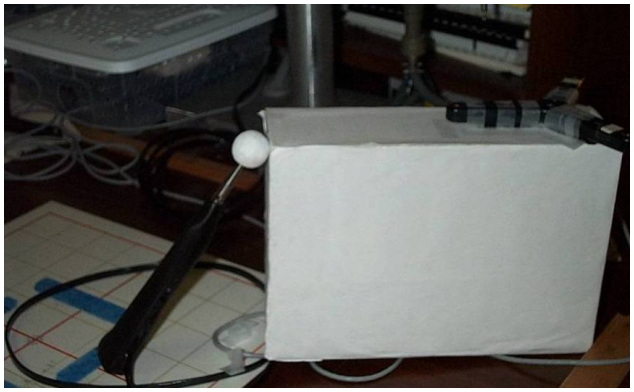
Dynamic Spatially Augmented 3D Painting

Deepak Bandyopadhyay Ramesh Raskar Andrei State Henry Fuchs

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



Abstract

We present techniques for creating a Spatially Augmented Reality for moving polygonal objects in an indoor environment, and a system that uses these techniques for painting in 3D on these objects in real time. By tracking the object and projecting on its different surfaces with two or more projectors, our system creates a dynamic spatially augmented version of the object that is registered with the real object as it moves. Along with modification of the lighting and material properties, our application allows real-time 3D painting on the surface of the real object. Even though extensive user studies for this application have not yet been done, the application is new and compelling enough even at the proof of concept stage to warrant further exploration into this area to solve all of the problems encountered, particularly into the problem of real-time blending of the projected images in the areas of projector overlap.

Keywords: Applications, 3D Painting, Spatially Augmented Reality, Projector, Tracking, Human-Computer Interface

Additional Keywords: Augmented Reality, Dynamic, Virtual Reality

1 Introduction

Spatially augmented reality ([13]) is a technique in which we introduce computer-generated imagery on top of real-world objects, that lets multiple independent viewers see an augmented version of their surroundings in stereo without the need for head tracking or head-mounted displays.

The **Shader Lamps** technique ([11]) shows a way of altering the color, lighting, diffuse and specular properties of static objects by projecting on them using image-based illumination. What if these objects could be allowed to move in arbitrary ways, retaining their current state of shading as they did so? For the purposes of interactive shading of the objects, a painting interface seems ideal. So in such a system I'd be able to draw something on an object's faces, and then pass it to you to examine and add to it. This was the vision that guided the starting of this work. It was accomplished as a course project in the UNC course on Exploring Virtual Worlds offered in the spring of 2000.

The rest of this document is organized as follows. Section 2 details the goals and objectives that we started out with. Section 3 deals with the actual proof-of-concept implementation, the issues that arose and the decisions made in the development of this implementation. Section 4 summarizes the initial reactions of users of the paint system, along with a visual sample of what the system is currently capable of. Section 5 talks about some of the issues that currently need to be addressed and how we are planning to deal with them in the short to medium term, and section 6 concludes with a summary and an indication of the exciting challenges and future possibilities for research in this area.

2 Objectives

Initially our technological objective was to create an application for painting on a moving object in a virtual environment using projected imagery. This was to be accomplished in two distinct stages:

1. Painting on a static real object using a hand-held tracker as a paintbrush, and a set of projectors as a display device for projecting color patterns onto the faces of the object.

2. Letting the object move, and the color patterns stay registered on its faces and move with it. This would create a truly dynamic environment.

In the current system, we take these objectives further by fusing the two and making moving and painting work together simultaneously, so that one could paint on a *moving* object and have the paint appear in the right place.

3 Implementation

3.1 System details

The modeling and rendering code in our system runs on a single pipe of an SGI InfiniteReality2[10] engine, though it is fully

portable to a PC platform, which we have demonstrated. The display is on Sony LCD projectors. The tracker we use currently has drivers only for the SGI platform, so on the PC version of the system we are investigating the use of alternate trackers and even camera-based tracking.

3.2 Issues

Some of the major problems that were dealt with were as follows:

- **Suitability** of any real object for being projected and painted on.
- **Setup and Calibration** of the projectors to display in world coordinates.
- **Modeling** of the real object into 3D geometry and texture coordinates.
- **Display and Registration** of the projections onto the object.
- **Tracking and Updates** of the real object's position and orientation as the user moves it around.
- **Lighting and Material** properties of the object, and how to modify them with image-based illumination
- **3D Painting** with brush functions and mapping from 3D points to texture coordinates
- **Rendering**, the actual projection step from two projectors, updated simultaneously for all-around viewing of the object.

3.3 Choice of Object

The ideal object must be *dull* (not specular) so that we may project any arbitrary illumination onto it. It must be *neutral* (not dark or colored) so that we can get a wider range of colors by projection. It should be lightweight (for easy manipulation) and readily available. The proof of concept can be shown even with an object of small polygon count, though the system will work with medium-sized polygon count objects as well. *Symmetry* in the object is to be avoided, though, so that the registration of a particular feature onto a particular polygon is discernible as all polygons do not look alike.

We chose a cuboid (made from a cardboard box) as it satisfies all the above properties and is sufficiently simple to model as a first object without sacrificing any of the power of the underlying techniques. So for example we can move it around arbitrarily and it looks unique at each orientation; and we can paint on its faces, or around the corners and edges so that the colors are shared between faces.

3.4 Tracking and Coordinates

We chose the area used for the Ultrasound Augmented Reality experiment setup for setting up our system, as it provides a large relatively flat surface along with an optical tracker (the Flashpoint^{TM} 5000 from Image Guided Technologies Inc., Denver, Colorado) that has multiple sensors with active LEDS that can be tracked simultaneously.

We attached a sensor (the *rig*) with three LEDs onto the object being tracked, and the current position and orientation of the sensor in tracker cordinates was computed by the tracker driver routines from the positions of the LEDs. Another sensor (the *pointer*) with two LEDs served as the paintbrush; the position of its tip was computed by extending the straight line joining the two LEDs forward by a fixed distance. This sensor had only five degrees of freedom as it could accurately compute the position of the tip and the pitch and roll components of the sensor's orientation, but not the *twist* component of rotation about its longitudinal axis. This was not perceived as a major problem due to the way the brush was modeled (see Section blahblah).

Another simplification we make is that the world coordinate frame is the same as as the tracker coordinate frame, whereas the object coordinates are specified in the coordinate frame of a sensor that is rigidly attached to the object. This allows us to read the matrix that gives the sensor position and orientation in tracker coordinates, multiply it with each point on the object (a constant in object coordinates) and get the coordinates at which the point is to be rendered directly.

3.5 Projector Setup and Calibration

Two projectors are mounted in the ceiling facing each other . The projectors tilt downwards and their field of view and zoom are set to cover the tracking area, and have a large overlap within the tracking area to ensure that all faces of the object are illuminated by at least one projector while it is being tracked.

Each projector needs to be calibrated so that it can draw correctly in world coordinates. For a static world, finding the internal parameters and the rigid transformation between the coordinate system of the world and the projector is enough. Raskar et al in [15] solve this problem by taking a set of fiducials with known 3D locations on the *physical object* and then finding the corresponding projector pixels that illuminate them, and solving for the 3x4 projection matrix correct to a constant scale factor, which is then decomposed to yield the internal and external parameters of the projector.

We modify this method by adding the stipulation that the calibration points span the overlap of the projection and tracking volumes, and are roughly uniformly distributed across this intersection volume. This ensures that the calibration is good no matter where and in which orientation we place the object.

Now the tracked object can be drawn directly by concatenating the internal and the external parameters of the projector. The rendering process uses the same internal and external parameters, so that the projected images are registered with the physical objects.

3.6 Registration

First, for the case of *one* projector, registration means that we can draw with some precision in world coordinates. Static registration is improved by improving the calibration, modeling the object more accurately and more accurate tracking. Dynamic temporal registration (for the image to stay registered as the object moves) needs accurate as well as fast tracking, along with some **prediction** to offset the effect of tracker latency.

Our application is more sensitive to tracking latency than conventional VR applications, as the rendered scene is observed together with the real environment. In this situation, tracker latency and error can translate into dynamic and static registration error, respectively. Dynamic registration error causes the moving user to perceive shearing in the rendered scene, and these factors destroy the illusion of presence.

Future versions of the system may use an optical, ceiling tracker such as the 3rdTech HiBall [18, 19]. This tracker samples over one thousand readings per second, has a high static tracking accuracy (low noise) as well as a low enough latency. Together with techniques for prediction[1], this should solve some problems with latency for the brush. For tracking the object(s) being painted on, the short term solution lies with multiple sensors connected to the same tracker coordinate frame, one to each object and preferably wireless. For the present Flashpoint LED-based optical system, this would mean defining a custom tracker for each object by planting three wired LEDs in it at "good" locations so that in **most** positions and orientations they are visible and the object can be tracked. To extend this for tracking in *all* orientations, these LEDs can be planted all around the object so that always at least three are visible, and any three visible ones are used to track the object with six

degrees of freedom.

3.7 Modeling

The modeling system used is essentially the same as used with Shader Lamps [15] with a couple of modifications. The vertex list instead of being hardcoded in arrays is stored in a file. This file is the output of a program that uses a tracker to measure points on the object. A similar thing is done with the edge list, using a surface reconstructor [9] to create the mesh connectivity.

For each triangle we store a pointer to a texture map object instead of just an OpenGL texture ID. This is because we need to store each texture in an array also, and modify that array during the painting process, at which point the GL texture object is updated with the new values in the array. The texture coordinates are stored per vertex in the model file. Right now texture coordinates are obtained manually but for any very complex example they will have to be automatically generated. One way is to use a cylinder or sphere mapping and blow out all triangles from a central axis or point to the surface of a cylinder or sphere respectively, and then map coordinates on the cylinder or sphere onto a rectangular map. Special care has to be taken that no triangles share the same region of a texture map, or else painting in one triangle will cause inadvertent and absurd changes in the texture of another. This constraint can be incorporated into a texture coordinate optimization stage ([16], [7]) that follows the generation stage.

3.8 Rendering

The shaded model is currently rendered from two projectors on two channels of one graphics pipe of an SGI InfinityReality2 engine[10]. The projector setup is with both facing each other and slightly tilted downwards, to best cover the tracking region and also minimize the overlap surface area 1, so that wherever the object is moved within the region, in most orientations it gets full coverage on most of its surfaces with double intensity artifacts on as few surfaces as possible. The window setup is with two viewports, each one mapped by screen placement to feed a separate 1024x768 display channel switched to a projector. The whole window is refreshed at the same time by swapping buffers, so that the viewports are updated synchronously.

On one graphics pipe we are able to achieve nearly 60 fps of rendering speed, with one or more tracker updates being read per frame time. The latency of 4-5 frames between the tracker and the projector display is a significant factor leading to break in presence when the object or the brush is moved at moderate to high speeds - the shaded version lags behind the real one, as shown in 3. For static or slow moving objects, however, the shaded object is dead on in regions where the calibration sample points were taken - thus we stress the importance of calibrating with sample points throughout the projector overlap region and beyond. The latency would improve with the use of a more advanced tracking technology and also some prediction of future tracker readings using a Kalman filter-based method.

Since the user is not head-tracked in our prototype system, there is no view-dependent (specular) lighting as part of the shader lamp vocabulary demonstrated. The diffuse shading that is there looks correct from all user viewpoints and allows multiple simultaneous users, following the paradigm of spatially augmented reality first introduced by [14] and subsequently applied to a table-top environment by [12]. View dependent effects can easily be integrated, though, for a single tracked user as described in previous work[13].

3.9 Interactive 3D Painting

Once shader lamps had been extended to moving surfaces, the next step was to provide interaction that lets one modify the attributes of the shader lamps (and hence of the object) in real time. One could preprocess these shading attributes for a demonstration of shader lamps with any given model, but ultimately we would like to be able to create these demos on the fly by modifying the shading attributes interactively. This is an interaction process similar to that of painting the model surface, so we implemented the first application of painting for spatially augmented reality in the form of shader lamps. While it currently paints color directly and modifies lighting on the fly with a tracked light attached to the brush, it can be extended to 'paint' a new material and apply a displacement or other filter to the existing color or material shading.

The painting scheme used in this system is based on the one used by Gregory et al [4] in the inTouch system for haptic 3d painting. We maintain one or more texture maps for the model, with all the triangle vertices having texture coordinates into one of these maps. The following are the steps in the painting process:

3.9.1 Contact Determination by Proximity Detection

Right now we compute an axis-aligned bounding box for the model and then check the transformed position of the brush head against the bounding box, and if outside we stop. This saves us wasted checks for triangles to paint most of the time. When the check succeeds, we find the perpendicular distance between the brush head center and the plane of every triangle within the bounding box, and for those within an *admittance threshold radius* (a brush parameter), we make sure the perpendicular dropped from the brush center falls within the triangle or upto an *outside triangle margin* (normally a constant times the brush radius) outside its edges. This helps to make sure that the blob of paint is not clipped to the triangle within which it lies but stretches across to neighboring triangles. We collect a list of triangles that have met both criteria for sending to the next stage.

Note that this process could be improved vastly with an oriented bounding box for the model, or some other form of space subdivision, most effectively hierarchical subdivision as in an AABB or OBB-tree or an octree[3, 2, 8, 5]. The proximity test between a point and a polygonal surface is best handled as a sequential test within a region culled using bounding boxes. However for a more realistic brush geometry a promimity package such as PQP [6] may be used.

3.9.2 Brush Function

This function, used in blending in the brush color with the color of a point, provides the factor α to be used for the blend as a function of 3D point position given the position of the brush. We assume a spherical geometry for the brush head and a corresponding spherical distribution of the function around the brush center c, with the simple formula for BF at point x:

$$BF(x) = 1 - (r/R)^2$$

where r is the distance from c to x, and R is the constant "brush radius".

This is always between 0 and 1 for r¡R. Thus for this BF to work properly, the brush admittance threshold must be less than or equal to the brush radius. When it is equal, a very fine film of paint is deposited on triangles at the fringe of the threshold. This when coupled with a larger than normal brush radius leads to what we call the *airbrush mode*. When the threshold is markedly less than the brush radius, however, deposition will be immediate, strong and bold. We use this with a smaller brush radius and a threshold set to

the radius of a physical sphere attached to the brush head to simulate the haptic or *contact painting mode*.

3.9.3 Painting as Triangle Rasterization

One by one, the triangles chosen to paint are scan converted or rasterized using a special routine taken from [4]. The routine steps through 2D points on the triangle's texture map and corresponding 3D points interpolated from the 3D positions of its vertices.

3.9.4 Texture Map Modification

The brush function evaluated at the 3D point is used to calculate the new value in the texture map at its corresponding position. The formula used is a simple alpha blending:
$RGB(x_{2d}) = RGB(x_{2d}) * (1 - BF(x_{3d})) + RGB(brush) * BF(x_{3d})$

4 Initial User Reactions

Since the preliminary proof of concept implementation of our system has just been completed, systematic user study data is not available. The working system was demonstrated to leading graphics researchers and students, as well as to a media crew with no previous exposure to projector-based graphics research. It seems to hold universal appeal due to its creation of art on real objects in an augmented setting, while also demonstrating a step ahead of the power of shader lamps in a direction that may be critical to achieve the goal of true interactivity and scalability.

To the question of how this compared with other painting programs - 2D image based, 3D model based or fully immersive (virtual), the response was that this is certainly different from those applications and has a whole new range of applications encompassing those visualized for shader lamps. One user reinforced our feeling that the feedback of real paint-based and digital artists would be valuable. Accordingly in some future extensions we plan to study what these "power users" think about the system. Advanced technical users loved the artistic capabilities, but could see that tracking technology and its latency bounds were the significant bottleneck in performance as well as break in AR presence for the system.

5 Future Work

There are a number of unsolved problems with the current system and the whole paradigm of painting for SAR, the significant ones from which we list here:

- **Tracking:** The *latency* problem will not go away soon as the display, the tracker and the projector each add one or two frames of latency. Rather than go back to the case of not moving the object (static shader lamps) or moving it in a slow, restricted manner (as in bolting it to a turntable or a mechanical arm) to reduce latency, the preferred solution is to fine-tune the tracking code and add prediction [1] to it to offset most of the effects of latency.

- **Scalability:** In order to extend the *range* of operation of the system to cover the large room-sized environments we visualize it will be used in, we need more projectors and higher trackng range. For more projectors, the current solution scales readily (note though that the blending problem that we haven't solved is made more complex by the need to scale to n projectors). Projector placement to cover the space efficiently has been investigated [17]. Extending the tracking range is trickier, as is setting up the projectors and tracker to get the highest possible volume of intersection between the projection volume and the tracking volume.

- **Projection:** Dynamic blending in real-time is a problem not yet solved. The static case was solved in [15] by computing intensity weights. However, in our system with the surfaces moving around it is hard to solve for these weights in real time unless one can drastically minimize the area of overlap by partitioning the set of surfaces between the projectors in some way. Thus we ignore them and it shows up as double intensity in some regions and imperfect overlap between the images of the projectors at angles when both are contributing to a surface at an oblique angle.

- **Applications:** There is a need to develop and demonstrate applications for this technology, which clearly show its superiority to the existing way of painting on real objects as well as models inside the computer with no real object there. Creating an AR painting space is a technologal milestone, but commercial applications are needed to drive its development. Wide-area teleconferencing using the paint system on real objects is an obvious choice; so are various medical, cosmetic (painting on human skin), artistic and architectural applications.

6 Conclusions

We have presented a new system for 3D painting and projection on most real objects as they move. All that is required that there be two projectors facing each other, a tracker attached to the object (and in the paint brush), the projectors both be calibrated to the tracker's frame of reference, and that the object's geometry and texture coordinates be pre-acquired. The system is demonstrated with a cuboidal object, and this object is good enough to test the registration of vertices and edges in the real object and the projected texture. The same framework will work with an arbitrary number of overlapping projectors and an arbitrarily complex polygonal object (with the restriction of it being locally convex). The performance for high polygon count models can be accelerated using hierarchical collision detection techniques to pick the surfaces to paint. Some outstanding issues are dynamic blending for projector overlap, increasing tracker range, reducing latency to tolerable levels and developing compelling applications.

References

[1] R. Azuma and G. Bishop. Improving static and dynamic registration in an optical see-Through HMD. In *SIGGRAPH 94*, pages 197–204, July 1994.

[2] S. Gottschalk. *Collision Queries Using Oriented Bounding Boxes*. PhD thesis, Department of Computer Science, University of N. Carolina, Chapel Hill, 2000.

[3] S. Gottschalk, M. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In *SIGGRAPH '96 Conference Proceedings*, pages 171–180, 1996.

[4] Arthur Gregory, Stephen Ehmann, and Ming C. Lin. intouch : Interactive multiresolution modeling and 3d painting with a haptic interface. In *ACM Symposium on Interactive 3D Graphics (I3D'99) Conference Proceedings*, March 1999.

[5] Arthur Gregory, Ming C. Lin, Stefan Gottschalk, and Russell Taylor. A framework for fast and accurate collision detection for haptic interaction. In *IEEE Virtual Reality 1998 Conference Proceedings*, 1998.

[6] Eric Larsen, Stefan Gottschalk, Ming C. Lin, and Dinesh Manocha. Fast Proximity Queries with Swept Sphere Volumes. Technical Report TR-99-018, Department of Computer Science, University of N. Carolina, Chapel Hill, 1998.

[7] S. R. Marschner. *Inverse Rendering for Computer Graphics*. PhD thesis, Department of Computer Science, Cornell University, 1998.

[8] D. Meagher. Geometric modeling using octree encoding. In *Computer Graphics and Image Processing*, volume 19, pages 129–147, June 1982.

[9] Gopi Meenakshisundaram and Shankar Krishnan. Surface reconstruction using lower-dimensional delaunay triangulation. In *Eurographics 2000 Conference Proceedings*, June 2000.

[10] J. S. Montrym, D. R. Baum, D. L. Dignam, and C. J. Migdal. Infinitereality: A real-time graphics system. In *SIGGRAPH 97*, pages 293–302, August 1997.

[11] R. Raskar. *Projector Based 3D Graphics*. PhD thesis, Department of Computer Science, University of N. Carolina, Chapel Hill, 2000.

[12] R. Raskar, G. Welch, and W.-C. Chen. Table-top spatially augmented reality : Bringing physical models to life with projected imagery. In *Second International Workshop on Augmented Reality (IWAR'99)*, pages 11–20, November 1999.

[13] R. Raskar, G. Welch, M. Cutts, A. Lake, L. Stesin, and H. Fuchs. The office of the future: A unified approach to image-based modeling and spatially immersive displays. In *SIGGRAPH '98 Conference Proceedings*, pages 179–188, 1998.

[14] R. Raskar, G. Welch, and H. Fuchs. Spatially augmented reality. In *First IEEE Workshop on Augmented Reality (IWAR'98)*, pages 11–20, November 1998.

[15] Ramesh Raskar, Kok-Lim Low, Deepak Bandyopadhyay, and Greg Welch. Shader lamps : Animating real objects with image-based illumination. In *Eurographics Rendering Workshop 2001*, August 2001.

[16] P.-P. J. Sloan, Weinstein D. M., and J. D. Brederson. Importance driven texture coordinate optimization. In *Eurographics 1998 Conference Proceedings*, volume 17, June 1998.

[17] Wolfgang Strzlinger. Imaging all visible surfaces. In *Graphics Interface'99 Conference Proceedings*, pages 115–122, June 1999.

[18] G. Welch and G. Bishop. SCAAT: Incremental tracking with incomplete information. In *SIGGRAPH 97*, pages 333–344, August 1997.

[19] G. Welch, G. Bishop, L. Vicci, S. Brumback, and K. Keller. The hiball tracker: High-performance wide-area tracking for virtual and augmented environments. In *Symposium on VRST*, December 1999.

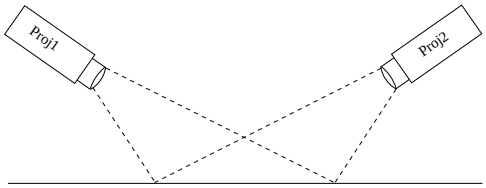


Figure 1: Schematic diagram of the setup. Two projectors face each other and projecting downwards, cover the volume where tracking is active.

Figure 2: Users demonstrating their creations with the 3D paint system

Figure 3: Demonstration of the latency in the tracking pipeline. Notice the blurring and trails artifacts.

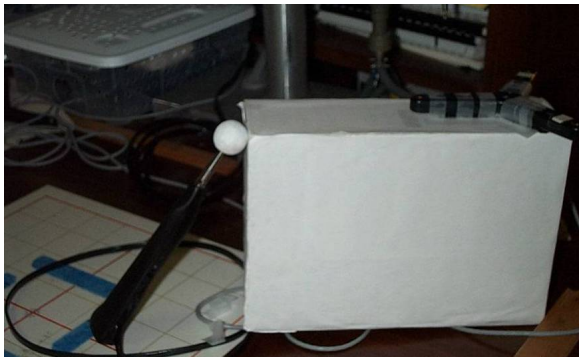


Figure 4: Completed system (projectors off) with a tracker placed on the object

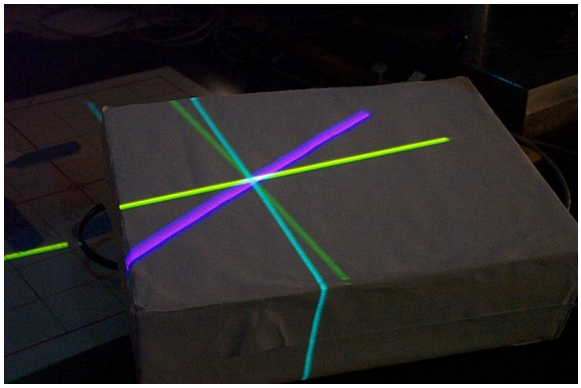


Figure 5: Projector calibration

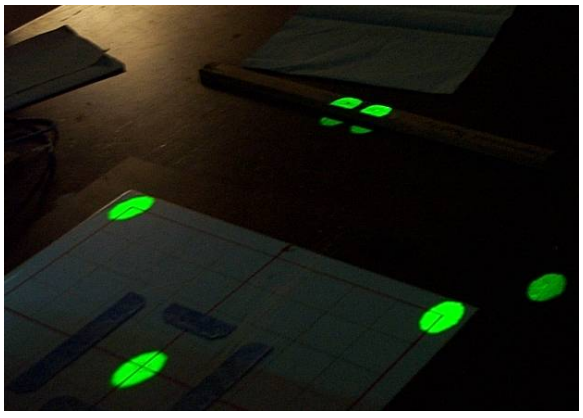


Figure 6: Verification of calibration - drawing markers at known positions in the real world using calibration matrices. When the calibration is a little off, this method can be used to detect this and roughly recalibrate by moving the projector until the images all snap to the correct locations.