Pointing into Remote 3D Environments

Robert Ulichney[a] and Matthew Gaubatz[b]
[a]Hewlett-Packard Co., 200 Forest St., Marlboro, MA 01752-3085;
[b]Hewlett-Packard Co., 76 Cedar St, Seattle, WA 98121-4110

ABSTRACT

In anticipation of the proliferation of micro-projectors on our handheld imaging devices, we designed and tested a camera-projector system that allows a distant user to point into a remote 3D environment with a projector. The solution involves a means for locating a projected dot, and for adjusting its location to correspond to a position indicted a remote user viewing the scene through a camera. It was designed to operate efficiently, even in the presence of camera noise. While many camera-projector display systems require a calibration phase, the presented approach allows calibration-free operation. The tracking algorithm is implemented with a modified 2D gradient descent method that performs even in the presence of spatial discontinuities. Our prototype was constructed using a standard web-camera and network to perform real-time tracking, navigating the projected dot across irregularly shaped and colored surfaces accurately. Our tests included a camera-projector system and client on either side of the Atlantic Ocean with no loss of responsiveness.

Keywords: Projection; Camera-projector system; 3D environments; real-time interaction

1. INTRODUCTION

Tiny mobile camera-projector systems will likely be commonplace in the not-too-distant future. The motivator for micro-projectors is to address the display needs of the growing number of small "image challenged" devices in use. While the main purpose of these projectors will be to display images, their existence opens the door to new applications. The goal of this study is to develop one such application in anticipation of this new portable projection technology.

The particular application of interest is a remote light pointer. Current portable devices with image capture functionality, such as web-cameras and cell phones, allow a representation of the surroundings to be conveyed to a remote destination. The addition of a projector onto such a device would further enable a remote user to interact with these surroundings in three dimensions. For example, a remote user would have the ability to point at objects in the capture device's environment, much in the way a laser-pointer is used during a lecture or presentation. With the growing possibility of ubiquitous tiny projectors on handhelds such as PDAs and cell phones, it is easy to consider situations were mere audio and video communication is not sufficient to locate specific objects in a scene. Examples might include remote service of complex hardware, web-based laboratory demonstrations, or even simple identification of a needed item on a work bench. This pointing approach is more flexible and simpler than other efforts to interact with a 3D environment where, for example, transponders are needed on objects of interest[1].

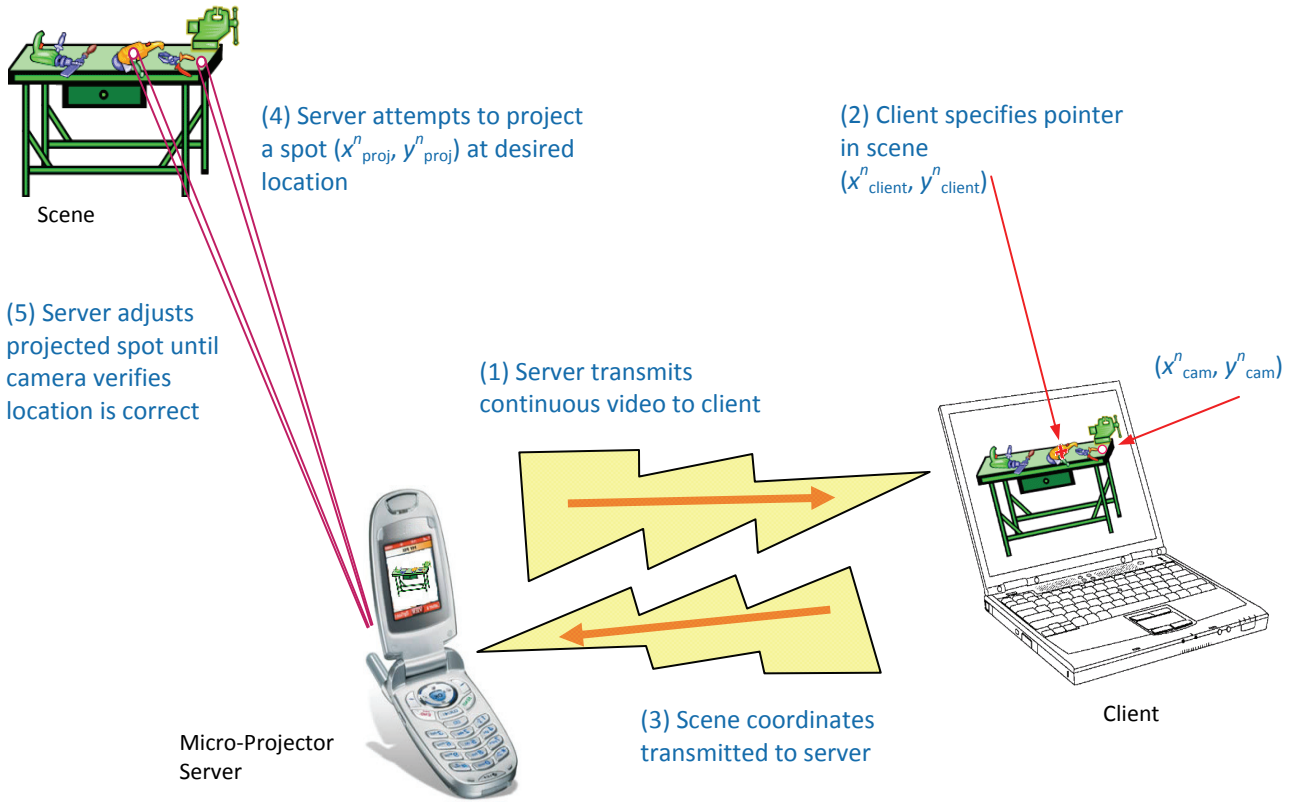
Bove of MIT presented a vision for micro-projectors in his 2003 paper[2] with goals of very small size, cost and power. He also described his prototype based on based on vertical cavity surface emitting lasers, or VCELS. VCELS, manufactured for the optical communications industry, are tiny arrays of lasers, typically 25. Today several manufacturers[3] are developing micro-projectors, and prototypes of cell- phone-based projectors are starting to appear. The tiny projectors use a number of creative display means. These include single solid state lasers with micro mirrors for scanning in two dimensions, white light sources reflected from LCoS (liquid crystal on silicon), or DLP mirror arrays. To avoid moving parts one manufacturer is using computer generated holograms to steer rather than reflect the light.

2. PROBLEM FORMULATION: RESOLVING COORDINATES

The image processing challenge associated with building such a system is the implementation of a server with the following functionality: given a requested scene coordinate from a remote client, the device must project a dot in the three-dimensional environment, such that it will appear at the desired location in the captured image in real time. In order to do so, the server must effectively resolve the difference between the coordinates of the projection system and coordinates in the captured image. In lieu of working with state-of-the-art micro-projection technology, the approach

adopted herein is to implement this functionality with a camera-projector system constructed from readily available components. There exist methods to do so which require a calibration, or even initialization stage.[4-8] These methods, however, need to be re-calibrated/initialized if the relationship between the camera and projector changes. (Note that it is not necessary that the camera and projection system be located in the same place). The following notation is used to formally describe this problem, and is illustrated in Figure 1. Because this system must adjust over time, a superscript n is introduced to denote quantities associated with time instance n.

One purpose of the server at all times is to transmit a representation of the captured scene to a remote location. The module designed to perform this task, image transmission, is only discussed briefly in this paper. More importantly, the server must be able to adjust (x^n_{proj}, y^n_{proj}) such that (x^n_{cam}, y^n_{cam}) matches $(x^n_{client}, y^n_{client})$. This second task can be split up into two simple subroutines, which are discussed in more detail in the following sections: (1) robust detection of $(x^n_{cam'}, y^n_{cam'})$, i.e., estimation of (x^n_{cam}, y^n_{cam}), based on the captured image, and (2) adjustment of (x^n_{proj}, y^n_{proj}) based on $(x^n_{client}, y^n_{client})$, and $(x^n_{cam'}, y^n_{cam'})$.



The different coordinate systems:

$(x^n_{client}, y^n_{client})$: location in the camera/captured image coordinate system where the user desires the projector to point.
(x^n_{proj}, y^n_{proj}): location in the projector coordinate system where the projector is specified to point by the server.
(x^n_{cam}, y^n_{cam}): location in the camera/captured image coordinate system where the projector is pointing.
$(x^n_{cam'}, y^n_{cam'})$: location in the camera/captured image coordinate system where the server detects the projector is pointing.

Figure 1. Diagram of scene, server and client with projector and camera coordinate systems.

3. ASYNCHRONOUS SERVER DESIGN

If the server could be designed as a single integrated piece of hardware, certain design challenges, such as synchronization, could be avoided. Nevertheless, in order understand how to construct a system that is applicable on a variety of different platforms, it is reasonable to consider how to implement it for a more general purpose multi-threaded software environment. Modularization of the server ensures that as improvements are made to its respective components, updates can be applied without adversely affecting performance. Asynchronous design can similarly benefit the server in a number of ways. Figure 2 represents an abstract view of the server. It is a module that performs tasks in two different loops, one that focuses on interacting with client and user, and another that controls the projector. Image capture, image compression and transmission, dot-detection and beam adjustment are considered asynchronous events.

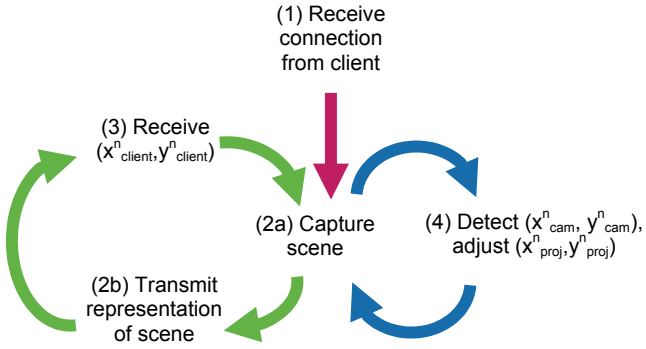


Figure 2. Illustration of the asynchronous loops in the server.

The purpose of this design is to (1) prevent connectivity issues from interfering with the operation of the beam and (2) assume no temporal relationship between events in the image capture and beam adjustment routines. The first issue arises due to uncertainty with respect to the server's connection to a remote client. Asynchronous design attempts to compensate for lost data, differences between when network data is written to output buffers and when it is actually transmitted, communication delays, heavy network traffic, etc. A side benefit is that as camera frame-capture rates increase, the speed of convergence of the algorithm will continue to improve, regardless of the bandwidth available to transmit a captured scene. The second issue is important simply because other than causality, no relationship can be inferred between the time (1) the projector coordinates are modified, (2) the projector displays a modified beam, (3) the image with the modified beam is captured by the server and (4) the location of the modified beam is detected by the server. To ensure the remote light pointer is robust to variety in camera frame capture rate, server computational power and projector refresh rate, these assumptions are respected.

4. DOT DETECTION STRATEGY

Dot detection, or projected beam detection, is simply the estimation of (x^n_{cam}, y^n_{cam}), based on a captured scene. The estimated location of the "dot", i.e., the coordinates where the server determines the projector is pointing, is denoted **(x^n_{cam}, y^n_{cam}).** Dot-detection is hampered by various factors, including: computational limitations of the server, optical/electrical limitations of the camera, and conditions of the viewing environment. The computational power of the server determines how quickly the dot can be detected and therefore, how quickly (x^n_{proj}, y^n_{proj}) can be adjusted. In the same way, the qualities of the camera lens and sensor have a significant effect on what can be detected in a captured scene, while the bandwidth between the camera and the server determines how often images can be captured. The amount of camera noise and extent of lens blur influence the smallest artifact that can be robustly resolved. Factors that are not tied to the camera also have an effect; extreme lighting conditions, as well as a high degree of motion will prevent even a casual viewer, much less an automated detection algorithm, from discerning certain details.

Arguably, the most important problems to overcome in dot-detection algorithm design are server and camera limitations. Otherwise the dot-detector will yield poor performance, regardless of the conditions of the viewing environment. Thus, a low-complexity yet robust solution is desirable. The key to satisfying this design criterion is in the specification of the behavior of the dot itself.

Figure 3. Comparison between red pointer (left) and white pointer (right). Note that the white pointer is much more visible in the image.

Many traditional laser pointers employ a red-colored beam to located objects. This color light when generated by a projected, however, is not always easily detected by a digital camera. Figure 3 (left) illustrates an example scene with a red dot. Note that though the dot is discernable, it is certainly not the most easily visible artifact in the scene. Figure 3 (right) shows the same illustration using a white dot. Note that the dot is considerably more visible. This color dot can more fully harness the dynamic range of a light projection device to improve dot-detection accuracy. Given a single captured scene, a simple dot-detection algorithm would seek out a bright (white) spot of a certain size. Unfortunately, objects of such description are not necessarily unique within a given scene.

The issue of dot- or object- detection is not novel. Red-eye detection algorithms have been proposed to detect small red dots in digital photographs[9,10], and similarly, nodule detection algorithms have been proposed as diagnostic aides for medical images.[11,12] On one hand, these algorithms do provide robust detection of dot-like artifacts, but on the other, they are designed for high-quality imagery (captured by still digital cameras), can exploit more structure, i.e., that of a face, and often require considerable computational resources. While this type of approach is appropriate for a prototype involving a laptop, it is not necessarily desirable for implementation in a cell-phone or PDA.

One particular red-eye detection approach utilizes hardware in a creative manner to simplify the problem[10]; images are captured with and without use of a flash. Since flash-induced eye glint is often present in the flash image but not in the other, it can be located by computing a modified version of the difference between the two images. A similar approach is proposed for the dot-detection strategy; by causing the dot to blink, it can be located by examining the difference between two successive captured images. When the dot is not moving, therefore, the projector will cause the dot to blink. (At a high enough rate, this blinking is not detectable by the human eye.) When the dot is in motion, on the other hand, it does not blink, since the presence of a bright moving object is easier to detect than the reoccurring absence of one. The result of this behavior is that the dot-detection algorithm can be implemented to proceed in the same manner, regardless of whether or not the dot is in motion.

Recall that n represents a discrete time index, and let $R^n(x,y)$, $G^n(x,y)$ and $B^n(x,y)$ represent the red, green and blue channels of the captured image associated with time instance n. Since the dot is white, choosing the color space in which the artifact is detected is not a critical decision. To reduce the overall storage required, the following quantity is used:

$$F^n(x,y) = R^n(x,y) + G^n(x,y) + B^n(x,y)$$

$F^n(x,y)$ represents an approximate version of the luminance of the captured frame. As each such frame is received, it is spatially filtered to reduce the effect of noise, as well as jitter introduced by beam-adjuster. Temporal filtering could also be performed, but requires access to multiple fames. Therefore, reduction of temporal artifacts is carried out in a slightly different manner, as discussed below. The spatial smoothing operation is performed with a linear kernel as follows:

$$F_{spa}^{n}(x,y) = h_{spa}(x,y) * F^{n}(x,y)$$

After this smoothing, a thresholding operation is performed, to reveal pixels that have changed over the last two frames:

$$D^{n}(x,y) = \begin{cases} 1 & \text{if } |F_{spa}^{n}(x,y) - F_{spa}^{n-1}(x,y)| > K \\ 0 & \text{otherwise} \end{cases}$$

Next, the centroid of $D^n(x,y)$, denoted $(x^n_{\text{cent}}, y^n_{\text{cent}})$, is computed, where:

$$x^{n}{}_{\text{cent}} = \mathop{\mathrm{E}}_{x,y}\{x \cdot D^{n}(x,y)\}$$

$$y^{n}{}_{\text{cent}} = \mathop{\mathrm{E}}_{x,y}\{y \cdot D^{n}(x,y)\}$$

The final estimate, $(x^n_{\text{cam}'}, y^n_{\text{cam}'})$, is computed by temporally filtering the *centroids* associated with the last several frames. In particular,

$$x^{n}{}_{\text{cam}'} = h_{\text{temp}}(n) * x^{n}{}_{\text{cent}}$$

$$y^{n}{}_{\text{cam}'} = h_{\text{temp}}(n) * y^{n}{}_{\text{cent}}$$

This filtering also helps reduce jitter in the behavior of the beam-adjuster. If $h_{temp}(n)$ has extended temporal support, the beam adjuster will move very smoothly, but very slowly as well. This system a whole is summarized in Figure 4.

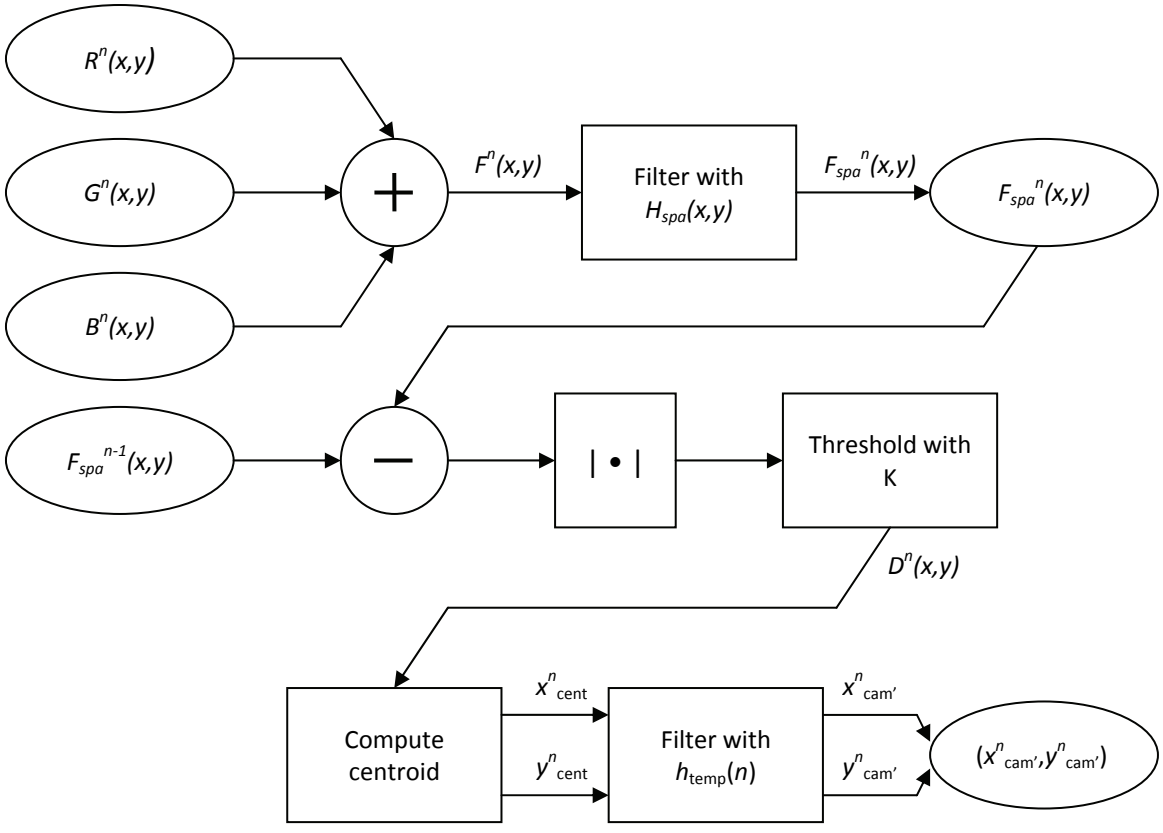


Figure 4. Diagram of dot-detection strategy.

5. BEAM-ADJUSTMENT MECHANISM

The purpose of the beam adjustment mechanism is to modify (x^n_{proj}, y^n_{proj}) such that (x^n_{cam}, y^n_{cam}) is equal to $(x^n_{client}, y^n_{client})$. One challenge is that (x^n_{cam}, y^n_{cam}), as detected by a human, can never be known explicitly, and hence, $(x^n_{cam'}, y^n_{cam'})$ must be used instead. This issue is unavoidable, but can be mitigated with a robust implementation of the dot-detector. In any case, note that where the server detects the beam depends on where the projector points it. In other words, $(x^n_{cam'}, y^n_{cam'})$ is a function of (x^n_{proj}, y^n_{proj}).

The following notation is defined to reflect this relationship:

$$x^n_{cam'} = f(x^n_{proj}, y^n_{proj}),$$
$$y^n_{cam'} = g(x^n_{proj}, y^n_{proj}).$$

In strict mathematical terms, the goal of the beam adjuster is to solve the following equations:

$$f(x^n_{proj}, y^n_{proj}) - x^n_{client} = 0,$$
$$g(x^n_{proj}, y^n_{proj}) - y^n_{client} = 0.$$

Newton's method is a classical approach to such a root-finding problem. One benefit is that it converges quadratically to the desired solution. The trouble with this approach, however, is that it requires $f(\cdot)$ and $g(\cdot)$ to satisfy regularity conditions. These requirements may not hold due to (1) inaccuracies introduced by the dot-detector, i.e., time-varying differences between $(x^n_{cam'}, y^n_{cam'})$ and (x^n_{cam}, y^n_{cam}), and (2) the three-dimensional geometry of a captured scene, which results in discontinuities in $f(\cdot)$ and $g(\cdot)$ if there are any edges present.

A more robust solution is to use a gradient-descent type update for quantities x^n_{proj} and y^n_{proj}:

$$x^{n+1}_{proj} = x^n_{proj} - \alpha(f(x^n_{proj}, y^n_{proj}) - x^n_{client}),$$
$$y^{n+1}_{proj} = y^n_{proj} - \alpha(g(x^n_{proj}, y^n_{proj}) - y^n_{client}).$$

If α is too small, the algorithm will approach the desired solution, but slowly. If it is too large, the algorithm will converge more quickly, but will oscillate around the desired solution. The key to success of this algorithm is the adjustment of α. Assuming correct behavior of the dot-detector, the server will have exact knowledge of when convergence is achieved. A heuristic is employed that adjusts α such that it is directly proportional to the Euclidean distance between where the beam appears in the camera view and where it should appear. In particular,

$$\alpha = L\sqrt{(f(x^n_{proj}, y^n_{proj}) - x^n_{client})^2 + (g(x^n_{proj}, y^n_{proj}) - y^n_{client})^2},$$

where L is a constant. The value of α can be clipped to prevent erratic behavior. This module can be further optimized for faster convergence, for example, by storing the mappings given by $f(\cdot)$ and $g(\cdot)$ in a lookup table. The module is summarized in Figure 5.

6. IMPLEMENTATION AND DISCUSSION

The general server prototype consisted of a Compaq projector, a laptop computer with a 2.0 GHz processor, and a web-camera attached to the computer. The client was installed on a separate laptop. The server and client were implemented in native Win32 C++, with the server using a generic camera API. A projector was driven with the VGA output of the server laptop. The benefits of this approach are that (1) no calibration is required, that (2) it can be used with any camera hardware that connects to a Windows PC and that (3) the camera and projector can be placed in a wide variety of configurations, as long as the projector cannot place the beam at a location inaccessible to the camera. A number of different configurations of this method were evaluated in different environments, and for different purposes.

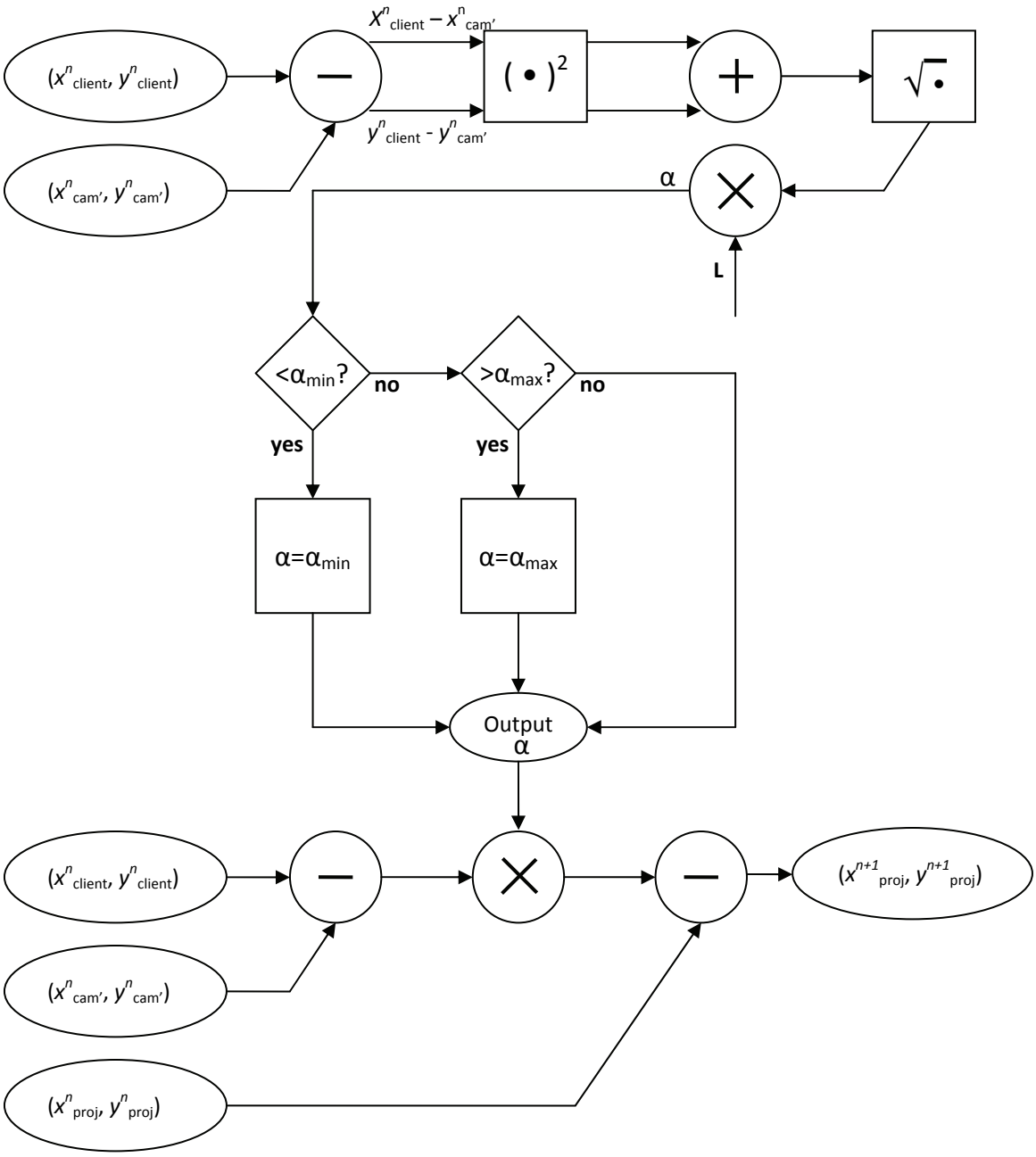


Figure 5. Diagram of the beam-adjuster mechanism.

First, the ability to allow users to accurately specify objects in a scene from a remote 3D environment was evaluated. A visual representation of the remote client UI used for this purpose is depicted in Figure 6. The client application essentially displays a video feed of the scene to a user. The user can click on any part of the scene to control the projection of the beam. The desired location is denoted with a set of red crosshairs. The server transmits the current location of the detected beam back to the client, which in turn displays a set of green crosshairs centered on that location to emphasize the systems movement of the beam for the user. A visual depiction of an example interaction that occurs using the client is given in Figure 6. Under correct operation, the beam moves to the location depicted by the user.

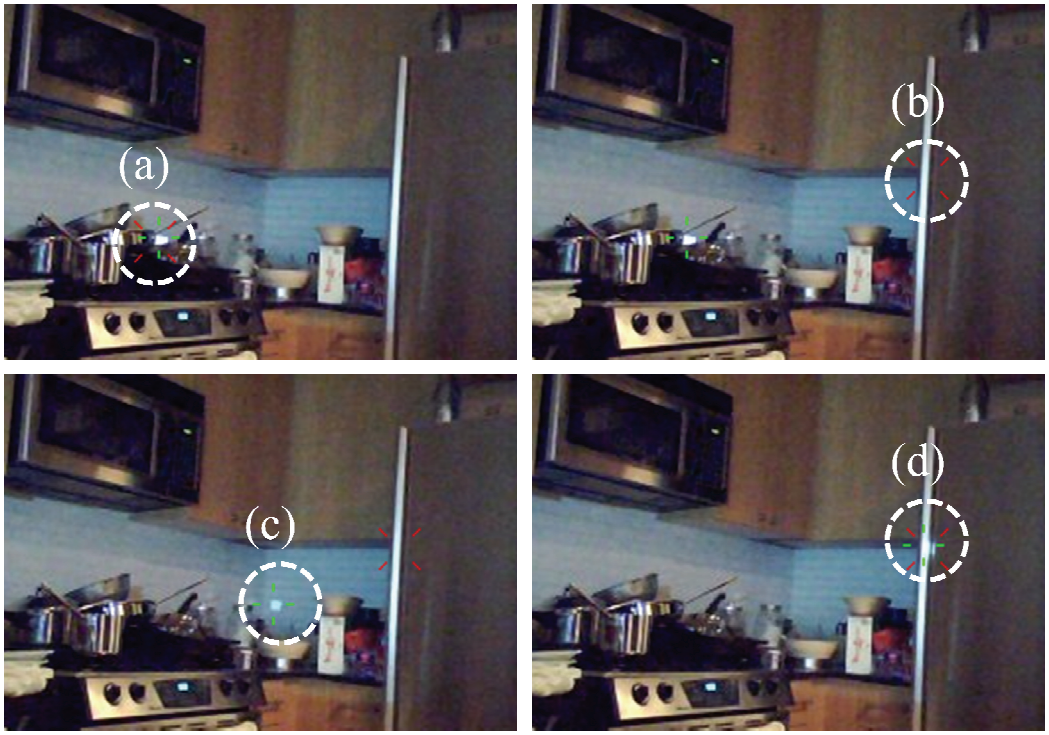


Figure 6. Diagram of a user interaction via a remote client. The four frames depicted here represent images transmitted to the client over a network. (These images have been cropped and marked with labels for improved clarity.) A client user may specify the location in the image to which the beam should point, and the client application represents this choice with a diagonally oriented red "target" crosshairs. The client application also illustrates the location of the detected bean with differently oriented green crosshairs. An example interaction proceeds as follows. (a) The beam is resting on a location specified by the user. (b) The user specifies a new location. (c) The projected beam moves towards the desired location. (d) The beam converges to the location specified the user.

This particular test was more qualitative in nature in that it was designed primarily to determine whether or not calibration-free remote 3D interaction was achievable, i.e., whether or not it was possible to use the system as a remote light pointer. The prototype successfully connected a three-dimensional scene in Ithaca New York, with human users in Palo Alto, California and Bristol, England. Tests demonstrated the ability to point to different locations, and even to "drag" the beam in a manner similar to a mouse interface. The 3D environment used for this purpose was lit at different times by natural light, incandescent light, or a combination of the two. In some cases it was necessary to add *more* lighting (*weakening* effectiveness of the projector beam) simply to make the scene clear for remote viewers. In most cases when the user pointed the beam at a new location, it would converge to that location in less than a second, even when the geometric configuration of the camera, projector and people in the scene changed in real time.

The convergence properties of the algorithm were evaluated with a set of quantitative tests, designed to analyze performance under a range of different lighting conditions as well as variety in the geometry of the scene structure. Four different environments were tested: a wall, lit by very low ambient light, a sofa lit by a normal incandescent lamp, a set of belongings in a three dimensional corner lit by natural light, and a cluttered kitchen lit by a combination of natural and incandescent light. These choices were picked to test the robustness of the projected beam adjustment mechanism to different *amounts* of light, different *kinds* of light and different degrees of 3D complexity, and are shown in Figure 7. In each test, the beam was pointed at 400 randomly generated points in the scene. The server was given each coordinate in successive fashion. If after 30 iterations of updating the beam location the algorithm did *not* converge to the desired location, the next location was presented to the server. This choice was made simply to limit the effect of cases demonstrating difficulties in convergence. The speed of convergence, and the relative distance to the desired location were recorded for each location successfully converged upon.

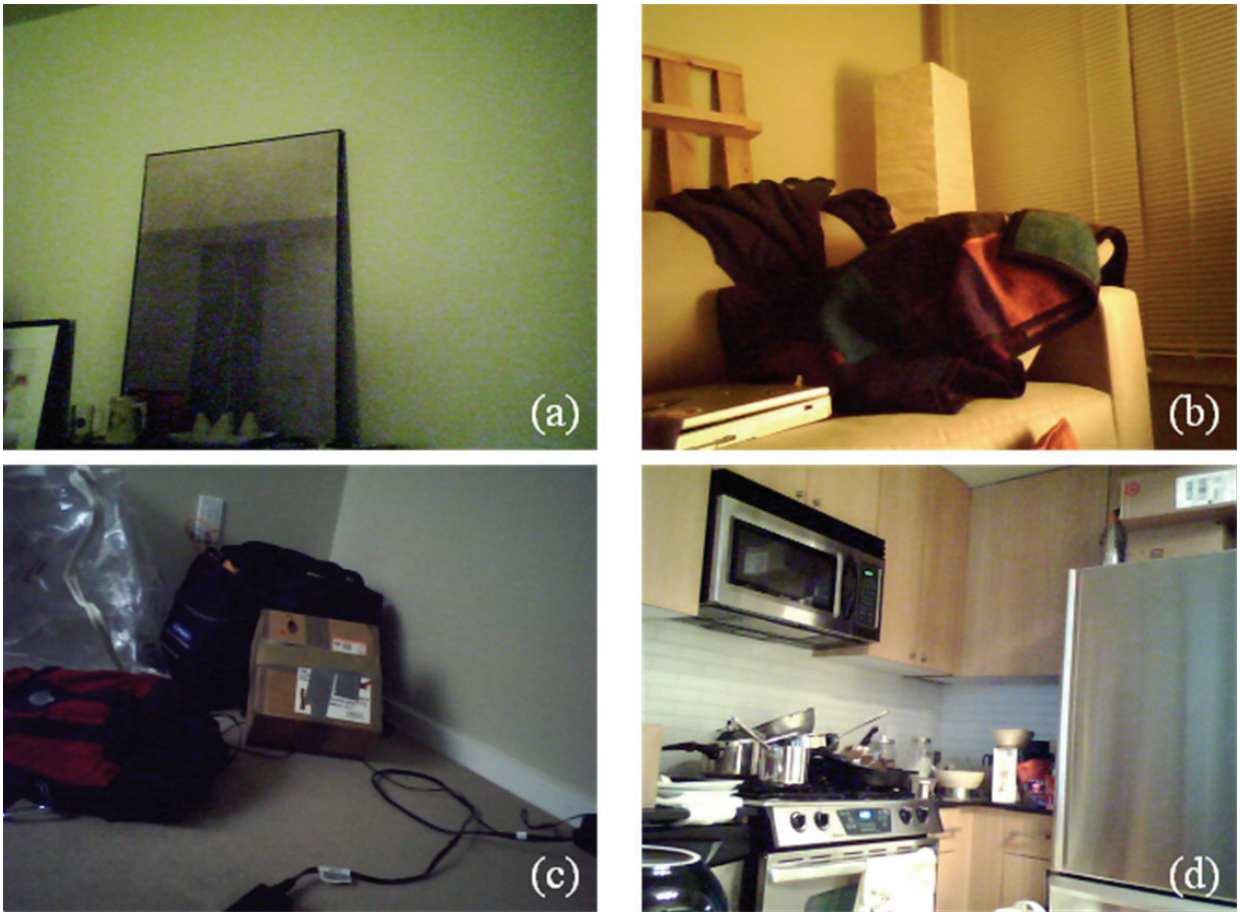


Figure 7. Different environments used to evaluate convergence properties of the proposed algorithm. (a) A wall image, with very low ambient light (a "best-case" scenario). (b) A sofa corner lit by normal incandescent lighting with a few non-planar objects. (c) A floor corner, with a variety of non-planar surfaces, and with differing reflective properties. (d) A kitchen, light by natural light as well as incandescent track lighting, cluttered by objects including those with metallic, ceramic, cloth and wooden surfaces.

Table 1. Convergence properties of different 3D environments. Note that the results are relatively consistent across environments, and that the convergence time, on average, is less than one second.

scene	mean relative distance	standard deviation of relative distance	mean iterations	standard deviation of iterations	mean time (msec)	standard deviation of time (msec)	cases that did not converge (%)
wall	0.0241	0.0034	13.2	3.54	878	235	2.25
sofa	0.0232	0.0042	13.0	5.01	866	333	6.75
floor	0.0234	0.0044	13.9	4.88	923	324	3.75
kitchen	0.0228	0.0049	13.5	3.88	899	259	11.0
average	**0.0234**	**0.0043**	**13.4**	**4.33**	**892**	**288**	**5.94**

Table 1 gives the mean and standard deviation values associated with the quantities recorded in this test, as well as percentage of failure cases. The distances are normalized by the linear distance of a diagonal line across the camera viewing window, i.e., they are listed as a fraction of this diagonal linear distance. Clearly, the simplest case (the wall environment) exhibits the best convergence properties – the most consistent convergence iteration counts/times and the fewest failure cases – but the differences in performance demonstrated by the other cases are quite small. The mean converged distances, mean iteration counts, and mean convergence times for *all* other cases were within 5% of the values associated with the wall case. (In fact, the mean converged distance for the kitchen environment, arguably the most difficult case, was actually 5% lower!) This result suggests that when convergence is achieved, regardless of the complexity of the environment, that it proceeds in approximately the same manner.

The same trends generally hold even when all failure cases are included in the averages, though differences between results for different environments are more pronounced. Specifically, there is greater variety among the resulting distances and times, and the average converged distances are up to 30% higher. Note that failure cases do not indicate that the algorithm could not point to the desired location, just that it could not do so in the number of allotted iterations. These failure effects often occurred due to several reasons, primarily occlusions. The camera and projector were necessarily separated by several feet such that the field of view of both devices was as similar as possible. As a result, the projector could point to locations unseen by the camera, or that correspondingly, the camera could see locations to which the projector could not point. Such effects on convergence are more pronounced in environments with greater 3D complexity, but also would decrease along with the distance between camera and projector, which would likely be the case if both were installed on a single, portable device.

7. CONCLUSION

This experiment demonstrated that indeed, it is possible to create a web-based light pointer device. A key factor affecting convergence time is the maximum rate at which frames can be captured by the camera. The faster this process occurs, the faster the light pointer converges to the desired location. One of the biggest surprises that resulted from this experiment was the challenge of generating a pointer beam that could be easily seen by a human user through a camera. In order to make a compelling demonstration, and in order to be useful, the light pointer needs to be visible through the camera to a remote user, as well as the environment into which it points. By nature, more complex 3D scenes are not always so easily lit by ambient light, and thus the dynamic range of the ambient lighting has a large effect on how visible the pointer is to a human observer, and the dot-detection module. Conceivably, future improvements in projection and fabrication technology could lead to improved performance of the system. Still, the visibility of the dot remains a consideration, whether or not the dot-detector and beam-adjuster function correctly.

Another important issue to be addressed is robustness to movement in the scene. In a static or even low-motion scene, the system performs quite reliably. Nevertheless, too much non-dot movement in a scene adversely affects the dot-detection module. The ability to address this issue will improve along with capture speeds and beam projection technologies, but can also be addressed by improved algorithms. Thus, an important direction of future research involves improving robustness to this effect. Object tracking is certainly not a new field, and there are many sophisticated tracking algorithms[13,14] that could potentially be used to augment the current approach. It is of interest, however, to consider how constraints associated with this particular problem can lead to more efficient solutions.

REFERENCES

[1] D.Merrill and P. Maes, "Augmenting Looking, Pointing and Reaching Gestures to Enhance the Searching and Browsing of Physical Objects", *Proceedings of the 5th International Conference on Pervasive Computing (Pervasive'07)*, Toronto, Canada, May 13-16 (2007).

[2] Bove, V.M. and W. Sierra, "Personal Projection, or How to Put a Large Screen in a Small Device," *Proc. SID* (2003).

[3] Graham-Rowe, D., "Micro-Projectors Set to Be Big", *Technology Review*, May 28 (2009).

[4] Chang, N.L. and A. Said, "Automatically Designed 3D Environments for Intuitive Exploration," *IEEE International Conference on Multimedia and Expo (ICME)*, Taipei, Taiwan, June (2004).

[5] Rocchini, C., P. Cignoni, C. Montani, P. Pingi and R. Scopingo, "A Low Cost 3D Scanner Based on Structured Light," *Computer Graphics Forum*, vol. 20, (2001).

[6] Rusinkiewicz, S., O. Hall-Holt and M. Levoy, "Real-time 3D Model Acquisition," *ACM Transactions on Graphics*, vol. 21, (2002).
[7] Zhang, L., B. Curless and S. M. Seitz, "Rapid Shape Acquisition Using Color Structured Light and Multi-Pass Dynamic Programming," *1st International Symposium on 3D Data Processing, Visualization and Transmission*, Padova, Italy, June (2002).
[8] Tardif, J.P., S. Roy and M. Trudeau, "Multi-Projectors for Arbitrary Surfaces Without Explicit Calibration Nor Reconstruction," *IEEE International Conference on 3-D Digital Imaging and Modeling (3DIM)*, (2003).
[9] Gaubatz, M. and R. Ulichney, "Automatic Red-Eye Detection and Correction," *IEEE International Conference on Image Processing (ICIP)*, Rochester NY, September (2002).
[10] Xiao-Ping, M. and T. Sim, "Automatic Red-Eye Detection and Removal," *IEEE International Conference on Multimedia and Expo (ICME)*, Taipei, Taiwan, June (2004).
[11] Yoshino, S., T. Tanaka, M. Tanaka and H. Oka, "Application of Morphology for Detection of Dots in Tumor," *Annual Conference of the Society of Instrument and Control Engineers (SICE)*, August (2004).
[12] Agam, G., S. G. Armato III, Changhua Wu, "Vessel Tree Reconstruction in Thoracic CT Scans with Application to Nodule Detection," *IEEE Transactions on Medical Imaging*, vol. 24, April (2005).
[13] Park, D.K., H. S. Yoon and C. S. Won, "Fast Object Tracking in Digital Video," *IEEE Transactions on Consumer Electronics*, vol. 46, August (2000).
[14] Chao, H., Y. F. Zheng and S. C. Ahalt, "Object Tracking Using the Gabor Wavelet Transform and the Golden Section Algorithm," *IEEE Transactions on Multimedia*, vol. 4, December (2002).