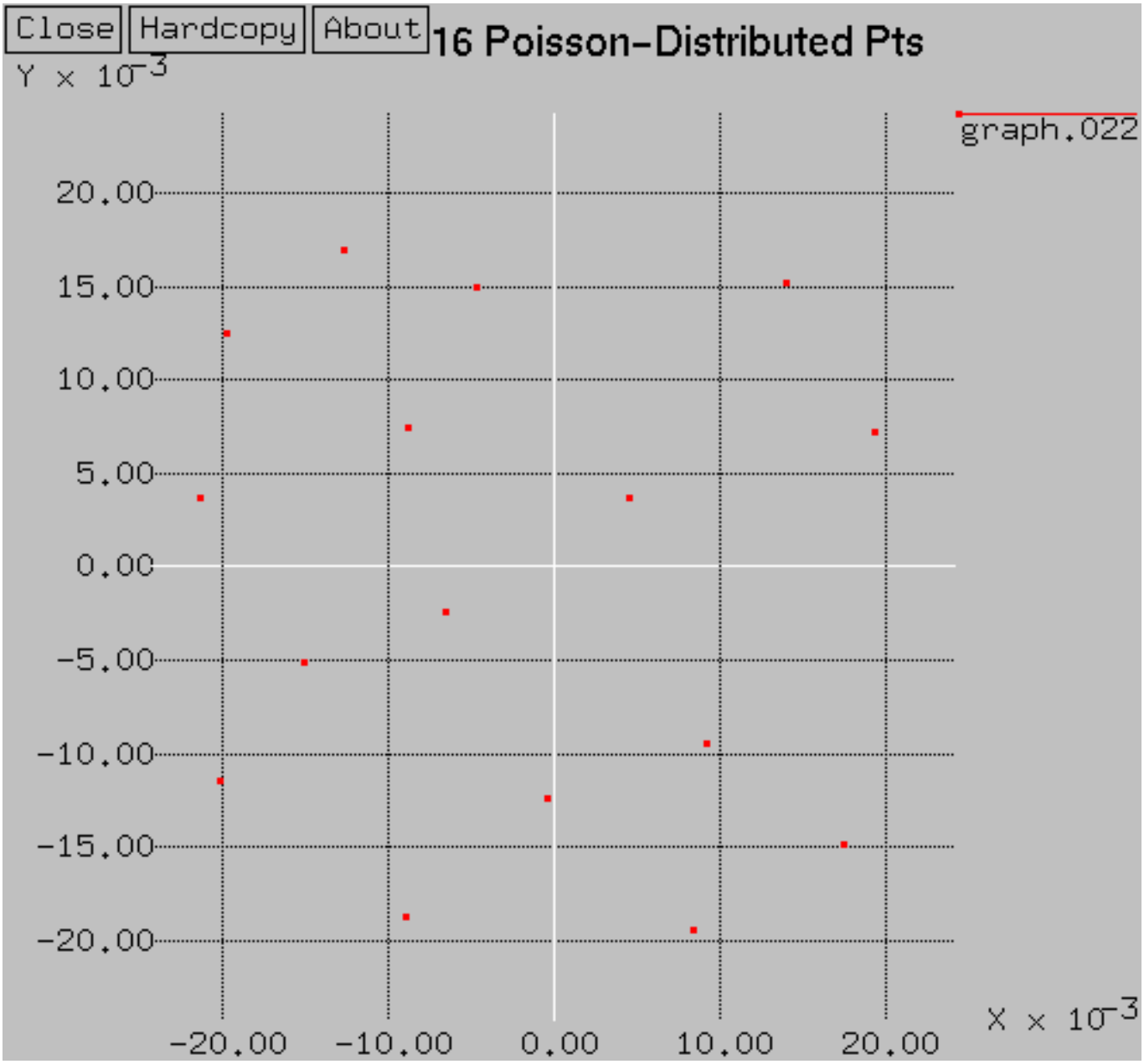

Figure 5. 16 sample points, Poisson-disk distribution. Interval: (-.022, .022)

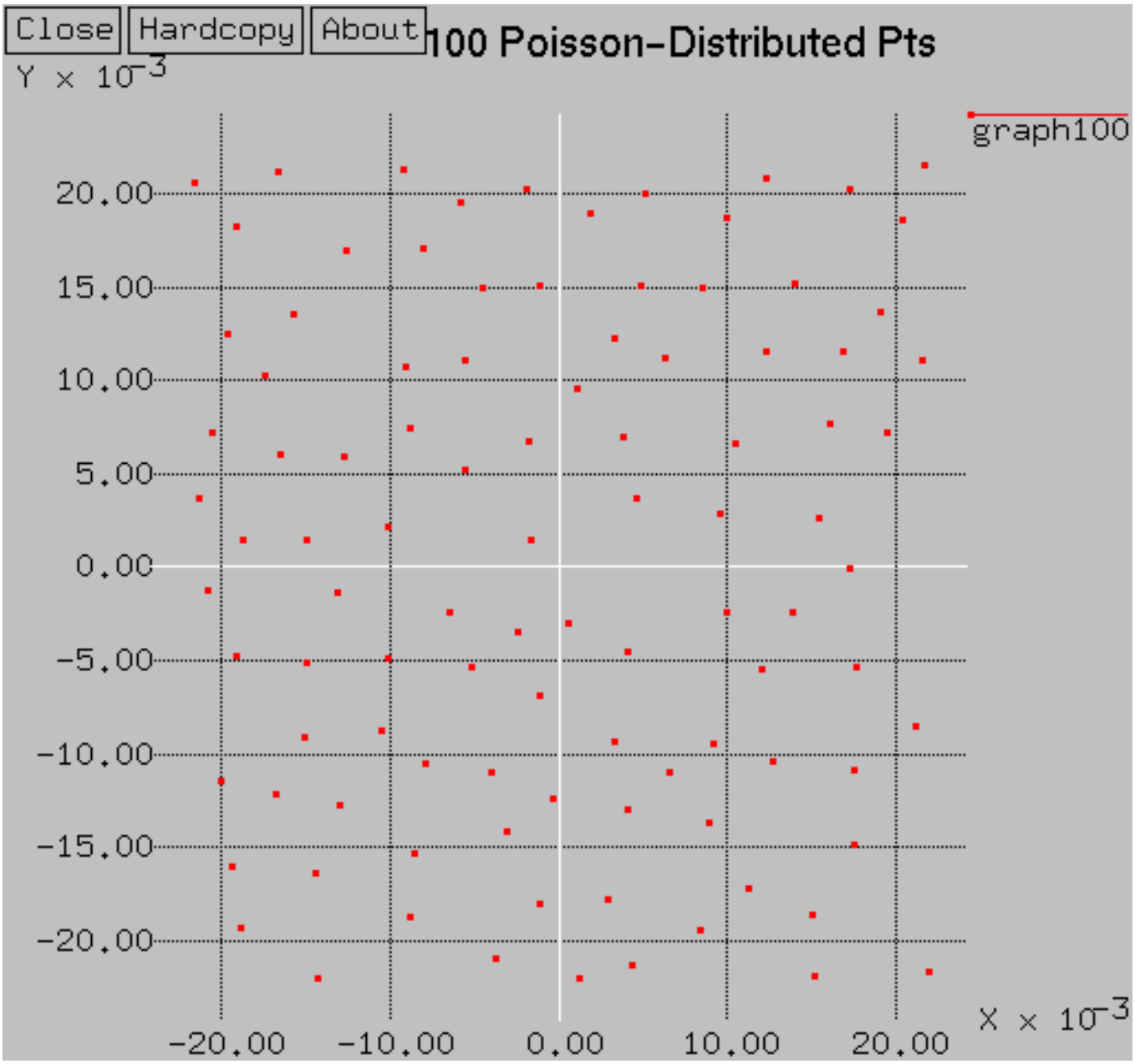

Figure 6. 100 sample points, Poisson-disk distribution. Interval: (-.022, .022)

Frameless Antialiasing Strategy:

The frameless antialiasing test is just a proof of concept simulation. This is alluded to in the overview in the 'Frameless Antialiasing' section of this paper. The simulation is executed on an HP workstation and actually uses double buffering for both the 'frameless' and double buffered animations. To simulate a machine capable of pixel level updates, we assume a maximum frame rate for the double buffered example of well below the frame rates capable on a workstation. That way the 'frameless' case can update in between this so called limitation, simulating an update at a non-frame interval.

Using a side-by-side comparison, with equivalent sample budget over time for both cases, the tradeoffs become immediately apparent. (See Figure 7).

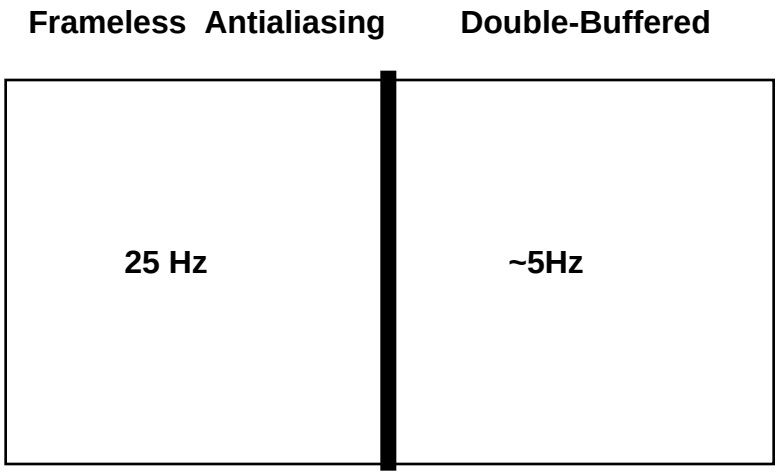

Figure 7. Side-by-side comparison setup.

Results and Conclusions:

The side-by-side comparison illustrates a clear advantage to updating samples as computed vs. waiting for all samples to be computed. For example, if 3 samples are updated at each frame, then 5 frames of animation are available in the same time only one of the double-buffered can be displayed. The smooth motion of only slightly degraded images is preferable in many applications to the jerky motion of fewer, but higher

quality frames. Further, automatic adaptive refinement occurs as all pixels are updated when the motion has stopped and computation time is no longer a limitation.

Using varied lookpoints as pixel samples, even with a constraint on distance between sample points (Poisson-distributed), did not achieve as good antialiasing effects as sampling techniques which compute a different set of samples at each pixel and force each sample to be within a certain region within the pixel. Further, experimenting with weighted sampling, where samples closer to the center of the pixel are weighted more heavily than those farther away, may approximate a truer intensity value at the pixel.

Future Work:

The logical next step is to combine frameless antialiasing with frameless rendering. This would be implemented on a first pass by updating a random percentage of pixels with a random set of samples per frame. This should be implemented using an improved antialiasing strategy (as suggested in the 'Results' section).

Pixels in our peripheral view may not require as high an update rate relative to those near the center of our viewing frustum. This information can be used in terms of choice of pixels to update and which to allot the greatest number of samples. That is, the simulation can be improved by implementing logical prioritization. This can be exploited in Head Mounted Display (HMD) applications. Further, in a HMD environment, when the user is moving his/her head rapidly, although update computation is a bottleneck, it is precisely during this motion that the user's real life view would be blurred anyway. This is just the time to decrease pixel or sample percentages for fast updating. This should be experimented with as well.

The ultimate goal, of course, is to implement frameless rendering and frameless antialiasing on a pixel addressable hardware. Hopefully incurring only a small performance hit, I may be able to use PixelFlow as the host machine.

References:

[1] Bishop, Fuchs, McMillan, Scher Zagier. Frameless Rendering: Double Buffering Considered Harmful. Proceedings of SIGGRAPH '94.

[2] Molnar, Steven. Efficient Supersampling Antialiasing for High-Performance Architectures; UNC Tech Report 91-023. April 1991

[3] Watt, Watt. Advanced Animation and Rendering Techniques: Theory and Practice. Addison-Wesley, ACM Press, 1993.

[4] Foley, van Dam, Feiner and Hughes. Computer Graphics: Principles and Practice. Addison-Wesley, 1990.

[5] IEEE Spectrum. The Flat Panel's Future. Kenneth I. Werner, Contributing Editor. November 1993.

[6] Rayshade User's Guide and Reference Manual. Craig E. Kolb. Draft 0.4 January 10, 1992.

[7] Mitchell, Don. Spectrally Optimal Sampling for Distribution Ray Tracing. Proceedings of SIGGRAPH '91.