Inexpensive Immersive Projection

Nancy P. Y. Yuen[*] — California State University East Bay

William C. Thibault[†] — California State University East Bay

Abstract

Most projector-based immersive displays have numerous problems relating to complexity, space, and cost. We present a technique for rendering perspectively correct images using a casual arrangement of a projector, a mirror and a display surface.

Our technique renders an arbitrarily wide field of view using an efficient GPU-based single-pass rendering algorithm. The rendering algorithm is preceded by a one-time camera-based geometric correction calibration step. As we will describe, this technique can be implemented with inexpensive, commodity hardware and using readily available display surfaces. Thus, this technique enables immersive projection systems to be used in casual locations, such as a classroom or even in the home.

Keywords: display algorithms, viewing algorithms, camera calibration, projector-camera systems

Index Terms: I.3.3 [Computing Methodologies]: Computer Graphics—Picture/Image Generation; I.4.1 [Image Processing and Computer Vision]: Digitization and Image Capture—Camera Calibration

1 Introduction

Projectors are often used to create a large field of view in an immersive environment. Such environments have many useful application in entertainment, production, and education. However, existing projector-based immersive environments have numerous complexity, space, and cost problems. They often require an expensive screen, and expensive projectors and lenses to function. Also, precise and time-consuming positioning of the projector is often needed. These factors drive the cost of immersive environments beyond the reach of many institutions.

Our goal is to build an immersive projection system that meets these requirements: uses minimal, affordable hardware; takes advantage of existing surfaces for display; requires minimal setup and configuration; produces a large field of view, greater than 180 degrees; requires only a casual arrangement of system components; avoids determining display surface geometry; supports real-time interaction.

This paper describes an inexpensive immersive projection system built using commodity parts to display images on diffuse surfaces. It uses a one-time camera-based geometric calibration step to determine how the final image should be displayed. It does not require the computation of projector intrinsics or extrinsics, and functions for any arrangement of projector and display surface geometry. The result is a system that eliminates the usual cost and complexity associated with projector-based immersive display environments. Further, our approach uses a GPU-based single-pass rendering algorithm suitable for real-time interaction. Existing applications can be ported to use this approach with minimal modification.

[*]e-mail:nyuen@horizon.csueastbay.edu

[†]e-mail:william.thibault@csueastbay.edu

IEEE Virtual Reality 2008
8-12 March, Reno, Nevada, USA
978-1-4244-1972-2/08/$25.00 ©2008 IEEE

2 Prior Work

Projectors are useful for creating large displays on surfaces of any shape. However, to surround the viewer with imagery, 3D computer graphics rendering algorithms must address the limitations of perspective projection. Wide fields-of-view create distortions when using perspective, and even the widest perspective viewing volumes must have a field-of-view of less than 180 degrees.

The usual approach is to use a cubemap [11] to store the panoramic image prior to mapping it to the display surface. Each face in a cubemap is created with a separate rendering pass over the dataset. In each pass, the viewing and projection transformations are set to that particular face. GPUs that support multiple render targets can create a cubemap with a single pass over the geometry. Cubemaps are attractive as they have simple, fast hardware implementations. However, most cubemap texture filtering implementations fail to correctly handle edges of the cube.

Traditional "high-end immersive projection VR" systems such as the CAVE [10] [9] feature high-resolution rear-projection, stereo, and head-tracking. These systems are effective for creating a sense of immersion that fills the viewer's visual field while allowing freedom of motion. Although effective, these systems are expensive in terms of hardware and physical space.

Small, inexpensive dome displays can be created using a single projector and a spherical mirror, such as those sold for safety or security applications [7]. These inexpensive mirrors are far from perfect, however. Simple geodesic domes can be easily constructed from common materials such as cardboard [14]. Software written assuming a specific geometry for the projector, mirror, and dome screen is becoming available [4].

Cameras are increasingly used to calibrate projected displays [17] [16] [5] [15]. These typically use homographies (projective transformations) to model the projection process, which limits their use to planar display surfaces. Another approach to the use of camera-based calibration is Raskar's two-pass algorithm [17], which first renders the desired image to a texture, and in a second pass renders a model of the display surface that is textured with the image, using texture coordinates derived from the camera calibration.

The direct use of camera-projector correspondences to perform this real time warp [19] [8] [13] has the advantage that non-linear distortions in the projectors are handled. Also, screens of any shape are supported, as display surface geometry is not needed. Commercial systems using this approach have begun to appear [2] [3]. The technique creates an image that appears perspectively-correct from a specific viewing location. The camera is placed at this location during calibration. Any effects due to screen geometry are captured in the resulting map from camera to projector coordinates. Thus, the general technique of calibrating with a camera at the intended viewing position can support screens of arbitrary shape, given that the result will only appear correct from the intended viewpoint.

Other systems have used a vertex program to compute non-standard projections. Raskar [18] used a vertex program to compute a warp for a quadric display shape.

Our system differs from previous ones by allowing the creation of the projected immersive imagery with a single pass over the input geometry. We do not attempt to compute a warp and we do not use a cubemap. We avoid the resampling errors possible when using the

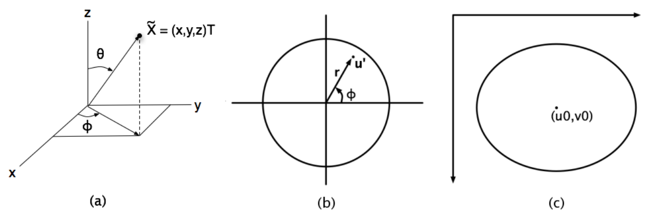

Figure 1: **(a) eye coordinates; (b) ideal camera image coordinates; (c) actual camera image coordinates**

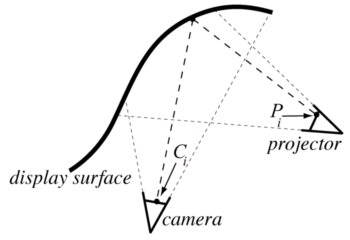

Figure 2: **Pixels on the camera and projector image planes for point on the display surface.**

typical two-pass (render to texture, then render textured geometry) algorithm. We use a texture to store what is essentially the mapping from eye coordinates to projector (screen) coordinates. A vertex program uses that texture to map vertices from eye coordinates to projector coordinates in a single pass, replacing the perspective projection with a non-parametric one derived from the display system geometry and projector optics. It therefore supports a wide range of possible configurations. Also, our technique requires neither expensive equipment nor specialized infrastructure.

3 System Overview

This section gives a brief overview of our immersive projection technique. It is comprised of two steps: (i) a one-time calibration step and (ii) a geometry transformation step. The one-time calibration step determines the correspondences between display pixels (as imaged by a camera) and projector pixels. We call this set of correspondences ”camera-projector correspondences.”

The geometry transformation step is applied at runtime to each vertex of the input geometry. It is implemented as single-pass algorithm in a vertex program.

These steps are described in more detail in the following sections.

4 Calibration

The calibration step determines a set of camera-projector correspondences. Camera pixels correspond to direction vectors in eye coordinates, once the camera’s intrinsic (lens) parameters are known. We use a fisheye lens to enable imaging of a display with a large field-of-view. By projecting a sequence of known patterns, and imaging them with the camera, correspondences between projector and camera pixels are found.

4.1 Fisheye Camera Calibration

Camera intrinsics are used to transform camera image coordinates to and from direction vectors. We use a fisheye lens to capture an extremely wide field-of-view in a single image.

Calibration of fisheye lenses is not supported in most off-the-shelf camera calibration software. We adopt the camera model of Bakstein and Pajdla [6]. Their imaging model assumes a radially-symmetric lens, and accounts for pixel aspect ratio and image center.

The transformation from world coordinates, X, to camera coordinates, $\tilde{X}$, is described by a rotation matrix R and vector T: $\tilde{X} = RX + T$

Let $\tilde{X} = [x, y, z]^T$. Then, let θ be the angle between the ray and the z-axis, and ϕ the angle between the x-axis and the vector $[x, y]^T$ (Figure 1(a)). Then, $\theta = \tan^{-1} \frac{\sqrt{x^2+y^2}}{z}$, and $\phi = \tan^{-1} \frac{y}{x}$ Now, the distance from the image center, r, of the projection is modeled as $r = a \tan \frac{\theta}{b} + c \sin \frac{\theta}{d}$ Let the location of the feature in the “ideal” camera image be $\mathbf{u}' = [r\cos\phi, r\sin\phi, 1]^T$ (Figure 1(b)). To account for image center, $[u_0, v_0]^T$, and pixel aspect ratio, β, in the position of the projection of the world point X onto the image plane, $\mathbf{u} = (u, v, 1)$, in “actual” image pixel coordinates (Figure 1(c)), $\mathbf{u} = K\mathbf{u}', where\ K = \begin{pmatrix} 1 & 0 & u_0 \\ 0 & \frac{1}{\beta} & v_0 \\ 0 & 0 & 1 \end{pmatrix}$ We express this fisheye transformation as $F(\tilde{X}) = \mathbf{u}$.

The camera parameters are the three rotation angles, a three-vector for the translation, the four parameters a, b, c, d for the non-linear term, β, and u_0, v_0.

Calibration uses an image of a calibration object with a number of point features at known world coordinate positions. Let N be the number of points, $\mathbf{X_i}$ the world coordinate position of the i-th point, $\tilde{u}_i$ the position of $\mathbf{X_i}$ observed in the camera, and $\mathbf{u_i}$ the position predicted by the parameters. The parameters are found by minimizing $\sum_{i=1}^{N} ||\tilde{\mathbf{u}}_\mathbf{i} - \mathbf{u_i}||$.

4.2 Camera-Projector Correspondences

The geometry transformation step requires a mapping from camera image coordinates to projector image coordinates. Our technique builds this mapping using camera-projector pixel correspondences. We use the Brown and Seales [8] method, but any comparable method will suffice. A projector projects known patterns of pixels onto the viewing surface. A camera is placed at the intended viewing location. There, it captures images of the projected patterns (structured light) on the display surface as an observer would see them. These camera images are correlated with the projected images to derive a set of correspondences between camera image pixel coordinates, C_i, to projector image pixel coordinates, P_i. Completion of this calibration step yields a set of image pixel coordinates pairs (C_i, P_i), that we refer to as camera-projector correspondences. Figure 2 shows the (C_i, P_i) pair for a point on the display surface. We interpolate this sparse set of correspondences to obtain a dense (per-pixel) mapping from camera pixels (essentially direction vectors in eye coordinates) to projector pixels. This dense mapping is stored in a texture and used at runtime to project vertices, replacing the usual perspective projection.

5 Geometry Transformation

This step maps input geometry from an application from eye coordinates to clip coordinates. For efficiency, we build a camera-to-projector coordinate map, M, and store it in a 2D texture.

5.1 Initialization

The camera-projector correspondences only map a sparse subset of camera image pixel coordinates to projector image pixel coordinates. The remaining camera pixels are mapped by interpolating from the nearest known correspondences.

Our technique stores M in a floating point texture, where each texel represents a camera pixel and the texel’s red and green channels’ values are normalized projector pixel coordinates. The camera-projector correspondences form a distorted grid in camera image space that can be treated as a set of triangles. The position of each triangle vertex is the camera coordinate position C_i of a

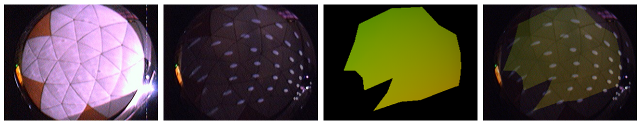
Figure 3: **From left to right: (a) Display surface as captured by camera. (b) Structured light patterns. (c) M map as texture. (d) Overlays of the structured light patterns and the texture map.**

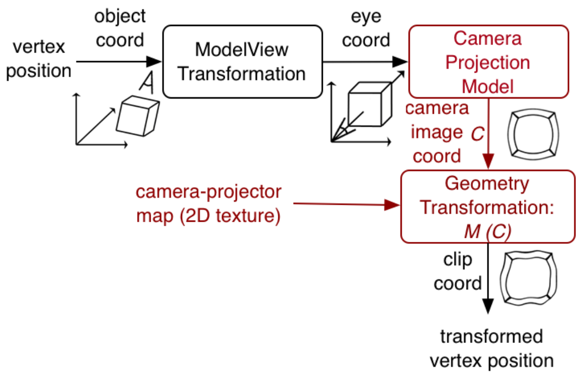

Figure 4: **Vertex processing pipeline, with our customized processing in red replacing the usual perspective transformation.**

correspondence. The color of the vertex is the projector coordinate position, P_i, of that correspondence. We create the texture by rendering these triangles directly to the texture memory and configure the graphics pipeline to automatically interpolate the vertex colors across the interior pixels of the triangles.

The resulting M map will not completely fill the camera image plane, but it will cover the maximum area given by the camera-projector correspondences. Figure 3 shows the display surface, a set of the structured light patterns captured by the camera, and the resulting M map texture from one of our setups. The irregular border is due to the low resolution of our current structured light system.

When reading a texel's color from the texture, extra information is needed to be able to determine if that texel was filled or if it is simply background color. To help make this distinction, we set the vertex color's blue channel to a known constant value.

The texture-based representation of the mapping only needs to be generated once for a given projector arrangement. Once the texture has been generated, it provides a fast method to map from camera image coordinates to projector image coordinates.

5.2 Vertex Program

A vertex program allows for custom per-vertex processing and our technique takes advantage of it to implement the geometry transformation. It's the vertex program's responsibility to transform the incoming vertex from the application into homogenous clip coordinates before passing it down the graphics pipeline for further processing. So, our vertex program first transforms the incoming vertex, V, to eye coordinates using the modelview transformation. Next, the vertex program replaces the perspective transformation that usually follows with our custom processing. The calibrating camera's parameters are applied to the eye coordinate position of the vertex to compute its camera coordinate position, C.

However, camera image coordinates must be mapped to projector coordinates in order for the projector's output to appear perspectively correct to the observer. Thus, we apply an additional transformation by using the vertex's camera image coordinates to index into the M map texture. Figure 4 shows the vertex processing pipeline with our customized processing in red.

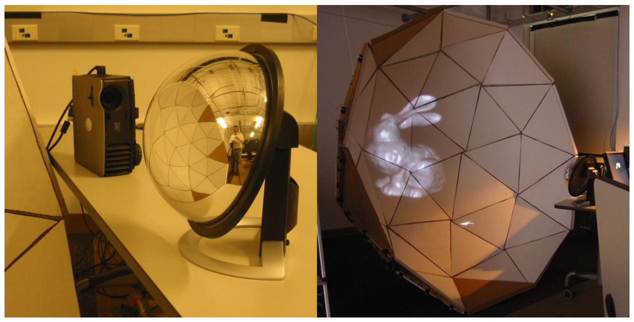
Figure 5: **(a) A configuration for a geodesic dome screen. (b) Projection using our system.**

Texture coordinates range from 0.0 to 1.0. Depending on the projection model, it may be necessary to shift and normalize the camera coordinates into that range. The texel color read represents the normalized projector pixel coordinates $P = M(C)$.

Finally, to determine the clip coordinates of the projected vertex, the z-coordinate is set to be the distance to the vertex in eye coordinates, and the w-coordinate is set to 1.0. The vertex coordinate output from the vertex program is $V = [P_x, P_y, distance(modelview(V)), 1.0]$. The pipeline's subsequent division by w will have no efffect. Points lying along the same viewing direction will have the same x and y coordinates, but different zs, allowing for correct z-buffering.

One limitation of our approach is that the fixed pipeline rasterization of line, triangle, and polygon primitives assumes they were transformed by a line-preserving transformation. Under certain situations, this condition does not hold and the geometry will be rendered incorrectly if line or triangle primitives are used. This is however only a practical concern when extreme close-up viewpoints are chosen, or when models are coarsely tesselated. There is also an issue with clipping primitives that are partially outside the extent of the M map. Point-based rendering can be used to overcome both of these problems.

We currently render vertices as hardware-accelerated point primitives and therefore we require densely sampled models to avoid gaps in the rendered image. However, our method can be extended to support alternative point-based rendering techniques [12], such as hierarchical splatting. Our system optionally uses triangle primitives for rendering: used with moderately tessellated models, this yields images of good quality for typical views.

6 IMPLEMENTATION

This section describes the components in our immersive projection implementation.

Physically, the system uses a Dell 1850 XGA projector reflected off an inexpensive, approximately hemispherical security mirror to spread the light in a wide field of view on the display surface, thus creating an immersive projection. Figure 5 shows one of our geodesic dome configurations. A dual-core 2GHz Xeon PC equipped with NVIDIA's GeForce 8800 GTS video card runs the application program and generates output images that are sent to the projector. The calibration step uses a low cost, low resolution, Unibrain Fire-i firewire camera with a relatively inexpensive fisheye lens from OmniTech Robotics.

The camera was calibrated using a 3D calibration object (Figure 6(a)). The calibration object is made from three pieces of plywood, arranged orthogonally. A checkerboard of 1.5-inch squares is affixed to the surface. This forms a natural world coordinate system with the innermost corner of the object as the origin.

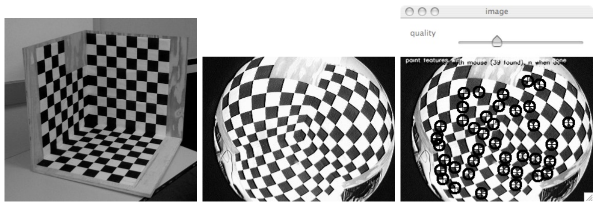

Figure 6: **From left to right: (a) Calibration object. (b) Calibration object captured by fisheye lens. (c) Feature locations.**

The camera is placed within the calibration object, so as to fill as much of the camera's field of view as possible, and a image taken (Figure 6(b)). Our feature locating software uses the the OpenCV library's [1] corner detector to locate the strongest corner features with sub-pixel accuracy. The corner detector's strength threshold is controlled by the user via a slider. The world position of each corner is determined by manual inspection of the image. We used the 39 features marked in Figure 6(c).

The camera parameters R, T, a, b, c, d, u_0, v_0, and β were determined using the numerical minimization function NMinimize in Mathematica 6 [20].

7 Future Work

As previously mentioned, point-based rendering overcomes issues that arise from the rasterization implemented in the fixed graphics pipeline. Another possible solution to the rasterization issues is to tesslate the input geometry. The latest graphics cards feature geometry shader support that could be used perform this adaptively on the GPU. We are pursuing alternatives in point-based rendering and geometry shader tesselation.

The use of bilinearly-interpolated texel colors for the screen coordinates of projected vertices introduces artifacts (small ripples) in the image that we are still investigating as of this writing.

Stereo could be supported, albeit at greater equipment cost, using separate calibrations for each eye position. For passive stereo, the two projectors could be casually positioned. Active stereo requires expensive shutter glasses.

A possible modification to the texture-based transformation we implement would be to use a cubemap texture instead of a 2D texture. This would allow direct indexing of the texture with the eye coordinates of the vertex, bypassing the need to perform the fisheye projection to the camera image plane.

We could also eliminate the need for the parameterized fisheye projection calculation in the vertex program if we create the 2D texture using a simpler spherical projection, and use the parameterized fisheye projection's inverse to transform the camera coordinates used to create the texture. That is, if the simple projection is $S : \mathbb{R}^3 \to \mathbb{R}^2$, we would use $S(F^{-1}(C_i))$ as the triangle vertex position for correspondence i when creating the texture, and $S(\widetilde{X})$ as the index into the texture at runtime.

8 Conclusion

We have presented a technique for creating wide field-of-view images from input geometry using a GPU-based, single-pass rendering algorithm. Our system has several advantages. Existing applications can be easily modified to use our vertex program during rendering. There are no special requirements for the geometry of the display surface. Any available diffuse surface in a room, such as the floor, wall or ceiling, will suffice. Thus any room can be outfitted with this immersive projection system. The components only need to be casually aligned, unlike other existing systems that require precise alignment of projectors and display surfaces. Thus, our system is easier to build and maintain. Our system uses affordable components. Most institutions already have projectors and suitable computers. All that's additionally needed to build our system is a low cost camera with fisheye lens and a spherical security mirror; a total cost of approximately \$350. Thus, our system does not require an sizable investment to build, and is suitable for low-budget situations such as schools, classrooms, and the home.

Acknowledgments: Thanks to Ifeyinwa Okoye and Robert Sajan for their work on the correspondence finder, to Michael Leung for equipment support, and to the anonymous reviewers for their helpful comments and suggestions.

References

[1] http://sourceforge.net/projects/opencvlibrary/.

[2] http://www.mersive.com.

[3] http://www.scalabledisplay.com.

[4] http://www.stellarium.com.

[5] M. Ashdown, M. Flagg, R. Sukthankar, and J. M. Rehg. A flexible projector-camera system for multi-planar displays. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR04)*, 2004.

[6] H. Bakstein and T. Pajdla. Calibration of a fish eye lens with field of view larger than 180. In *Proceedings of the CVWW 2002*, pages 276–285, February 2002.

[7] P. Bourke. Spherical mirror: a new approach to hemispherical dome projection. In *GRAPHITE '05: Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 281–284, New York, NY, USA, 2005. ACM Press.

[8] M. S. Brown and W. B. Seales. Low-cost and easily constructed large format display system. Technical Report HKUST TR-CS-01-02, Hong Kong University of Science and Technology, 2001.

[9] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti. Surround-screen projection-based virtual reality: the design and implementation of the cave. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 135–142, New York, NY, USA, 1993. ACM Press.

[10] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart. The cave: Audio visual experience automatic virtual environment. *Communications of the ACM*, 35(6):65–72, 1992.

[11] N. Greene. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, 6(11), November 1986.

[12] M. Gross and H. Pfister. *Point-Based Graphics*. Elsevier, 2007.

[13] M. Harville, B. Culbertson, I. Sobel, D. Gelb, A. Fitzhugh, and D. Tanguay. Practical methods for geometric and photometric correction of tiled projector. In *CVPRW '06: Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop*, page 5, Washington, DC, USA, 2006. IEEE Computer Society.

[14] http://www.astronomyteacher.com/.

[15] C. Jaynes, W. B. Seales, K. Calvert, Z. Fei, and J. Griffioen. The metaverse: a networked collection of inexpensive, self-configuring, immersive environments. In *EGVE '03: Proceedings of the workshop on Virtual environments 2003*, pages 115–124, New York, NY, USA, 2003. ACM Press.

[16] A. Raij, G. Gill, A. Majumder, H. Towles, and H. Fuchs. Pixelflex2: A comprehensive, automatic, casually-aligned multi-projector display. In *IEEE International Workshop on Projector-Camera Systems*, October 2003.

[17] R. Raskar. Immersive planar display using roughly aligned projectors. In *IEEE Virtual Reality*, March 2000.

[18] R. Raskar, J. vanBaar, and T. Willwacher. Quadric transfer for immersive curved display. In *EUROGRAPHICS 2004*, 2004.

[19] J.-P. Tardif, S. Roy, and M. Trudeau. Multi-projectors for arbitrary surfaces without explicit calibration nor reconstruction. In *Fourth International Conference on 3-D Digital Imaging and Modeling*, October 2003.

[20] Wolfram Research, Inc. *Mathematica*. Wolfram Research, Inc., version 6 edition, 2007.