Recent Advances in the NYU Autostereoscopic Display

Ken Perlin, Chris Poultney, Joel S. Kollin, Daniel T. Kristjansson, Salvatore Paxia

Media Research Lab
New York University - Courant Institute of Mathematical Sciences
719 Broadway, 12th Floor
New York, NY 10003
{perlin, crispy, jkollin, danielk, paxia}@cat.nyu.edu

ABSTRACT

The NYU Media Research Laboratory has developed a single-person, non-invasive, active autostereoscopic display with no mechanically moving parts that provides a realistic stereoscopic image over a large continuous viewing area and range of distance [Perlin]. We believe this to be the first such display in existence. The display uses eye tracking to determine the pitch and placement of a dynamic parallax barrier, but rather than using the even/odd interlace found in other parallax barrier systems, the NYU system uses wide vertical stripes both in the barrier structure and in the interlaced image. The system rapidly cycles through three different positional phases for every frame so that the stripes of the individual phases are not perceived by the user. By this combination of temporal and spatial multiplexing, we are able to deliver full screen resolution to each eye of an observer at any position within an angular volume of 20 degrees horizontally and vertically and over a distance range of 0.3-1.5 meters. We include a discussion of recent hardware and software improvements made in the second generation of the display. Hardware improvements have increased contrast, reduced flicker, improved eye tracking, and allowed the incorporation of OpenGL acceleration. Software improvements have increased frame rate, reduced latency and visual artifacts, and improved the robustness and accuracy of calibration. New directions for research are also discussed.

1. INTRODUCTION

Our single observer solid state dynamic parallax barrier autostereoscopic display [Perlin] achieved a number of key design goals: *(i)* no moving parts, *(ii)* full display resolution for both views, and *(iii)* complete freedom of movement for the observer in position, including distance from screen, as well as in orientation.

Our original autostereoscopic display prototype [Perlin] was designed to demonstrate the concepts for the display. Its purpose was mainly to show the feasibility of our solid state dynamic parallax barrier approach. While the display worked, in the sense that it embodied the concept effectively, it suffered from a number of deficiencies common to first proof-of-concept prototypes. These deficiencies, due to expedient engineering choices that allowed inexpensive and rapid engineering and construction of the prototype, included visible stripe artifacts, visible cross-talk between the left and right images, dimness of display, deficiencies in user tracking, and low frame rate.

Our recent work has focused on addressing these engineering deficiencies, and we have now demonstrated a display having a much closer to commercial grade quality and performance. This paper describes the engineering and algorithmic steps we pursued to achieve these increases in quality and performance, and reports on the results we achieved.

2. HARDWARE IMPROVEMENTS

For the original display prototype, we used a vintage 1994 Digital Light Processor (DLP) micro-mirror demonstrator projector from Texas Instruments [TexasInstr], because DLP projectors handle R,G,B sequentially. This allowed us to use color to encode the three time-sequential phases—albeit at the expense of being able to use color in our display. While this was advantageous in that the projector was designed so that the color wheel could be lifted out of the optical path of the system, they were several drawbacks to the older projector prototype. Aside from the gains in brightness and portability made in the past several years, the newer commercial DLP-based projectors can be connected directly to the video output of any standard video card (up to its resolution limit). The older projector required a card to be placed in the ISA bus of an (older) PC. This essentially precluded the use of a modern hardware OpenGL video card, and limited the update rate of the system as a whole to about 4.5 fps. Therefore we replaced the TI demonstrator with an [InFocus] LP330 video projector which allowed us to us an [ELSA] video card with hardware OpenGL acceleration. The new display system is shown in figure 1.

The LP330 is a bright projector for its size; but a major reason we used it for our second prototype was the relative ease with which it could be modified to run without the color wheel. The InFocus 330 uses a photocell to measure light passing the different phases of the rotating color wheel to provide feedback on the system timing for the projector as a whole; the projector will not work without the photocell signal. Thanks to the invaluable assistance of InFocus, we were able to replace the photocell feedback signal with the synch signal from the video input. The color wheel must still be allowed to spin freely so as not to trigger a safety shutdown, so we mounted it on the outside of the projector (figure 2). Because the timing of the color wheel is set up for two turns for every frame, the system actually performed better at an 85Hz frame refresh rate (non-interlaced) than at the 60Hz used in the original prototype. This meant that each of the three phases was being displayed in less than 1/255th of a second, as there is part of the color wheel cycle that cannot be used (described below).

Much of the hardware was similar to that used in the original prototype. We still used a Ferroelectric Liquid Crystal (FLC) element from [Displaytech] to shutter the start/stop time of each temporal phase. In addition to blocking out transitions between colors, modifications were necessary to block out the “white” part of the color wheel which is normally used to provide a boost to the luminance common to all colors. We did develop a fine control over the FLC shutter which could be addressed by modifying a 256-bit buffer that controlled whether the shutter was off or on for a given time slice of the frame.

For the light-blocking shutter, we had a custom pi-cell liquid crystal screen built to our specifications by [LXD], which we drove from power ICs mounted on custom-made Printed Circuit Boards (PCBs). The pi-cell was similar to that used in previous prototype, the main difference being in the removal of one polarizing sheet which allows us more flexibility in where we put the polarizer upstream of the pi-cell. We were also able to increase contrast by using improved analog circuitry. To control the sub-frame timings, we used a Field Programmable Gate Array (FPGA) from [Xilinx], the timing of which was controlled by 3 more 256-bit buffers. These were all driven from a Pentium III PC running Windows NT.

3. RENDERING IMPROVEMENTS

Numerous changes have been made to the rendering sub-system. The enabling factor for most of the changes was the switch to the InFocus projector, which allowed us to use video cards with OpenGL acceleration. The rendering software was entirely rewritten, replacing our custom software renderer with standard OpenGL calls.

Our first task after changing projectors was to choose the best OpenGL card for the job. The primary consideration in choosing a card was to find the one that would allow us to perform the image striping operations while maintaining the highest possible frame rate. We considered three algorithms for image striping.

The first algorithm, the “memory” algorithm, is based on the algorithm used in the original system: i) Render left and right eye views, ii) Copy images to buffers in CPU memory, iii) Copy image stripes into a third buffer in CPU memory, and iv) Copy the final buffer back to video memory. The second algorithm, the “stencil” algorithm, uses the stencil buffer, which is implemented in hardware on some OpenGL cards: i) Write the stripe pattern into the stencil buffer, ii) Render left eye view with stenciling for stripes and a red color mask, iii) Render right eye view, reversing the meaning of the stencil buffer, with a red color mask, and iv) Repeat steps i-iii with green and blue color masks. The third algorithm, the “copy” algorithm, makes use of the fast copying between buffers which is available on some OpenGL cards: i) Render the left eye view to a back buffer, ii) Copy left eye stripes, using red, green, and blue color masks in turn, to the main buffer, iii) Render the right eye view to a back buffer, and iv) Copy right eye stripes, using color masks, to the main buffer.

We evaluated a number of cards for features and speed, and were able to run a test suite on several of them. Our final choice was the ELSA Gloria II card, which showed impressive performance using the stencil algorithm. The original system, using software rendering and CPU-memory striping, rendered a 200-polygon model at 4.5 frames/second at a resolution of 300x300 (in a window). The Gloria II, using OpenGL acceleration and the “stencil” algorithm, is able to render a 4000-polygon model at 30+ frames per second at a resolution of 800x600 (full-screen).

Despite the improved frame rate, there was still a visible lag in the image update during continuous user movement, caused by the elapsed time involved in acquiring the user position and rendering the appropriate image. To compensate, we introduced a simple predictive filtering algorithm, which estimates what the user position will be in F frames based on a weighted average of the changes between the last N user positions. Some experimentation showed that values of F=1.0, N=6 yielded the best results, eliminating lag for all but the most jerky of user movements.

The switch to OpenGL also resulted in a much higher output of polygons, which also allowed us to incorporate more complex and realistic models. Using software built on the QvLib library, a C++ VRML parser from SGI, we wrote a quick interpreter for OpenInventor/VRML 1.0 files that converts Coordinate3, IndexedFaceSet, Translation, and Sphere nodes to a format the renderer understands. This allows us considerably more flexibility in the models we can display.

The original system used an algorithm for drawing stripes on the shutter that started calculating stripe locations at the left edge of the screen. The result was that as the user moved closer to and farther away from the display, changing the width of the stripes which were drawn, the pattern remained anchored at the left edge, with the change from frame to frame increasing toward the right edge of the screen. This resulted in some image breakup at the edge of the screen when the user moved along the Z axis. We chose instead to start the algorithm at the center of the display, continuing to calculate stripes out from the middle toward the edges. The result is that the stripe pattern varies least in the center of the screen (which is generally where the center of the model is located) and more toward the edges, with the variation at the edges reduced to half of that which occurred at the right edge when the striping was anchored at the left edge.

4. EYE TRACKING

4.1. Hardware and location of eyes

A reliable eye tracker is needed for the display to work, preferably an affordable one that does not consume too many cycles on the host PC. We chose the IBM BlueEyes tracker [Morimoto] as our starting point (figure 3). The principle is based on the "red-eye" effect noticeable when illuminating people with light emanating from a point close to the camera lens. The choroid layer of the retina diffusely reflects the light that reaches it, and the optics of the eye re-image (retroreflect) the light back to the source. Thus, only on-axis illumination returns to the camera in significant amounts, forming a "bright-pupil" image. As described in [Morimoto] the IBM system subtracts this bright-pupil image taken with on-axis illumination (a ring of infrared LEDs close to the camera lens) from a dark-pupil image taken with off-axis illumination (a surrounding ring of LEDs further away). This highlights the pupils of the eyes, while canceling out other bright spots in the image. Due to the optics of the eye, the retroreflection effect is stronger for light imaged close to the foveal region, thereby constraining us to place the tracker as close to where the user is looking as we can.

The original tracker IBM lent us used on-axis illumination on the odd scan lines and off-axis illumination on the even scan lines of one camera, and the reverse on a genlocked second camera. This is a clever way of getting only a 1/30 second delay instead of the 1/15 second delay one gets using frame-to-frame differencing with off-the-shelf NTSC cameras. Unfortunately this also acts as a horizontal edge detector since an even line is one pixel lower in the frame than its corresponding odd line. Frame differencing gives the fewest false positives for a slower moving user, while the field-to-field differencing is considerably better if the user is active. We changed the timing so that in the span of four fields, the two bright fields span an odd and an even field. This allows us to do frame differencing and field differencing with the same illumination pattern, so that we can switch "on-the-fly" to the one that gives the best results.

To use two cameras, we first tried using one camera's on-axis illuminator as the other's off-axis illuminator. This did not work for us, since we needed to mount the cameras close to where our display user was looking in order to get acceptable tracking. This meant mounting the cameras on either side of the display, and this large baseline made the light from one camera shade the scene very differently than the light from the other camera. Shadows cast by the ears were the prime offenders because they were often surprisingly difficult to distinguish from eyes. The next solution was the one shown at SIGGRAPH. It involved giving each camera its own on and off axis illuminator, as in single camera use, but using polarizing filters to separate one camera's illumination from the other's, which has the added advantage of not requiring the two cameras to be synchronized. We are still experimenting with this solution, but the linear polarizing filters we used did not separate the camera illumination consistently, and any polarizing filters cut the light by more than 50%. Unlike nocturnal animals, humans do not have highly reflective tapetums, so we need bright illumination for a discernible red eye effect. One of our reasons for using polarizers was to remove the glint off the cornea, which is just noise in our application. The polarizers were not as effective as we had hoped, but we achieved better results with our next solution: We replaced the polarizers with diffusers mounted in front of each light source, which cuts down on the sharpness of all shadows and highlights (glints) but does not change the red eye effect. This setup always has an off axis light turned on (from the other camera) but in practice works very well, at least with our genlocked cameras.

A pupil tracked with one camera must be grouped with its twin in the other view. Once the differently illuminated fields have been subtracted, isolating the eyes (and some movement- and reflection-based artifacts), the resulting data passes through several stages of analysis to extract the resultant eye positions. First, the difference image from each camera is analyzed for pairs of points that may be pairs of eyes. The pairs from each camera are then subjected to some simple 2-dimensional analysis to determine which pairs of left/right eye pairs from each camera might be images of the same pair of eyes. We do this by finding possible eye pairs in one image and finding the closest match in the other image. A combinatorial explosion lurks in this innocent statement, but we use adaptive thresholding based on the number of eye candidates found in the last frame to keep the number of eye candidates at low. This is necessary because at close range (10-12 inches) the shading on the face is significantly different for the on- and off-axis illuminators; but since the pupils are also bright in the on-axis illumination, a high threshold is good. At 4 ft the pupils are very dim, but there it no discernible shading difference between the on and off axis illumination, so a low threshold works well.

The final step is to use the lateral shift between the eye spots to determine the actual positions of the eyes in 3-space, as follows: Assume that we have the positions, in camera coordinates, of the spots representing the left eye in both the left and right cameras. Each 2-dimensional position in camera coordinates should represent a line of possible positions in 3-space that extends out from the camera lens. The actual eye position represented by that pair of camera coordinates would then be represented by the point in 3-space where the two lines cross. In practice, we found that there was some distortion in the cameras, and therefore chose to use splines instead of lines to compensate. Once the eye tracker is calibrated (see below), the correspondence between camera coordinates for each camera and 3-space coordinates at a range of distances from the display is known. Interpolation among the four closest known points at each of the known distances gives the approximate 3-space locations of the two camera points at each distance; we then draw interpolating splines through these points for each camera. To find the actual eye position, we then find the intersection of the two splines (figure 5).

4.2. Calibration

Calibration of the eye tracker turned out to be an interesting and challenging problem. To calibrate the cameras for the area in front of the display, we made a square target on which we printed a 1” square grid. We then placed the target vertically (in the XY plane) in front of the display and took a series of images of the target, one from each camera, moving the target 1” further from the screen after each pair of pictures. From these images, we would then be able to ascertain the position in 3-space of any camera coordinate at each of the target distances.

The first challenge was to find a target that would allow us to algorithmically find the point of the 1” grid. We first used a series of dots, printed in a grid pattern, but found it difficult to reliably extract the dot positions from the images. We then switched to a checkerboard pattern, printed in black and white, and concentrated on finding the saddle points where the squares meet. We perform the following steps at each point of each image to find the best saddle points (figure 4). The function c(x) returns the grayscale color at the point x.

1. For the point P being tested, find the points A, B, C, and D which lie N pixels away from P diagonally in each direction
2. Find the average grayscale color V_1 along the line AC, and V_2 along the line BD
3. Find $X = -(V_1 - c(P)) * (V_2 - c(P))$
4. Require each of the following:
    a. $X > T * c(P)$
    b. $(c(A) - c(P)) * (c(B) - c(P)) < 0$
    c. $(c(C) - c(P)) * (c(D) - c(P)) < 0$
    d. $(c(A) - c(P)) * (c(D) - c(P)) < 0$
    e. $(c(B) - c(P)) * (c(C) - c(P)) < 0$
    f. X is a local max within the U-pixel square in which it is centered

Rule 4a insures that the point being tested is a saddle point along both of the diagonal lines. It also insures that the signs of the differences between the color at P and the averages along the lines are different—in other words, the color values along the lines AC and BD approach the color at P from opposite ends of the grayscale continuum.

Rules 4b-4e insure that the colors at A and C are the same (either black or white), and that the colors at B and D are 1) the same, and 2) opposite, relative to the color at P, of the colors at A and C. Rule 4f prevents the finding of multiple saddle points at one intersection. We achieved the best results with a threshold value of $1.0 <= T <= 5.0$ and a square U of 7 pixels.

Once the saddle points have been located, the next step is to find the center of the target. The center of the target is marked with a ½" square of 50% gray centered in one of the white squares. To find the center, we start at the approximate center of the image (defined as the average of the saddle points). Moving out in a spiral pattern from the center of the image, at each pixel, we find the average G of the minimum and maximum colors along a Y-pixel vertical line centered at that pixel. The first pixel which lies at the center of an R-pixel square, each of whose pixels' colors differs from G by a factor of F or less, is assumed to be the center. Our best results were with a 3-pixel square and value of .1 for F, although we also found this part of the calibration to be particularly sensitive to lighting conditions.

When the center has been located, we can then determine the grid pattern described by the saddle points. We start by finding the four saddle points closest to the center in each diagonal direction. Once these four points have been found, we can start to extrapolate the expected positions of the next points along horizontal and vertical lines. The next point along the line is defined as the saddle point closest to the point extrapolated from the previous two points on the line, no more than K degrees off the line defined by the previous two points, and no more than a factor L of the distance between the previous two points from the last point. We can also weed out incorrect points by taking advantage of the fact that both of the quantities $V_1 - c(P)$ and $V_2 - c(P)$ defined above will reverse sign at every successive point along a line. If no point can be found that matches all the above tests, the line is terminated. Values of 30° for K and 1.67 for L were sufficient to find the correct grid pattern.

When the grid pattern analysis in complete, we write out a configuration file which describes the calibration—the correspondence between camera and 3-space coordinates for each camera at each plane in Z—to a file which can be read by the renderer.

There are occasional inaccuracies with the approach above that sometimes result in a 1- or 2-pixel inaccuracy in the location of a saddle point. This happens when the target illumination is either too bright or too dim, which causes some color bleeding and can make the true saddle point difficult to find. In the future, we may decide to use the saddle points found by the algorithm outlined above as a guide, then perform a more rigorous check in the vicinity of each point. This approach should allow us not only to eliminate the occasional inaccuracy, but also to achieve true sub-pixel resolution in the saddle point location.

4.3. Visualization

To test the accuracy of our calibration and spline-fitting algorithms, we wrote custom visualization software. This software has two major modes, one to display the output of the calibration, and one to help visualize the spline-fitting. The calibration visualization mode renders the camera coordinates of the target saddle points that were placed on the grid in each plane for one camera. By connecting coordinates in X or Y in one camera plane, or in Z across many planes, we can test the accuracy of our calibration algorithms—particularly the grid-drawing algorithm, which can produce subtle errors which are difficult to spot. Improvements in the calibration software have made the calibration visualization mode less necessary, though it remains quite useful.

The spline-fitting visualization mode (figure 5) has been crucial for debugging and fine-tuning the spline-fitting code, and has also proven to be the best method for testing the accuracy of the eye tracking output. Left- and right-camera splines are drawn for one or both eyes within a field of dots representing the area calibrated in front of the display. Splines can be drawn based either on live input from the eye tracker or on numbers chosen on a control panel. Any discontinuities or inaccuracies in the splines are immediately obvious, and a bar representing a confidence rating in the accuracy of the calculation is also displayed. When the spline drawing is based on live eye tracker input, any jump, lag, or inaccuracy is immediately obvious, making the spline visualization an invaluable aid in improving eye tracking accuracy.

5. FUTURE WORK

We are now constructing an alternate version of this display that will work in full color with modified stereo-ready CRT monitors. This requires a more sophisticated light-blocking shutter, since CRT monitors use a progressive scan, rather than displaying an entire image at once. This version will be in full color, since it will not rely on sequential R,G,B to produce time phases, and should have a fairly low production cost. This work will likely be the topic of our next paper. We anticipate a wide variety of applications for a CRT-based display in the medical, design, and scientific visualization areas, as well as an obvious market for gamers. We will be developing an API for game developers, so that users of accelerator boards for two-person games can make use of the on-board two-view hardware support provided in those boards to simultaneously accelerate left and right views in our display.

Another application area we intend to explore is teleconferencing. With a truly non-invasive stereoscopic display, two people can appear to be having a normal conversation with each other across a table. Aside from being able to each see a common object or scene on the table (from either the same or opposite viewpoints), the head movements by each participant reinforce the sense of presence and solidity of the other, and proper eye contact is always maintained. In order to provide the most effective interface, the display would probably be based on a scaled-up, full color version of the projection-based implementation described in this paper.

We are also working with manufacturers of rapidly switchable flat-panel displays to create a portable version. One of our goals for this flat-panel based version is a hand-held "gameboy" or "pokémon" size platform for personal autostereoscopic displays. Since the cost of the pi-cell light shutter and its associated control electronics is roughly proportional to display area, we believe that portable hand-held autostereoscopic displays can be a practical low cost platform. Aside from games, there may be a viable market for portable visualization devices; one example would be for maintenance and repair guides for complex systems such as aircraft or automobiles. It may also be useful to have a small portable window on a larger “virtual” world to circumvent the limitations of current hand-held displays.

REFERENCES

Displaytech: http://www.displaytech.com/shutters.html

ELSA: http://www.elsa.com/

InFocus: http://www.infocus.com/

LXD: http://www.lxdinc.com/

C. Morimoto, D. Koons, A. Amir, M. Flickner. “Pupil Detection and Tracking Using Multiple Light Sources.” Image and Vision Computing; Vol. 18 No. 4, March 2000, pp. 331-335.

K. Perlin, S. Paxia, J. Kollin. "An Autostereocopic Display." SIGGRAPH 2000 Conference Proceedings; Vol. 33 No. 3. 2000.

Texas Instruments: http://www.ti.com/dlp

Xilinx: http://www.xilinx.com

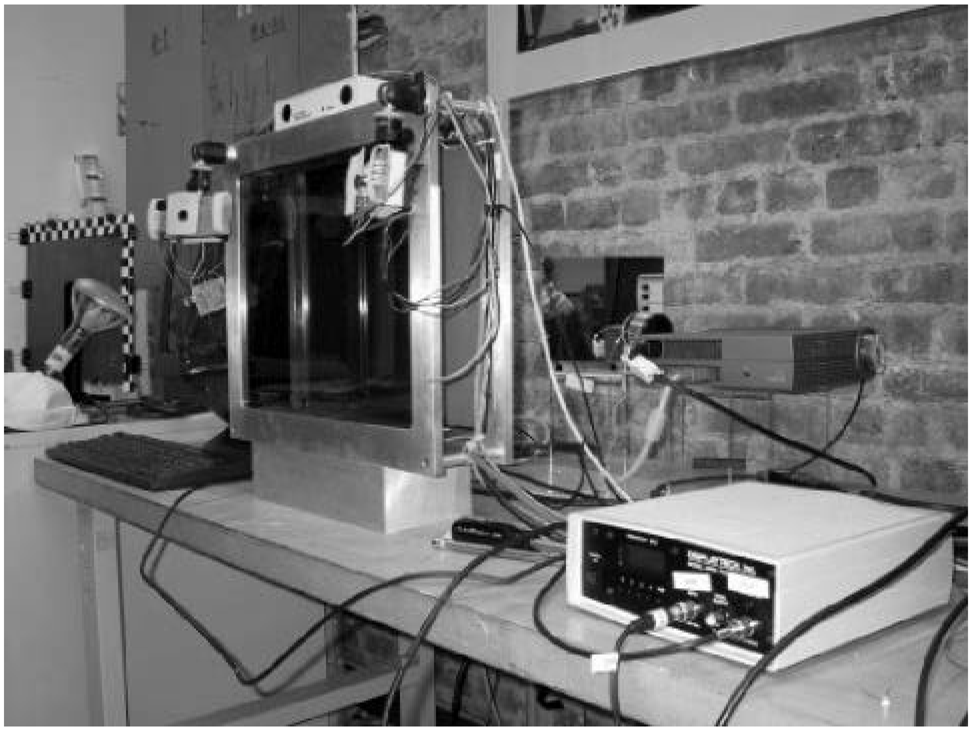

Figure 1 - The autostereoscopic display
The display is shown at the left, with the BlueEyes cameras mounted on its front. Light from the projector at the right passes through the ferroelectric shutter, bouncing off of the mirror onto the screen at the back of the display. The front of the display holds the LCD shutter. The back of the calibration target can be seen at the extreme left.

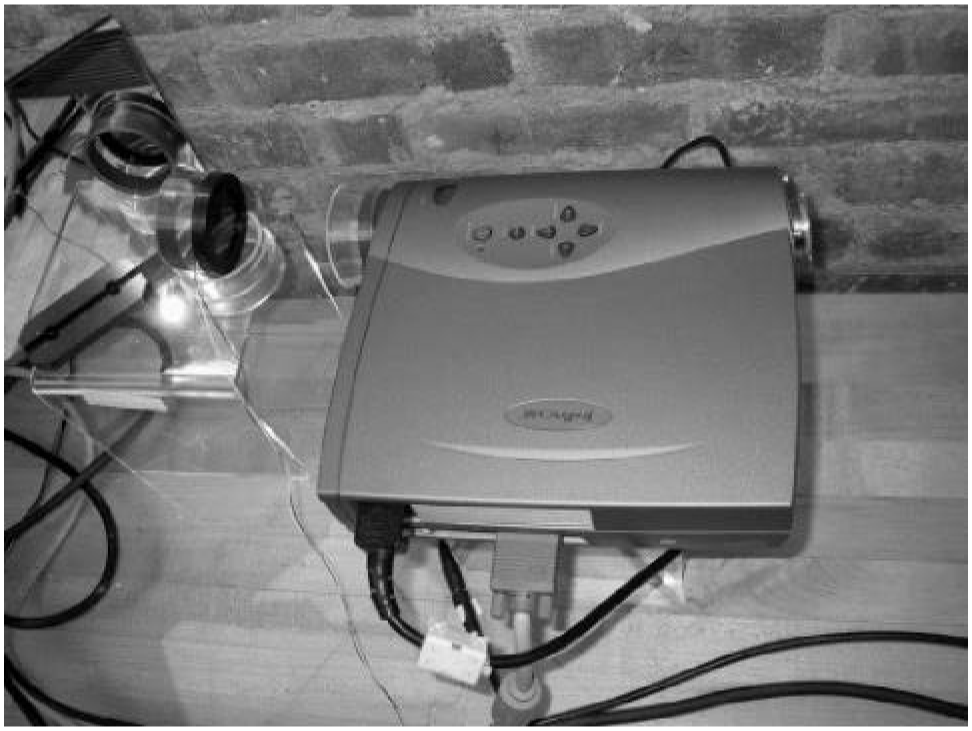

Figure 2 - LP330 projector detail
The externally mounted color wheel is at the back of the projector, at the right of the image. The ferroelectric shutter can be seen at the left of the image, directly in front of the mirror.

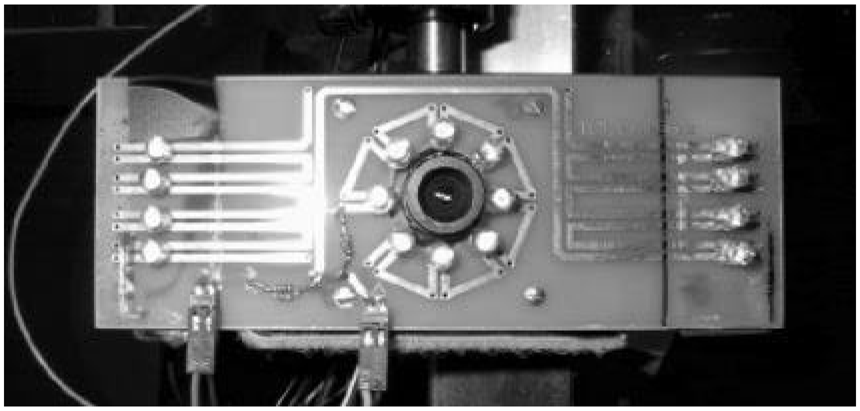

Figure 3 - Detail of a BlueEyes camera
The diffusers have been removed to reveal the inner ring and outer strips of LEDs

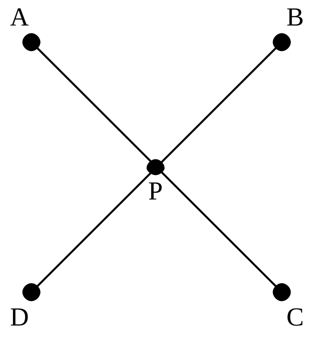

Figure 4 – Saddle point algorithm

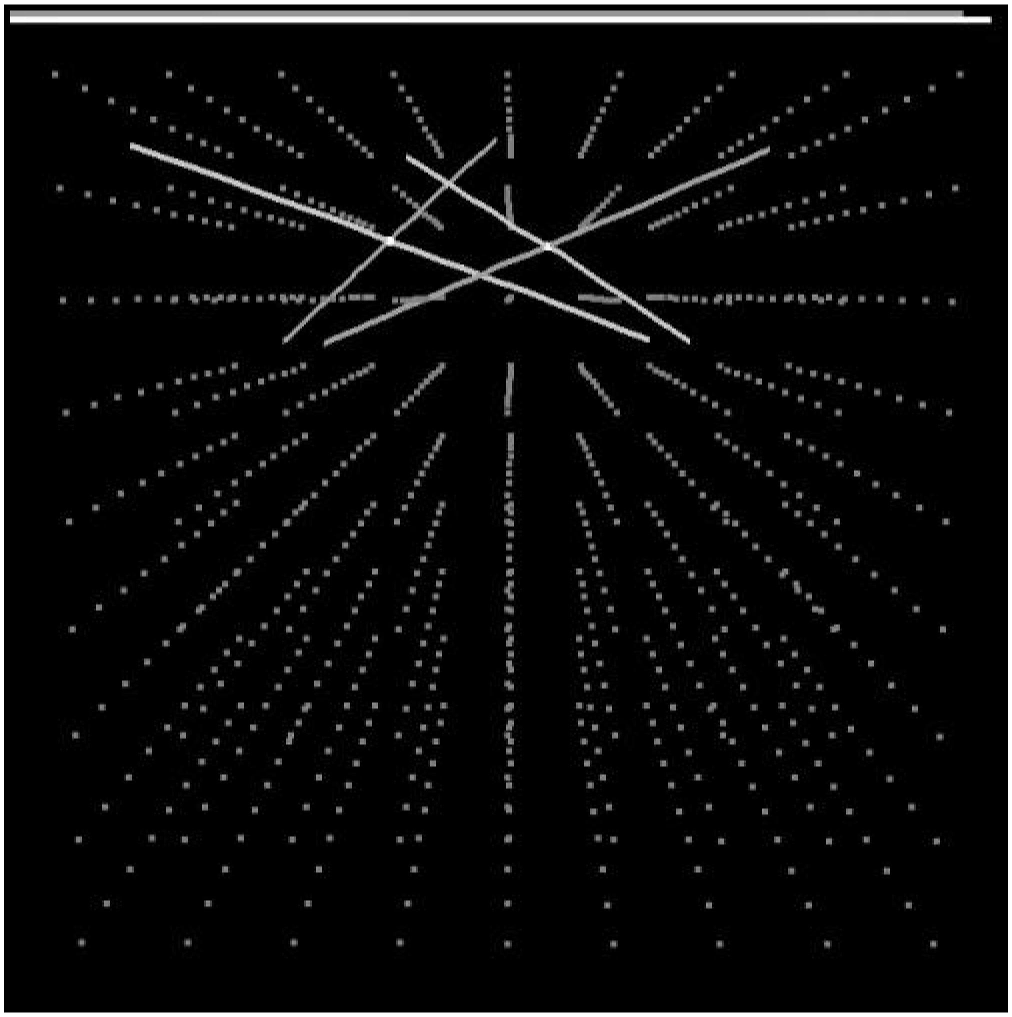

Figure 5 - Spline-fitting visualization
This screens shot shows the intersecting splines and eye positions for two eyes. The horizontal bars at the top represent the confidence rating for each eye position. The field of dots is a sparse grid of known saddle points.