Continual Surface-Based Multi-Projector Blending for Moving Objects

Peter Lincoln Greg Welch Henry Fuchs

The University *of* North Carolina *at* Chapel Hill
Department of Computer Science*

ABSTRACT

We introduce a general technique for blending imagery from multiple projectors on a tracked, moving, non-planar object. Our technique continuously computes visibility of pixels over the surfaces of the object and dynamically computes the per-pixel weights for each projector. This approach supports smooth transitions between areas of the object illuminated by different number of projectors, down to the illumination contribution of individual pixels within each polygon. To achieve real-time performance, we take advantage of graphics hardware, implementing much of the technique with a custom dynamic blending shader program within the GPU associated with each projector. We demonstrate the technique with some tracked objects being illuminated by three projectors.

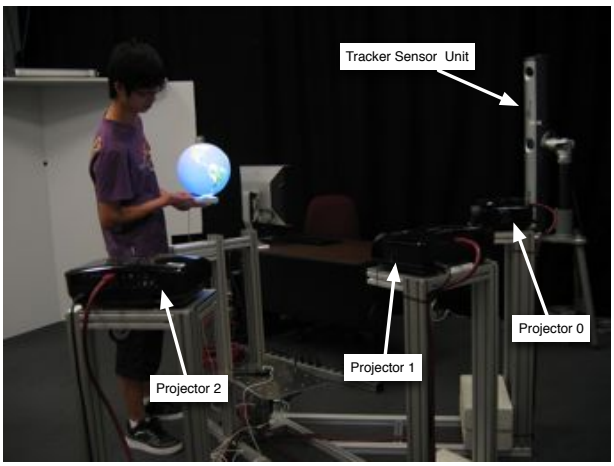
Index Terms: H.5.1 [Multimedia Information Systems]: Animations—Artificial, augmented, and virtual realities I.3.7 [Computer Graphics]: Three Dimensional Graphics and Realism—Virtual Reality; I.3.8 [Computer Graphics]: Applications;

1 INTRODUCTION

One long-term dream of VR has been to integrate synthetic objects with a user's immediate surroundings and to enable the user to examine, manipulate, and change aspects of those synthetic objects. One part of this dream that is close to being realized is the control of the appearance of a physical object by projection techniques. To achieve this effect, a projector need only render a virtual model of the object from the point of view of the projector. In order to allow a user to view the physical object from a wide range of viewing positions, multiple projectors need to be used, each projecting from sufficiently different angles to illuminate all the visible parts of the object. To achieve a consistent appearance, the intensity from any projector needs to be attenuated in surface regions illuminated by multiple projectors. This blending of multiple projected images has to be performed precisely, and the shape and pose of the object needs to be accurately known, because even slight errors in illumination patterns can result in obvious and disturbing artifacts. If the user also wants to move and manipulate the object, the required real-time computation increases significantly as the attenuation mask of every projector could change for every pixel. However, we believe that the resulting natural object-manipulation capability will be widely useful if it can be achieved without creating disturbing visual artifacts on the surface of the object. It is toward this end that the efforts of this paper are aimed. This paper presents preliminary work. Future efforts in geometric and latency analyses should lead to further reductions in the remaining artifacts that appear during object motion.

2 RELATED WORK

Historically, the first multi-projector systems were designed for static surfaces. Perhaps the simplest approach is that of mechanical blending via *aperture modulation* [6]. However the majority of work in this area has been aimed at software methods for blending imagery from multiple projectors [10, 11, 13, 3, 7, 2, 12, **?**, **?**]. These approaches, which usually generate per-projector blending masks as a non-interactive pre-processing step, have demonstrated compelling results, but again for static surfaces. There has been some work aimed at projecting onto moving surfaces, but prior work has been limited to using only a single projector, recomputing the blending masks only when the object has stopped moving, or dynamically assigning a single projector to each planar facet of the surface [1, 4, **?**]. The work in [4] is particularly interesting in that the author uses continual camera imagery of the surface (and whatever is being projected onto it) to *track* the object's surface. However, the approach is still limited in terms of its multi-projector blending in that it only supports piece-wise planar objects, where each complete facet is illuminated by a single projector. We are unaware of any work aimed at real-time, on-line blending of multi-projector imagery onto non-planar moving surfaces.



(a) System layout with ball object

(b) View near Projector 1 of hand-held head object

(c) View near Projector 0 of hand-held head object

Figure 1: Our demonstration system, in (a), consists of three projectors, a tracking system, and a tracked physical object. Parts (b) and (c) are two photos of a hand-held head object from two different viewing positions, recorded at the same time.

*e-mail: {plincoln, welch, fuchs}@cs.unc.edu, phone: 919-962-1700, fax: 919-962-1799

Figure 2: Two different target objects, each rigidly mounted to a opto-electric tracking tool.

3 METHOD

In this section we describe our procedure for enable multi-projector blending. We begin by describing the one-time operations required to calibrate the system of projectors, tracker, and objects. We continue by describing the rendering procedure necessary for generating the real-time masks and performing the blending.

3.1 Calibration

Our system, presented in Figure 1a makes use of a three projectors, a NDI Optotrak opto-electric tracker [8], and a target object, and it requires that these components be calibrated to a common coordinate reference frame; as a result, each of these components must be calibrated. These components only require a one-time calibration, as long as the physical parameters remain unchanged, the calibrations produced can be reused.

3.1.1 Projectors and Tracker Calibration

The blending procedure requires that both the geometric and photometric properties of the projectors be calibrated. In order to determine the geometric properties (intrinsic and extrinsic components) of the projectors, we use a custom application that utilizes the OpenCV library [9]. The "camera" calibration methods of the OpenCV library require a series of images containing checkerboards of known sizes in a variety of poses. By using a physical checkerboard and matching the projector imagery to that checkerboard, we collect a series of virtual "camera" images that are suitable for use in our OpenCV-based application for computing the intrinsic properties of the projector. By also collecting the 3D coordinates of each interior corner of the checkerboard with the tracker's probe tool, OpenCV can also compute the extrinsic properties of the projectors in the reference frame of the tracking system. Using this calibration technique typically results in a reprojection error on the order of one pixel or less. In order to compute the projectors' photometric properties, we use a standard procedure that uses a series of projected solid images with intensities in $[0..255]$ and captures these image sequences with a linear-response curve camera. Normalizing the relative projected intensities yields both a response curve and an inverse response curve for the target projector.

3.1.2 Object Calibration

The blending procedure also requires knowledge of the rendering surface of the target object in both its structure and its relationship to its tracking tool. The methods of constructing a model of an object are beyond the scope of this paper but can include (for complex objects) using a 3D scanner or modeling software or (for primitive geometric objects) computing the model directly from a few key points. For instance, the radius and center of a sphere can be directly computed from four points.

Computing each object's static relationship to its tracking tool requires that the object be rigidly mounted to the tracking tool, see Figure 2. For any pair of tracking tool pose and simultaneously captured probe point, it is possible to compute that probe point's locating in the tracking tool's reference frame. We collect the calibration points using the NDI Optotrak tracker, which is capable of tracking objects with a maximum accuracy of 0.1 mm. After collecting a small series of probe points in the tracking tool's reference frame and matching these points to their corresponding points in the object's model, we run an automatic optimization function over the data to compute an homogeneous 4×4 transformation matrix between the tracking tool and the object's model. In order to compute the tracker space location of a vertex on the model, we can multiply the tracking tool's live pose, the optimized transformation matrix, and the vertex location.

3.2 Rendering

The multi-projector blending procedure is a multi-pass method implemented in OpenGL using the fixed-function pipeline, custom vertex and pixel shaders, and frame-buffer objects (FBOs). The algorithm consists of three stages: each projector rendering module first computes the min-depth maps for all projectors, then generates the blending masks for the current projector, and finally performs an optional remap operation to correct for distortion. As each module operates independently and assumes that the other modules cooperate, they all execute the same procedure. The procedure described below is that of a single module of the set of n projector modules.

The first stage is the simplest and operates similarly to generating shadow masks. It renders each projector's perspective of the scene to n separate FBOs; these FBOs save only the depth component to usable textures, the remaining components are discarded. The n depth component textures will be used in the next stage.

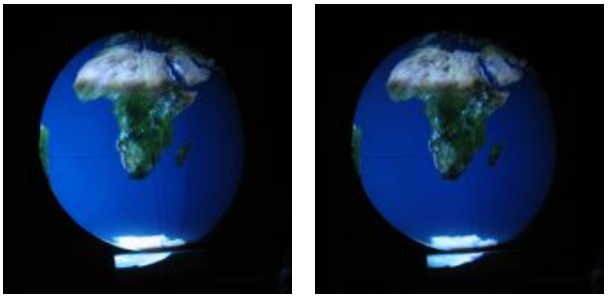
The second stage both generates the blending masks and performs the blending operation. This stage uses both a vertex and pixel shader and renders its result to a single color-only FBO. The vertex shader uses each of the n projectors Model-View-Projection matrices to compute the 2D+Depth coordinate for each vertex from each of the n projector's viewpoints. It also uses the current projector's Model-View matrix to compute the 3D tracker-space position and normal for each vertex. Finally, the vertex shader computes the texture coordinates and passes through the vertex color for each vertex in a similar fashion as the fixed-function pipeline. This stage's pixel shader performs the actual mask generation and blending, involving two parts: depth tests and angle comparisons.

The second stage's pixel shader's depth test acts as a validator. For each actively rastered pixel in the current projector, the pixel shader compares the active projector's distance to the surface of the object with the minimum depth computed in the first stage. If the two depths are sufficiently equal to a fixed tolerance, then the shader assumes that the pixel on the objects surface is unoccluded, else the shader is aborted with a rendered color of $(0,0,0,0)$. This test is repeated for the other $n-1$ projectors, but for the other projectors, a failure does not result in aborting the rest of the shader's computation.

The second stage's pixel shader's angle comparison performs the actual blending. For each unoccluded projector that passed the depth test, a blending factor is computed. A naïve blending factor would be $1/u$, where u is the number of unoccluded projectors; however, this does not take into account several factors presented in Section 5. A better solution computes the dot product between the pixel's unit surface normal, $\mathbf{n}$, and the unit direction from the pixel's location and the center-of-projection of the unoccluded projector, $\mathbf{p}$. This dot-product is clamped to $[0,1]$ to ensure only locally front-facing normals are considered. However, simply using $\mathbf{n} \cdot \mathbf{p}$ results in very large perceptively detectable overlap regions, which can accentuate errors. In order to skew the masks towards favoring locally face-on projectors, and thus reduce the size of the overlap regions, we raise the dot product to a power; thus the blending factor is $(\mathbf{n} \cdot \mathbf{p})^k$. Increasing the value of k reduces the size of the perceptively visible overlap regions. In practice, we have found

(a) $k = 1$ (b) $k = 7$

Figure 3: Full-screen blending masks of used by the center projector for the sphere object given different values of k in the blending mask function, $(\mathbf{n} \cdot \mathbf{p})^k$.



(a) Blending Disabled (b) Blending Enabled

Figure 4: Photos of the same globe model on a white-painted basketball, render both with (b) and without (a) projector blending. Note in (a) the sharp step-change in the color of the ocean to the west and east of southern Africa; these are places where the number of projectors illuminate the surface changes.

that a high value of k (e.g. $k = 7$) works well; the effects of varying the value of k are presented in Figure 3. The final value of the mask is computed as the ratio of the current projectors's blending factor to the sum of all unoccluded projector's blending factors. In order to account for the gamma response curves of the projectors, the shader computes the projector's response to color component of the texture-mapped and color-modulated un-masked pixel value, applies the mask factor, and finally computes the inverse response to determine what value to select for that projector for that pixel.

The final stage performs radial distortion correction on the result of the second stage. This ensures that radial distortion effects intrinsically present in the projectors are mitigated. In the unlikely case that the projectors in use do not have significant radial distortion effects, this stage could be skipped or implemented as a pass-thru stage; however, in order to minimize errors, it should be executed. CPU based radial distortion correction, like that in the OpenCV library, typically involves generating a lookup table of input pixel coordinates for each output pixel. GPU shader-based correction techniques can use essentially the same procedure; we use a lookup table and the color-texture result of the second stage to produce the final output that is written to the screen buffer.

4 Results

Some results using our multi-projector blending technique are shown in Figures 4 and 5 and in the attached video. Some analysis of these figures are presented in Section 5.

Our technique, as implemented in our test application, runs in real-time. The test system, which drives three projectors, is comprised of a single 4-core, 8-thread Xeon processor running at 2.26 GHz, 3 GB of RAM, and two NVIDIA GeForce 9800 GT graphics cards. When rendering the ball model, which is a 100x100 spherical grid model, we achieve around 70 fps for each projector. When rendering the head model, which has almost 15k polygons, we achieve around 50 fps for each projector. The shaders and implementation are compatible with typical optimizations such as display lists. Each render loop for each projector operates as an independent, unsynchronized thread. In theory the per-projector depth maps could be shared among the projectors as they are essentially identical. This would eliminate duplicated work in the first stage of the blending pixel shader. While the $O(n^2)$ (n-depth maps computed by n-projectors), repeated-work depth-map computation is slight for three projectors, an alternative may be needed when scaling up the system to more projectors.

5 Discussion

Using multiple projectors can increase the coverage over the surface of the target object. However, without taking special considerations for overlap regions, using multiple projectors can create noticeable bands at the boundaries of projector regions, as seen in Figure 4a, in the first row of Figure 5, and in the attached video. By using our multi-projector blending technique, the appearance of these bands and self-shadowing regions are significantly mitigated; this is evident by comparing the edges in the ocean to the immediate west and to the east of southern Africa in Figure 4 and the nose's shadow regions in Figure 5. Furthermore, by using a large value of k in the blending mask value computation (see Section 3.2), the relative contributions of each of the projectors are skewed in favor of the most face-on projector.

Optimizing the blending mask in favor the most-orthogonally located projector to the surface point has the primary effect of reducing blurring on the object's surface. This blurring, or double-imaging, on the surface typically arises from errors in projector or object calibrations, object model construction, and/or tracker pose, all of which contribute to produce misaligned imagery. While the pixels as computed in OpenGL as points, they have a very noticeable size when projected. As a result, the scale of the error's appearance depends on the relative difference in angles between the surface normal and direction from the surface point to the center of projection (these are the same vectors used in the blending mask computation); as the angle becomes more oblique, the pixel covers a larger surface area. Similarly, if the pixel is improperly located on the projector's image plane, due to calibration or modeling error, the degree to which that error manifests on the surface increases with the obliqueness of the relative angles. Where a 1 mm wide, well-aligned pixel (at the surface) appears as an in-place 1 mm wide region on a face-on surface, that same pixel on a relative 60° surface at 1 mm of error along a direction parallel to the projector's image plane becomes a 2 mm wide pixel with a 2 mm surface offset from its proper location. Errors in different directions by other projectors can increase the separation of what would otherwise be identically projected colors. Weighting the masks to favor face-on projectors reduces the blurring not by repairing the calibration, tracking, or modeling errors, but by reducing the size of equal-mask intensity overlap regions. The effect of these direction-weighted masks can be seen in rows 3-5 of Figure 5; despite the fact that two subsets of projectors can see each side of the model's nose, the masks favor the left and right projectors over the center projector.

6 Conclusions

We introduced a real-time surface-based multi-projector blending algorithm and presented a prototype implementation of that algorithm. It improves a surface's appearance across projector boundaries and reduces some of the effects of errors by reducing overlap regions and by favoring face-on projectors. The real-time capabilities of the algorithm and prototype enable this improved surface appearance to persist despite continuously moving the target object. However, there are still some problems to be solved in this domain. The errors, while reduced, are still present in the overlap regions, and can be caused by both static and dynamic error: static

error in the form of calibration, modeling, and tracking errors, and dynamic error in the form of tracking latency. Each of these errors can degrade the quality of the blended result. Thus future work in this area primarily consists of developing additional techniques for augmenting the presented technique in ways to handle the presence of static and dynamic errors to better provide an improved appearance.

ACKNOWLEDGEMENTS

We thank Herman Towles for his insightful suggestions, technical help, and advice. John Thomas provided mechanical and electronic engineering assistance. Ryan Schubert and Feng Zheng assisted with system calibration and video capture. This work was supported in part by the Office of Naval Research (award N00014-09-1-0813, "3D Display and Capture of Humans for Live-Virtual Training," Dr. Roy Stripling, Program Manager), and the National Science Foundation (award CNS-0751187).

REFERENCES

[1] D. Bandyopadhyay, R. Raskar, and H. Fuchs. Dynamic shader lamps: Painting on movable objects. In *Proc. IEEE and ACM international Symposium on Augmented Reality (ISAR '01)*, pages 207–216, New York, NY, USA, October 2001. IEEE Computer Society.

[2] E. S. Bhasker, P. Sinha, and A. Majumder. Asynchronous distributed calibration for scalable and reconfigurable multi-projector displays. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1101–1108, 2006.

[3] C. Jaynes, R. M. Steele, and S. Webb. Rapidly deployable multi-projector immersive displays. *Presence: Teleoper. Virtual Environ.*, 14(5):501–510, 2005.

[4] T. Johnson. *A Cooperative Approach to Continuous Calibration in Multi-Projector Displays*. PhD thesis, UNC-Chapel Hill, November 2009.

[5] W. Lee and J. Park. Augmented foam: A tangible augmented reality for product design. In *Proceedings of the 4th IEEE/ACM International Symposium on Mixed and Augmented Reality*, page 109. IEEE Computer Society, 2005.

[6] K. Li, H. Chen, Y. Chen, D. W. Clark, P. Cook, S. Damianakis, G. Essl, A. Finkelstein, T. Funkhouser, T. Housel, A. Klein, Z. Liu, E. Praun, R. Samanta, B. Shedd, J. P. Singh, G. Tzanetakis, and J. Zheng. Building and using a scalable display wall system. *IEEE Computer Graphics & Applications*, 20(4), 2000. ISSN 0272-1716.

[7] A. Majumder. *A Practical Framework to Achieve Perceptually Seamless Multi-Projector Displays*. Ph.d., University of North Carolina at Chapel Hill, 2003.

[8] Northern Digitial Inc. NDI: Optotrak Certus Motion Capture System - Research-Grade Motion Capture System for Life Sciences. `http://www.ndigital.com/lifesciences/certus-motioncapturesystem.php`, March 2009.

[9] OpenCV. The OpenCV library. `http://sourceforge.net/projects/opencvlibrary/`, May 2009.

[10] R. Raskar, G. Welch, and W.-C. Chen. Table-top spatially-augmented reality: Bringing physical models to life with projected imagery. In *IWAR '99: Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, page 64, Washington, DC, USA, 1999. IEEE Computer Society.

[11] R. Raskar, G. Welch, K.-L. Low, and D. Bandyopadhyay. Shader lamps: Animating real objects with image-based illumination. In *Eurographics Workshop on Rendering*, June 2001.

[12] B. Sajadi and A. Majumder. Auto-calibration of cylindrical multi-projector systems. pages 155 –162, mar. 2010.

[13] R. J. Surati. *Scalable self-calibrating display technology for seamless large-scale displays*. PhD thesis, MIT Department of Electrical Engineering and Computer Science, 1999. Supervisor-Thomas F. Knight, Jr.

Figure 5: A couple of viewpoints of a head model rendered onto a white-Styrofoam head. Each column is a different viewpoint of the same scene. Rows are (from top-to-bottom) blending disabled; blending enabled; individual contributions from projector 0, 1, and 2; and projector overlap counts. The projector overlap counts (bottom row) are color-coded as follows: red for regions with only one projector contributing, green for two projectors, and blue for three projectors.