Portable Projection-Based Augmented Reality

Nitish Padmanaban
Stanford University
nit@stanford.edu

EE 367
March 14, 2016

Abstract

Advances in projection techniques have opened up projection-based augmented reality as a possible alternative to head mounted display technologies for creating augmented reality systems. This project aims to show the viability of projection-based augmented reality in a portable format. The system is constructed using OpenGL and commercially available hardware that demonstrates some limited form of interaction in three dimensions with a projected 3D object, specifically a cube. Moving forward, this could be expanded into a much richer interaction paradigm for augmented reality.

1. Introduction

The prevalent mode of interactive display technology today is based around the touchscreen: a capacitive surface constrained in both display and interaction to two dimensions. Humans have evolved to be capable of interacting with the rest of their environment in 3D however, and the paradigm of the touchscreen severely limits the subset of motions that can transpire between a user and a device. On the other hand, augmented reality (AR) using head mounted displays (HMDs) are in development to provide three dimensional interactivity. However, unlike touchscreen devices, HMDs enclose or otherwise obstruct the user's head, which has issues of comfort and also impedes communication with other people that are physically present. On the other hand, a projection-based system that could project directly onto the user's environment could allow for unconstrained 3D interaction while also leaving the user's head free.

2. Related Work

As discussed by Marner *et al*. [1], projection-based AR has been used in a variety of ways based on the exact configuration of the projectors and surfaces. They show examples of projection of 2D interfaces onto various objects, but also full 3D projections with complex geometries. They further note that projection-based techniques, while not requiring the viewpoint of the user, can be much more powerful if the viewpoint is known, with effects for lighting and virtual geometries.

An ideal projection surface for these uses would be diffuse and white, but often, that isn't case with objects readily available in the user's environment. Nayar *et al*. [2] present a method to radiometrically compensate for non-ideal texture and color of the surface, with both an offline calibration process with known surfaces and an online one given photos of the desired projection surface. Grundhöfer and Bimber [3] extend this work to prevent intensity and color clipping artifacts during radiometric compensation by preprocessing the displayed images.

The radiometric compensation methods, and in addition the several other methods, including geometric compensation and high speed projection, presented by Bimber *et al*. [4] combine to allow for a portable projection-based AR system to work. One would be able to accomplish this via a camera that detected the surface and compensated in real time.

3. Approach

The approach taken here serves as a proof of concept for a portable projection-based AR system. The major aspects of the project are the hardware setup, the software and geometric models used, and finally, the calibration process to combine them in a functional manner.

3.1. Hardware

The two components that form the core of the setup are a Sony MP-CL1 mobile projector and the Razer Hydra by Sixense. The projector is a laser scanning projector, allowing for depth invariant focus, and is capable of operating fully wirelessly, which is advantageous for portable use. The field of view (FOV) is reported by the manufacturer to be a 40 in diagonal image at a distance of 1.15 m, with an aspect ratio of 16:9.

The Razer Hydra is a pair of gaming controllers capable of reporting full positional information (rotation

Figure 1. The assembled projection system, with the display surface, Razer Hydra base, and the projector and Hydra controller (which are bound to the same acrylic sheet).

and location) with nominally 1° and 1 mm of precision. The controllers operate via a magnetic field, reducing the computational load and possible detection errors relative to a camera-based tracking system.

A controller and the projector are coupled to each other by binding them to a sheet of acrylic so that they remain immobile relative to one another. The completed assembly is visible in Figure 1. Another controller is bound behind the display surface (not visible).

3.2. Software and Models

The software needed to be capable of two main tasks: communicating with the controllers using the Sixense API, and rendering a viewpoint dependent 3D scene. The API is provided as a set of C and C++ libraries, making OpenGL an ideal choice for the rendering software.

The scene to be rendered for the project is a cube that rotates with the display surface, as if it were bound to it. The two viewpoints (projector and user) map to the same physical surface from different angles, as seen in Figure 2. The forward model here is to project the scene onto the display surface, which is then viewed by the user in the correct perspective. It is however, more useful to consider the reverse for the rendering pipeline. First, we take a 3D scene, render it from the user's perspective, and project it onto the display surface. We then render the display surface (specifically the virtual scene what the user sees) from the projector's viewpoint.

Each of the rendering steps is done by a perspective projection transform, which is based on the geometry of a pinhole camera. Given a scene whose coordinates are (x, y, z), where z is measured as distance straight from the camera (or eye, or projector), the projected coordinates are $(x/z, y/z)$. This transform is illustrated from two to one

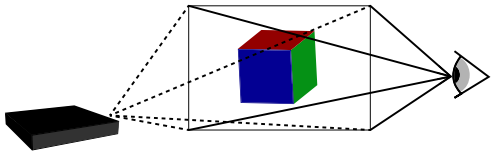

Figure 2. The projector and user view the same physical location from different viewpoints. The projection onto the surface is in the user's perspective.

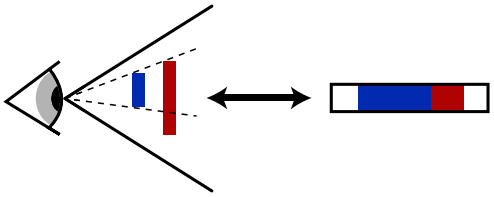

Figure 3. The perspective projection from two dimensions to one. The division by distance is effectively projection to any arbitrary plane along radial lines from the viewer.

dimension in Figure 3. Of course, a coordinate transform is required to get the z coordinate to be the distance from the camera, essentially just a rotation followed by a translation that moves the coordinate axis from the virtual object or display surface to the user or projector, respectively.

Unfortunately, the perspective transform assumes projection onto a plane perpendicular to the direction of view, but that isn't the case here. As the display surface is shifted and rotated, the plane to be projected onto changes angle relative to the user. A diagram of the required transformation is shown in Figure 4. To find the transform in 1D, we consider a point on the plane with coordinate y. We define the vector u_1 as the one pointing towards the viewer from y, and y_r as a vector along the rotated plane. We'll use a subscript y or z to denote the component along that direction (where z is away from the viewer). We want to find constants c and k such that

$$\begin{aligned} cu_1 + y &= ky_r \\ cu_{1,z} &= ky_{r,z} \\ cu_{1,y} + y_y &= ky_{r,y} \end{aligned} \tag{1}$$

where the second two lines are just the first one broken into their components. Solving, we have that the transformed point is at

$$y' = y + \frac{y_y}{u_{1,z}\frac{y_{r,y}}{y_{r,z}} - u_{1,y}} \cdot u_1 \tag{2}$$

While that applies to a line in 1D, it needs to be extended to a 2D plane. We accomplish this by finding the equivalent

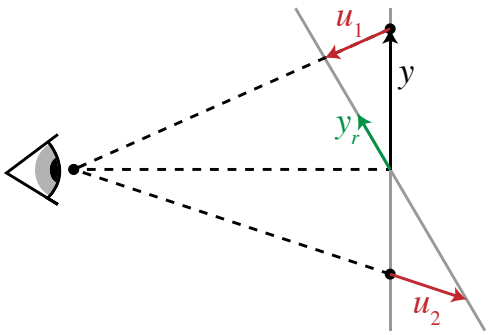

Figure 4. The perpendicular plane of the perspective transform (Figure 3), in the direction of y needs to be transformed to lie on a plane in the rotated direction y_r. We find this by looking at the intersection of the u_i directions from each point with the y_r plane.

y_r and x_r directions which have no x or y component respectively, and adding their contributions. Given the orientation of the plane, x_p and y_p, which are perpendicular to the plane's normal, y_r can be found as

$$y_r = y_p - \frac{y_{p,x}}{x_{p,x}} x_p \tag{3}$$

and similarly for x_r. The transform has the effect of stretching the projected image on the half of the plane further from the viewer.

3.3. Calibration

The Sixense API reports the position and orientation of the controllers, but these need to be transformed to the position and orientation of the projector and display surface. Using direction cosine rotation matrices, which are reported by the API, the problem of calibration comes down to finding the actual rotation R given the controller rotation R_c and the 0-rotation R_0 found via offline calibration. The three are related by

$$R_c = RR_0 \iff R = R_c R_0^\intercal \tag{4}$$

i.e., we want the rotation R in addition to R_0, the relative base rotation from the controller to the projector or display.

A few different approaches were attempted here. The first was simply to place the display surface and projector in an unrotated state (i.e. $R = I$) and use the reported R_c. Given the tradeoff of effort, and the nonlinearities that prevented a proper calibration (more below), this is likely the most practical method.

Another method that was attempted was to calibrate via manual registration to a physical box using the keyboard while simultaneously projecting. This was able to calibrate really well for that particular projector position, but ultimately proved no better than the previous method when the projector was moved.

Finally, the method that was used, was to use a calibration checkerboard pattern and align a projected checkerboard against it in various known rotations to find the z and y axes of the projector. Multiple measurements were taken while rotating the projector in 90° increments and aligning with the target. By design, by rotating the projector only around the z-axis, it is an eigenvector of the rotation. We have, for two rotations, R_1 and R_2, rotated about the z-axis relative to one another

$$R_2 = R_z R_1 \iff R_z = R_2 R_1^\intercal \tag{5}$$

where R_z is the z rotation from which the axis $\hat{z}'$ can be found. This then needs to be found relative to an unrotated system, by applying

$$\hat{z} = R_1^\intercal \hat{z}' = R_2^\intercal R_z \hat{z}' = R_2^\intercal \hat{z}' \tag{6}$$

which holds by Equation 5 and since $\hat{z}'$ is the eigenvector of R_z corresponding to eigenvalue 1.

The y-axis was found similarly by rotating the projector on a flat surface. The x-axis is then just orthogonal to these axes. Finally, the translation calibration was straightforward and simply measured empirically.

As mentioned above, it ultimately turns out that the Razer Hydra doesn't seem to report rotations accurately, despite the precision reported by the manufacturer. The inaccuracy furthermore was seen to be nonlinear such that even without calibration, one could place the controllers such that they point to one another, then move the tracking base and change their relative pointing directions. This resulted in the decision to change from a box as the display surface to a flat plane, to minimize distortions.

4. Results

Given the nonlinear behavior of the controllers, the results are interpreted qualitatively. In Figure 5, we see an example of the views of the cube when the display surface is rotated such that the desired surface should be visible (e.g. when the surface's top edge is tilted forward, the top of the cube becomes visible). Also, we see in testing that, within a reasonable range of motion, moving the projector or the display surface keeps a stable image of the cube on the display surface.

Furthermore, we also see the desired projection effect that would be seen for a tilted plane as calculated by the method shown in Figure 4. The farther half of the plane is more stretched than the near half. Unfortunately, the user viewpoint that's being calculated relative to the final image visible on the plane is unknown due to the nonlinearities, so it is difficult to evaluate why the tilt towards the right face shows the effect most strongly.

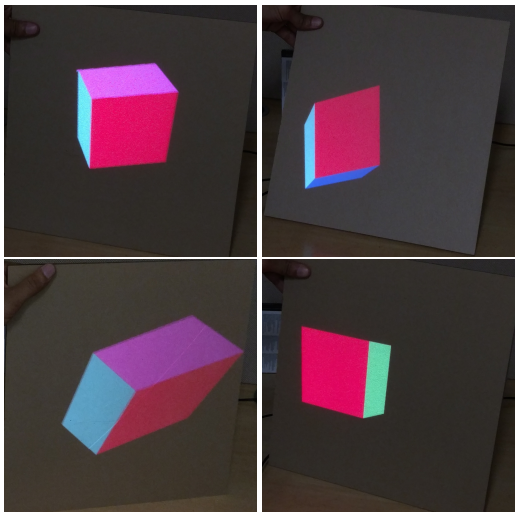

Figure 5. Clockwise from the top-left: views of the cube when the display surface is tilted such that we see the top, bottom, right, and left. The shearing based on the transform to the plane (Figure 4) seems qualitatively correct for some viewpoint.

Finally, it was also seen that the perspective illusion of a cube did not hold, from any single viewpoint as the plane was rotated freely.

5. Discussion

There are several possible reasons for why the perspective illusion for the cube failed to hold for any given viewpoint. The most obvious would be that the failure to calibrate to a high degree of accuracy. This resulted in not just an unknown exact viewpoint (though is was defined to be at a specific location, the variable projector and display location throws this off), but also that the cube shifts on the surface as it is rotated. If the cube could be held steady with accurate calibration, that would alleviate a major impediment to the illusion.

Another possible cause of the lack of 3D illusion could limit the uses of the technology. The desired distance for use, since this is meant to be a interactive display technology, is within arm's length. That's close enough for the binocular disparity between the eyes and the accommodation of each eye to register that the surface being projected onto is in fact not perpendicular to the eye, with part of the plane being further than another. If this is a fundamental limit of the technology, it would prevent formation of virtual geometries on a preexisting surface, though texturing the surface would still be viable.

However, that said, if the calibration issue is fixed, given the qualitative results, projection-based AR could easily be made usable in a portable fashion, and virtual geometries would not be an essential addition.

6. Future Work

The first path forward would be replace or augment the tracking system with another one to better calibrate the system. One approach would be to use a camera in conjunction with the projector to visually correct or obtain the calibration in real time. This would have the advantage of also being usable for real time radiometric compensation [4]. A better tracking method would allow expansion to projecting onto 3D geometries without fearing registration artifacts.

This would also allow for radiometric and geometric compensation, which can help hide the textures of the display surface and impose our own. Hiding the true surface texture from the user might also have the added benefit of fixing the lack of perspective illusion by sufficiently fooling the user.

7. Acknowledgments

The author thanks Sean Follmer for use of the Razer Hydra and both him and the Stanford Computational Imaging Group for help with various mathematical and software queries. This material is based upon work supported in part by the National Science Foundation Graduate Research Fellowship Program.

References

[1] M. R. Marner, R. T. Smith, J. A. Walsh, and B. H. Thomas, "Spatial user interfaces for large-scale projector-based augmented reality," *IEEE Computer Graphics and Applications*, vol. 34, pp. 74–82, Nov 2014.

[2] S. K. Nayar, H. Peri, M. D. Grossberg, and P. N. Belhumeur, "A projection system with radiometric compensation for screen imperfections," in *ICCV workshop on projector-camera systems (PROCAMS)*, vol. 3, Citeseer, 2003.

[3] A. Grundhöfer and O. Bimber, "Real-time adaptive radiometric compensation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, pp. 97–108, Jan 2008.

[4] O. Bimber, D. Iwai, G. Wetzstein, and A. Grundhöfer, "The visual computing of projector-camera systems," in *Computer Graphics Forum*, vol. 27, pp. 2219–2245, Wiley Online Library, 2008.