Interactive Multi-Projection Display System

by

Eoghan Cunneen, BA (Mod) Computer Science

Thesis

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Computer Science

University of Dublin, Trinity College

September 2009

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Eoghan Cunneen

September 8, 2009

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Eoghan Cunneen

September 8, 2009

Acknowledgments

As a part of this thesis, I would like to thank Dr Gerard Lacey for his supervision during the course of this project. I would like to thank members of the *GV2* group who generously offered help and suggestions, and to my own Interactive Entertainment Technology class for making the past academic year as enjoyable as it was. Finally, thanks to my family, Emily and my closest friends for their support and encouragement over the last 12 months.

EOGHAN CUNNEEN

University of Dublin, Trinity College
September 2009

Interactive Multi-Projection Display System

Eoghan Cunneen
University of Dublin, Trinity College, 2009

Supervisor: Dr Gerard Lacey

The subject of multi-projector and projector-camera pairs is currently receiving a lot of attention in both industrial and academic research fields. Multi-projector displays offer the new opportunity of creating large scale displays, paving the way for fully immersive environments in ways that a single projector could ever hope to achieve. The rise of interest in this research field is driven by ever-growing computer processing power in addition to the declining cost of high performance, highly portable commodity hardwares.

The use of multi-projection technologies can revolutionise a number of distinct areas. Virtual environments, public presentations, scientific visualisation, interactive entertainment (gaming), performance art and even architectural displays are but a few areas which can evolve with the use of emerging multi-projection display systems.

However, to achieve both geometrically and photometrically seamless imagery, the accurate co-calibration of each projector in the multi-projection system is imperative. This dissertation investigates the problem of single and multi-projector calibration with the use of a camera device. Utilising camera-projector homography, the calibration of projector and planar display surface can be solved. To resolve the issue of free-form surfaces, Gray-coded binary structured light is used to analyse the surface geometry before remapping the display image accordingly. The whole process adopts accelerated graphics hardware for speed and efficiency.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

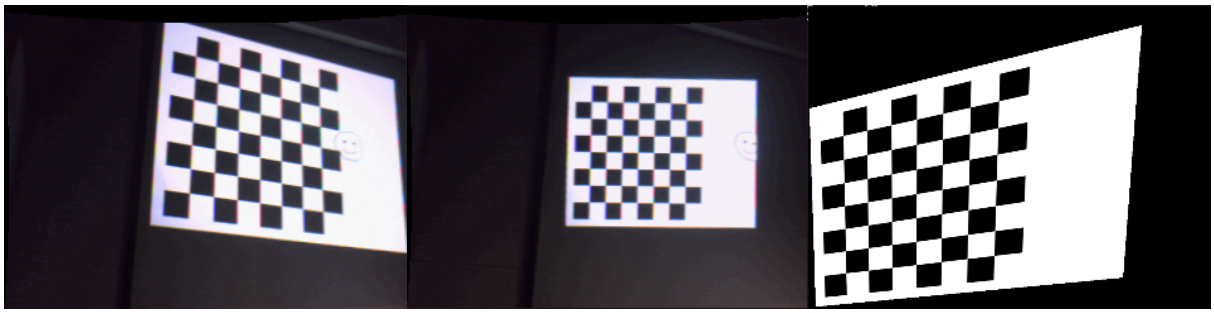
"Web 2.0 is Yesterday. Prepare for World 3.0" - Tom Melamad, HP Labs

We are currently experiencing a paradigm shift in the way we communicate and interact with technology and the immediate world around us. Web 2.0 brought world wide collaboration, information sharing, social networking and distributed knowledge. Now we see web technology creeping off our desktops and laptops and into, not just our mobile or smart phones, but into the physical landscape around us.

The use of projection technology in the public domain can and *is* creating a new and exciting form of media. *Pervasive media*, where human-computer interaction is not confined to the desktop, does not have to be experienced via your web browser, but can interact with you as you walk through the physical environment surrounding you. This technology creates an augmented space, a space in which you retrieve information, interact, play, and can be entertained by.

The goal of this dissertation is to develop a flexible calibration framework for a multi-projection system and input interface for large-scale pervasive media systems. The system will be capable of solving projection problems such as *keystoning* artifacts, produced by a projector when it is not aligned with the normal of the projection surface, and *geometric* artifacts caused when projecting upon non-planar, or *free form* projection surfaces. The application draws from techniques that are considered state-of-the-art, and combines them into one, manageable framework.

The scope of applications for this research includes, but is not limited to, information communication, commercial advertising, architectural displays, augmented reality based interactive entertainment and performance art. The software application has been built upon Openframeworks, a C++ based library of various graphics, vision and sound libraries. Through this framework, a multi-projector and camera network can be easily calibrated and deployed. The desired outcome will be a robust and easy-to-use calibration workflow and intuitive user-interface system with which artists or operators can project their desired graphical content onto any arbitrary projection surface.



(a) Uncalibrated, keystoned image

(b) Rectified image

(c) Pre-warped projected image that is projected in (b)

Figure 1.1

1.1 Motivation: The Projector-Camera (PROCAM) Model

The goal of the projector-camera model is to fully automate the projector calibration process by using the camera device to correct keystone artifacts caused by non-orthogonal projections, and to remap the projection image to display a seamless and coherent image on a free-form, non-planar projection surface. The system is calibrated once, offline, before the parameters are saved and the application is started.

The projector can be modeled as the inverse of the pinhole camera model and together with the camera device creates an uncalibrated stereo pair. This allows for an opportunity to calculate a relationship between both devices, regardless of any intermediary surfaces.

The camera device must be calibrated first to rectify potential lens distortion that could affect the projector-camera calibration phase. If the projection surface is planar, a homography between the camera device and the projector is calculated and is inverted to pre-warp the projection image to appear square to the camera and audience. However, if the projection surface is a non-trivial or non-planar surface, Gray-coded structured light is implemented to encode the projection surface's feature points in the camera image plane. The displacement of corresponding points can be calculated, inverted and applied to pre-warp the projection image for coherent viewing. In each case, graphics accelerated hardware can be exploited in the pre-warping stage of the calibration to increase the speed of the pre-warping process. Projector-camera models can be extended to multiple projectors, where each projector is calibrated individually from the perspective of the same camera, and the larger image is stitched together from the individual projections. These topics will be discussed and explained in further detail throughout this dissertation.

The reason for selecting a PROCAM model over an alternative, manual system will now be presented. While there are substantial strengths to a projector-camera system, it is not without its drawbacks.

1.1.1 PROCAM Advantages

- Aribitrary projector location, liberal setup
- Cut down presentation/display setup time drastically

- Full automation and no need for human involvement
- Accurate calibration results
- Larger displays with more projection devices

1.1.2 PROCAM Problems

- Photometric (colour) inconsistencies across surfaces and textures
- Utilisation of smaller percentage of projection frustum
- Projection technology is limited if the ambient light level in the environment is too high

1.2 Dissertation Layout

The rest of this dissertation will be laid out as follows: Chapter 2 will present a literature review of the current state-of-the-art in the field of projector-camera models, introducing papers and each individual aspect of the calibration framework that is important to this dissertation. Here, camera calibration, projector calibration, geometric calibration and multi-projector calibration techniques are presented and discussed. Research and development areas are also presented here, offering an overview of the real-world applications for these systems. In Chapter 3, the design of the system, which is based on the extensive research presented in Chapter 2, will be presented and defended. This will provide system architectures and details of how the system will operate. Chapters 4. 5 and 6 will present each step of the implementation of this system. First the camera calibration technique will be presented and its use in this system shown. Then the automatic projector calibration theory will be discussed and implementation outlined. Finally the geometric calibration step will be explained and its implementation discussed. At this stage, the testing of various different metrics regarding this system will be carried out and analysed in Chapter 7. Finally, Chapter 8 will conclude this dissertation and offer possible future work for the topic.

1.3 Glossary of Terms

Homography	3x3 projective transformation relating two images
Keystoning	The distortion caused to a projected image when the projector is at an angle to the screen
Structured Light	The projection of a known light formation into a scene to analyse geometric makeup of a scene
Gray-code	A binary numeral system where successive numbers differ by 1-bit
API	Application Programming Interface
CPU	Central Processing Unit
GPU	Graphics Processing Unit
FPS	Frames Per Second
GLSL	(OpenGL) Graphics Library Shading Language

Chapter 2

Related Work and Literature Review

2.1 Camera Calibration

In any situation that requires accurate camera information for accurate scene analysis, the first step is ensuring that the camera device itself is accurately calibrated. In situations such as stereo vision, where point correspondences between two cameras must be made, it is mandatory to know the intrinsic and extrinsic parameters of the camera devices. Zhang [49] provides a straightforward technique that will correct the radial and tangental distortion of a camera induced by the camera's lens. Kimura *et al* [20], Fiala [14] and Zhou *et al* [50] all rely on the calibration of their camera systems for the calibration of their projection systems.

Zhang's method simply relies on the camera to identify a known planar pattern from *at least* two orientations. In this case either the camera, or the plane itself, can be moved in any arbitrary direction. The user will grab images of the patterned planar surface via the camera device and detect the feature points in the image. A closed-form solution is used to calculate the camera's *intrinsic parameters* (focal lengths, principal points) and *extrinsic parameters* (rotation and translation of camera in real world coordinates with respect to the calibration target). Finally the radial and tangental distortions are estimated. An in-depth analysis of this method will be discussed in Chapter 4.

Another method, described in Raskar *et al* [31] uses a 3D calibration target, rather than a 2D planar surface. Raskar *et al* place the pattern, which has known feature locations, within the frusta of the stereo-camera system. By calculating the correspondences of the 3D target in 2D camera image space, the projection matrix of each camera in the stereo pair can be resolved. As in Zhang's method, the numerous unknown parameters of the projection matrix are solved using the least-squares method.

2.2 Projector Calibration

Sukthankar *et al* [39] [40] provide means of accurately calibrating a projector system with an arbitrarily placed camera and projector. This method depends on the use of a quadrilateral projection screen surface

to calculate perspective warp necessary for the audience to see a coherent, rectilinear image on the projection screen. Sukthankar *et al* use a *region growing* algorithm to ascertain the full projection surface before the homography between camera and screen is then calculated using the four corner points of this projection screen. The projector-camera homography is calculated by the projection of known points onto the projection surface by mean of calibration slides. These known projection points undergo a projective transformation related to the orientation of the projector to the screen, and the screen to the camera. This is then solved for by calculating each component of the 3x3 homography matrix, which is presented in Chapter 5. A heuristic optimisation is finally used to determine the largest rectangle, with matched aspect ratio, that can fit within the bounds of the projection frustum as it hits the projection surface. This ensures that the audience will see the largest to-scale image that the projection setup can permit.

Sukthankar *et al* claim that their system is not *"naive"* as it calibrates the projection image to appear coherent to the audience looking at the projection screen, rather than appearing coherent in the camera image plane. This dissertation will argue that, without depending on a quadrilateral projection surface, with just some extra effort to place the camera horizontally and position the camera at the *sweet spot* of the audience, the same results can be achieved.

Fiala [14] and Okatani and Deguchi [28] also outline methods for carrying out the automatic calibration of a projector using a camera. Fiala, first, uses computer vision techniques to calibrate a series of projectors. *ARTag*, an augmented reality API, is used to project unique patterns onto the projection screen. The ARTag patterns (fiducial markers or tags) are known and recognised and therefore can be detected by the camera system. The tags are light invariant but also immune to affine transforms and perspective warps. The tags are also capable of scaling, managing to be recognised from between the size of 15 pixels to the camera viewport dimension. Knowing the homography between projection surface and camera device and the projector-camera homography, the projector to projection surface homography can be calculated to pre-warp the projected image.

Okatani and Deguchi present a method for computing the projection surface-camera homography without the prior knowledge of the screen. Its goal is to project onto any arbitrary surface, rather than a designated screen. In this setup, the projector's extrinsic parameters are unknown while the projector's intrinsic parameters are known, while the camera and projector are based on pin-hole models. The method outlines their efforts to uncover both the screen-projector homography and the camera-screen homography. What is different about this implementation is that the camera-screen homography is estimated by projecting images onto the display screen and capturing them by camera, rather than feature points physically located on the screen and then captured by camera as in [14], [39] and [40].

To establish the relationship between the projector and the screen, the authors identify a 3D projector coordinate frame, $O - XYZ$, where the focal point of the (laser) projection beam corresponds with with the origin of the projector's image. A screen coordinate frame, $o - xyz$, is also identified where the xy plane coincides with the display screen plane. $M \equiv [X, Y, Z]^T$ represents the direction of each beam in the projector coordinate frame, while $m \equiv [u, v, 1]^T$ are homogenous coordinates in the screen coordinate frame, where u, v are the x and y screen coordinates. m and M are related by: $m \propto KRM$, where K is a scale and translation and R is a rotational component of the coordinate transform from the projector frame to the screen frame and $\propto$ signifies a proportional relationship.

To capture the homography between camera and surface, it is common to locate known feature points (at least 4) in the camera image plane to calculate the 3x3 transformation. However Okatani and Deguchi offer a solution that calculates a camera-screen homography where no feature points are detectable. The homography between the camera and screen is instead computed using a noniterative method whereby camera-screen homography, H_{sc}, is calculated from a series of camera-projector homographies, H_{pc} ($H_{pc1} \dots H_{pcN}$). Like in some camera calibration techniques, this problem is solved by factorising the matrices into a specific form. The goal here is to find a particular H_{sc} that permits the factorisation $H_{pc} \propto H_{sc}KR$ for a given H_{pc}.

Lee *et al* [22] use a different technique for projector calibration. Instead of relying on a camera for calibration and rectification, the projection surface is instead augmented by the addition of light sensors. These sensors receive and transmit the data to decode a structured light sequence, described later, and provide the relevant transformation that the projection image must undergo to project a coherent image on the projection surface. This is a novel approach to solving the projector calibration problem, and the results are highly accurate. While the initial prototyped system connected the fibre-optic light sensors via wires, Lee *et al* have also developed a wireless alternative. The trade-off between high accuracy and ease of setup must be assessed in this case.

Kimura *et al* [20], rely on the same projective geometry between a calibrated camera and the projector. Instead of using single homographies, Kimura *et al* consider the projector as an inverse of the pin-hole camera model. In this regard, they consider the camera-projector pair as a *stereo vision* pair and attempt to calculate the *fundamental matrix*. The fundamental matrix is an $n \times n$ matrix that relates the image planes between two cameras and can be easily acquired by registering several views of a specific observed plane and an epipole. Using the known camera projection, the projector's projection can then be computed by optimising the extrinsic parameters of the camera.

2.3 Geometric Calibration using Structured Light

While the calibration of a projector system is possible for trivial, planar surfaces using techniques described above, they are not suitable for calculating the deformation that occurs to a projected image when projected upon a non-trivial, free-form surface. The described techniques above depend on the projection surface to lie on a two-dimensional plane, however when a third dimension comes into account, an alternative projection calibration technique must be performed.

Geometric calibration using structured light is a well researched and documented topic, with the development of specifically coded patterns for specific purposes. Gray-coded binary, [22], [18], [1], can be used for robust, noise resistant surface acquisition. The process works by shining alternating black and white stripes of light onto the projection surface. For n pattern sequences (n-bit), the projection screen is divided into $2n$ sections, which encodes each pixel covered by the projection frusta with a 1 or 0 value. These values are recorded for each n pattern, creating an n-bit codeword, each of which corresponds to a pixel in the image to be projected.

Gray-code is preferred to binary-code, used in [44], [43], [42], for the following reason: When the stripe

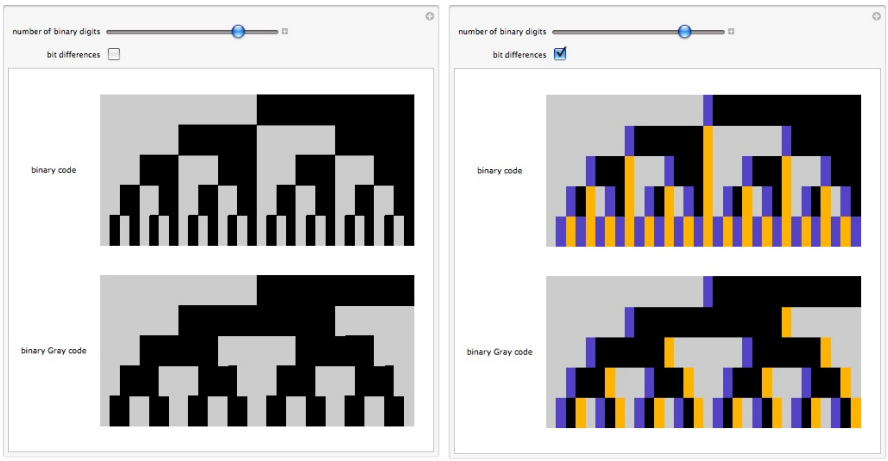


Figure 2.1: Binary versus Gray-code: Image(left) binary vs Gray-code pattern, while image(right) displays the boundary changes [47]

patterns of the Gray-code are projected, the boundaries between the black and white stripes never occur in the same space. This means that ambiguities as to whether a pixel is in one location or another, do not arise. Figure 2.1 depicts a *Mathematica 6* notebook output depicting the bit layout and the boundary changes.

Gühring [18] presents a combination of both Gray-coded binary and a phase shift. In contrast, Lee *et al* [22] use only the Gray-code on two axes, while the light sensors provide a more accurate capture than a camera would hope. Gühring opts for Gray-code for its robustness, however to overcome the resolution inadequacy of casting just vertical stripes, the author adds a phase shift step. The phase shift step is an on-off pattern cast as a sine wave that is shifted in steps of $\frac{\pi}{2}$. Lee alternatively opts to cast the light vertically as well as horizontally. This is assuming that the orientations of the projector and camera are unknown. If the information regarding the relative placement of either device is known, then only one orientation of structured light would be required.

In an attempt to create a full, low cost 3D scanner based on structured light, Rocchini *et al* [33] adapt the binary Gray-code projection technique and utilise the projector to its potential. Instead of projecting two-tone black and white images, this technique uses colour coded light. At every step, each region is divided by a one-pixel wide green stripe. The regions to the left and right of the green stripe are coloured blue and red respectively. Instead of analysing the discontinuity between regions, as in traditional structured light based 3D extraction, this method searches for the centre of the green light stripe. First, for every captured image the pixels that represent areas that are occluded, thus remaining black, are removed. The green stripes are then detected and indexed per-pixel and the medial axis of each stripe is calculated to sub-pixel accuracy. These computed medial axes are added to those already found in previous images and then, for each indexed stripe found in each image, the 3D point is calculated by triangulation and stored before the 3D point cloud gets re-meshed.

This alternative colour-coded structured light approach can be more accurate as the brightness caused

Figure 2.2: Structured light step proposed by Rocchini *et al* [33]

by the light and dark contrast in black and white structured light can cause a widening effect on the brighter stripe at the expense of the darker one. If the brightness of the red and blue regions are kept consistent with that of the green stripe, this becomes less of an issue.

Zhang *et al* [48] also use color coded structured light in an attempt to extract depth information from a scene. Rather than aiming to carry out this task accurately, the authors have developed a way of extracting depth very quickly. While [44], [22] and [1] all use a number of structured light images to encode each pixel in camera space, Zhang *et al* use just one colour coded image to acquire depth. In this method, equally sized coloured stripes in a single image are projected into the scene and captured by the camera device. Each stripe is enumerated in a string of colours, $P = (p_0, p_1...p_n)$, where the information required for triangulation is located at the discontinuities between the colour stripes, $Q = (q_0, q_1...q_{n-1})$, where $q_j^c = (q_j^r, q_j^g, q_j^b)$. At each transition, the separate r, g, b colour channels take a -1, 0 or 1 value depending on whether its value is falling, remaining constant or rising.

The pattern is observed by the camera device as a series of color edges, $E = (e_0, e_1...e_{M-1})$, where each edge is described as $e = (e^r, e^g, e^b)$, with the edge intensity levels stored in the r, g, b channels. Correspondence between Q and E must now be computed. In attempt to overcome inevitable problems caused by occlusions from irregularities in the surface, and colour mislabeling caused by specular or reflective surfaces, underlying textures or projector colour misrepresentation, the calibration technique introduces what they call *multiple hypotheses*, where every label assigned to a stripe is paired with a probability matching of that label. A global match optimisation technique then deals with these possibilities, labeling the stripe with its most likely value.

Ashdown *et al* [2] and Bimber and Rakesh [5], offer horizontal and vertical line-based methods of planar surface identification (Figure: 2.3). The structured light step taken in Ashdown *et al* is composed of vertical and horizontal light stripes being projected onto the arbitrary surface. As the lines cross separate surfaces, kinks appear in the line in camera space. These kinks are located and can be connected to determine the precise boundary. The projector-camera homography is then estimated using a Direct Linear Transform (DLT) method and applying it to a set of point correspondences. Each point x corresponds to a point x' and are directly related by a homography H. Each point provides two constraint equations (an x value and a y value). So these n points (where $n \geq 4$) can be combined into a $2n \times 9$ matrix and solved using Singlular Value Decomposition (SVD).

At this stage, the camera-screen homography must be calculated for each i screens. To do this, the camera views a number of right angled rectilinear (envelope) shaped objects placed on the surfaces.

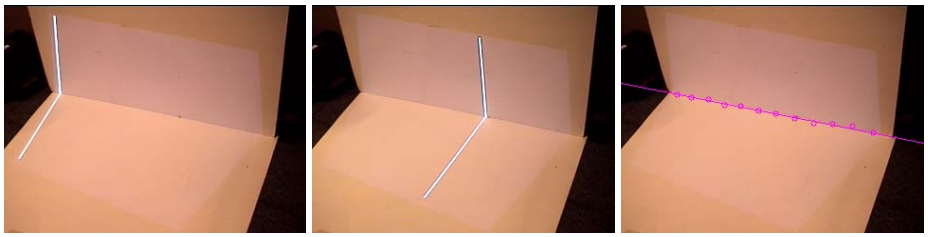


Figure 2.3: Structured light step proposed by Ashdown *et al* [2]

Metric rectification of the image of a planar surface is carried out by finding 5 or more line pairs, the postcards, that are orthogonal to the surface. These constraints, again, can be formed into a null-space problem and solved for by SVD.

Finally, an iterative refinement algorithm (homography refinement algorithm) is used to calculate the most accurate mapping possible. The aim of this step is to make sure that each common point on the boundary between surface i and surface j can map to its position in the projector image, regardless of whether the homography between projector and surface i, or surface j, is applied. As two separate surfaces are being considered, each shared point on the boundary creates two separate points in the projection image that should in fact be the same. This is rectified by taking the midpoint between these distinct points. The homography between surface i and surface j are recomputed with the new point correspondences. This process is repeated until the error metric falls beneath a certain threshold.

The projection of light need not be contained within the range of visible light. Frueh and Zakhor [15] use the infra-red spectrum in their solution to create a system that will capture the $2\frac{1}{2}$D geometry of a scene. The IR light, captured by a camcorder on *nightmode*, is invisible to the human eye and is used to extract the depth data, while the texture of the scene is preserved and captured using an ordinary firewire camera.

The depth is acquired solely in the IR domain, where equally spaced vertical IR stripes are projected at the scene, while a single horizontal stripe of IR light scans the scene from top to bottom. The horizontal IR scan line is used to distinguish the vertical lines from each other, as the vertical lines can become ambiguous due to occlusion or deformation on a steep surface. Differentiating the vertical lines with identification colours within the IR spectrum is impossible due to technical constraints. The depth then, is obtained at the horizontal line at different locations during separate IR frames (IR frames captured at 30Hz), resulting in a different set of vertical stripes being located at each frame. The vertical displacement between the IR camera, which captures the scene and is located directly next to the firewire camera, and the horizontal IR emitter is known, so the depth of the scene between the camera and the object in question can be resolved by triangulation.

Furukawa and Kawasaki [16] also present a scheme for extracting depth using structured light with an uncalibrated projector-camera pair. The authors opted for the structured light with a projector-camera pair rather than a camera pair, as the use of structured light will present far more correspondence points than a camera pair would hope to. This means that the interpolation methods to populate a dense point cloud from fewer feature points used in a camera pair system will not have to be applied to this system,

potentially making it more reliable. Again, Gray-code is opted for to project both vertical and horizontal light codes into the scene, permitting an arbitrary projector-camera setup. A laser pointer is used for depth analysis and the consequent scaling issues. If scale of the resulting image is not an issue, the laser is simply left out. To obtain the point correspondences, the vertical and horizontal stripes provide the 1D correspondence points obtained by the camera device which when paired offer the 2D point. The 3D reconstruction is carried out by solving the epipolar constraints in Euclidian space where a non-linear optimisation is applied to refine the accumulated results.

The coordinate system for this implementation is expressed as the projector coordinate system. The projector is considered a camera in this setup where the origin of the coordinate system is the optical center, or principle point, of the projector image and points in the $-Z$-axis direction. As in [28], the x, y axes of the coordinate system lie on the same plane as that of the image coordinate system of the screen and the focal length is expressed as the direction vector of the ith correspondence point in projector coordinates. The correspondence between the projector image plane and the camera image plane are expressed as a rotation matrix $\mathbf{R}_p$ and a translation vector $\mathbf{t}_p$, where the rotation matrix is made up of Euler angles α_p, β_p and γ_p and expressed as $\mathbf{R}_p(\alpha_p, \beta_p, \gamma_p)$. The epipolar geometry, where the lines of sights of projector and camera intersect in 3D space, is then solved for.

Batlle *et al* [3] and Pagès *et al* [29] provide a thorough overview of many of the structured light techniques discussed.

2.4 Calibration of Multi-projection Systems

Raskar *et al* [31] provides a framework to use an arbitrarily positioned multi-projector system to project onto an irregular projection surface. Their system tracks a moving viewer through the environment by use of an infrared tracking device, where Lagrangian multipliers are used to compute the transformation between the coordinate system of the tracking device and that of the world coordinate system defined by the 3D calibration pattern. This framework uses a stereo-camera model, rather than a single camera, and uses binary structured light to identify point correspondences in the 3D projection surface. 2D Delaunay triangulation is then used to reduce the points to a mesh structure. The projector's projection matrix is easily calculated by corellating each projector pixel to its corresponding 3D surface point. A 2-Pass rendering algorithm is presented that simply stores the desired image as a 2D texture, before using this to texture the 3D mesh of the projection surface. For the multi-projection stage of calibration the process is repeated, registering the projection surface and each of the projector calibrations to a common world coordinate space. A 3x3 rotation and 3x1 translation vector, together with two corresponding points are resolved using least-squares. The correspondence between the points, like before, is solved by binary-coded structured light.

Taking multi-projection calibration to whole new level is Chen *et al* [10], who present a method for calibrating any arbitrary amount of projectors with a single pan-tilt-zoom camera. Here, *24* 1024x768 XGA projectors are used in a 6x4 array, powered by a group of networked PCs. The projectors cover an 18' x 8' projection surface creating a surface display of approximately 18 megapixels. Like Sukthankar *et al*

[39], this method depends on homographies between the camera and the surface, and the camera and projectors. First *camera-camera* homographies are recorded to develop a relationship between numerous camera poses and the projection surface, essentially grabbing reference frames for the entire surface and building a mosaic of the projection surface. These mosaics must overlap - this is because the homography between two camera views cannot be calculated if the two views do not share common feature correspondences. As a result, a homography tree which spans the entire set of views is created, whereby the mapping of any view can be determined by compounding the homographies along the path to the reference point at the root of the tree.

The projector-camera homography is calculated as per other similar methods. Each projector displays a specific calibration target, whose well defined features are know in projector space. By locating the corresponding features in camera space, the homography can be simply found. However, due to the size of the display in question, inaccuracies can accumulate when the resolution of the display is far greater than that of the camera. As a result, the authors propose a more accurate estimation of the projector-camera homography. Instead of a simple checkerboard being projected, pairs of horizontal and vertical lines are projected onto the surface by each projector. The intersections of these lines can be found, not by edge detection, but by fitting a quadratic function to pixel intensities in the stripes. A strong peak of the function at a particular point indicates a feature and provides sub-pixel accuracy of the line's local position with floating point precision. The new Camera Homography Graph (CHG) takes on a latticed behaviour when projected upon a rectangular wall, where each vertex represents a camera view, and each edge represents a directly computable homography. When multiple paths exist between two homographies, a global registration method is used to resolve the issue.

Tardif *et al* [42] provide more research into multi-projector calibration. This method employs the use of a single camera, used to simulate the audience's point of view, and two projector systems to rectify geometric distortion caused by a non-planar projection surface. This method uses structured light to analyse the depth of the scene, opting for binary-coded light, the simplest form, as they are not concerned with the speed and optimisation of the this step. The calibration step is performed *offline* before the display image is warped in real-time with the use of graphics accelerated hardware.

The binary structured light is projected together with that bit's inverse. If in the first instance *white* (1) was projected, the inverse will be projected *black* (0). At this stage, the inverse image values are subtracted from the converse image values, creating a resulting image whose values range from -255 to +255. The points in the camera image space that capture the projection undergo a large contrast (difference typically greater than 150), while those outside of the projection will not have hugely contrasting values at all (difference typically less than 50).

Once the images are thresholded, and the bits identified, the horizontal bit values are concatenated with the vertical bits, creating $NumOfBits \times 2$ long word. As the projected binary code nears the least significant bit (LSB), the resolution of the camera is often not capable of distinguishing between the black or white code. This results in rejected pixel values for a number of the LSBs. These bits are set to 0, creating a down-sampled mapping.

For each coordinate in the camera image space that is not rejected from the previous thresholding step, a down-sampled mapping is created to inversely map the display image to the camera image plane.

An estimate of the centre of the display image in the camera image space is achieved by averaging the non-rejected points in the camera image space, this provides a rough mapping between the display image and camera image plane. This process is repeated for each coordinate in the camera image space.

The image is then reconstructed with the use of OpenGL textures. The down-sampling is used as a square mesh, where the original 1024 x 768 image is rebuilt as 128 x 96 which represents 12065 squares. This essentially texture maps the camera image space in real-time, utilising interpolation schemes in the process. The down-sampled image must undergo a final transform before the calibration step is deemed complete. It is necessary to find the largest *to-scale* rectangle that can reside within the projected area within the camera image space. This creates a similarity transformation between the newly created rectangle in camera image space, and that of the display image. This process is extended to multiprojectors by carrying out the described method for each projector for any arbitrary amount of projectors. For a correct image, it is important that the projectors must overlap before intensity blending is carried out.

Zhou *et al* [50] use projector-camera pairs for their calibration method. In this technique, there are i projector-camera pairs, $\{Pi, Ci\}$, each of which have their intrinsic and extrinsic parameters calculated. The cameras are calibrated by the same technique proposed by Zhang [49]. Once the cameras are calibrated, the uncalibrated projectors $\{P_1 \dots P_i\}$ each project a checker-board pattern respectively. Then, using the calibrated cameras and the same techniques as one would calibrating a camera, each projector's intrinsic and extrinsic parameters are then accounted for.

Zhou *et al* use an interesting method to project on to the projection surface. Harris corner detection is used to extract the local features (corners) from an image projected from another projector in the set. Normalised Cross Correlation (NCC) is then used to match points of the image already on the projection surface, and the image to be projected by the current projector. What differs from this technique in contrast to other similar techniques is that; while the initial projector-camera and geometric calibration steps are carried out offline, there is an *online*, automatic self-calibration step that runs for the duration of the application. The automatic self calibration step assesses any change that may be caused either by projector movement, or projection surface movement. Every frame, the camera image is compared to the camera image from the last frame. If there is any discrepancy, then there may have been a change in the system configuration and the calibration stage is carried out again for that camera-projector pair. Harris feature detection is used to locate the 2D image features in the camera plane. These matched 2D correspondences are then triangulated to obtain points in 3 dimensional space before RANSAC is used again to find the optimal plane fitting.

This method is also capable of projecting upon free-form surfaces using structured light to extract the 3D geometrical make up of the scene. 2D correspondences between Pi and Ci are found for each camera-projector pair. As the frusta of the cameras may overlap, a point of the projection surface will be present in different locations according to different cameras. To solve this, a rigid rotation and transformation is carried out align the point between the two cameras.

The performance of this system is limited to the camera-projector synchronisation (feature extraction per frame). At this rate, the system runs at 3.5Hz, however with a better synchronisation technique, the application can meet video frame rate standards ($\sim$24fps).

Once each of the projectors in the multi-projection set are calibrated, the overlap of projection beams

cause non-uniform light saturation on the projection projection surface. To compensate for this, photometric calibration must also be carried out to ensure a uniform and constant projection surface.

2.5 Other Applications

This section will give a brief overview of some other novel implementations of projection technology that are less related to this dissertation.

Bimber *et al* [4] present an interesting application for projection technology in the realm of the *augmented studio*. This setup provides the opportunity to create real-time compositing effects in a real world situation, using blue or green screen technology to place the scene actors into a computer generated environment. Using multi-projector arrays and digital illumination, the computer generated scene's colour scheme can be replicated in the LED lights, allowing the system to light the scene so that it is consistent with the computer generated imagery projected on the screen.

A novel table-top projection display system is presented by Cotting and Gross [12]. In this implementation, environment-aware display bubbles can be projected onto a user's work top. Conscious of its immediate surroundings, the projection utilises 2D rigid body physics and collision handling to only project on the projection surface, and not objects placed on top. These objects are found by using structured light to detect depth discontinuities before a Gabor filter is applied to the captured image. Thresholding is applied to the image after the filter is applied, resulting in a binary image where white space designates the area where the bubbles can be projected. The system is interactive, running at real-time rates.

A novel interaction wall is created by Summet *et al* [41] where redundant projection light from a number of projectors is used used to simulate a *back-lit* projection setup. Using two or more projectors, the potential for total occlusion caused by an obstacle is alleviated as the second projector compensates for the area that the initial projector cannot reach. This instead creates two partial or half shadows, rather than a single, full one. The use of a camera device allows interactivity between the projection surface and users of the system. The novel interaction setup uses similar automatic calibration and keystoning processes as were discussed before.

Song and Cham [36] use similar fiducial markers as those used in Fiala *et al* [14] to create a tangible game interface. In this system, four fiducial markers placed on a surface are used and tracked by a camera. The homography between the board can be found by locating each marker (as before, there must be at least four), where a tracking system can calculate the tilt of the surface. The game play is then based on the orientation of the board. In essence, this setup acts very much like an accelerometer that you would find in modern game controllers.

In any of the systems discussed thus far, the permanent location of the projector in the scene is imperative, while in some implementations such as [31] the viewer, or virtual camera, moves about the scene, the projector is always in situ. Chan *et al* [9] have developed a system that makes use of a series of ceiling mounted projectors, which remain in place throughout runtime, and *steerable* projectors that are able to be controlled by the system users. The steerable projector's role is to overlay the projection surface with more information about what the user is looking at. It may provide resolution or intensity enhancement

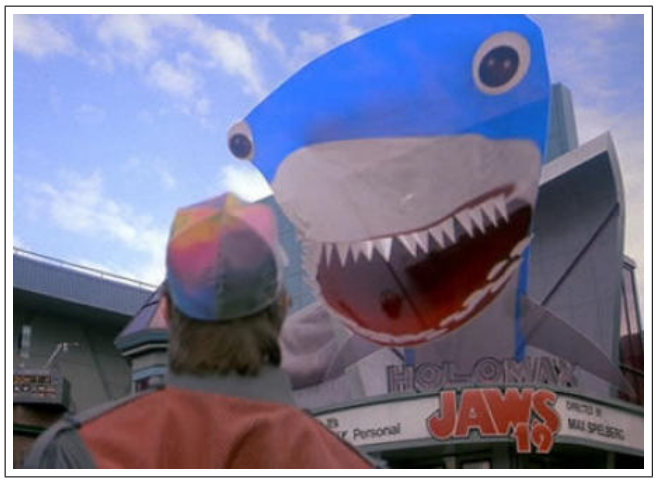


Figure 2.4: Example of advertising potential. Reprinted from [8]

for sub regions of the overall projection surface, thus essentially augmenting the augmented space. The user can interact with the system using personal digital assistants (PDAs), while a camera, also mounted to the ceiling also tracks potential interaction.

Kondo *et al* [21] and Tardif *et al* [43] use free form projection technology to project medical data onto patients. This could mean superimposing an X-Ray or CAT scan onto a patient's body to provide the surgeon with more information regarding an internal injury. A leading car manufacturer, [6], is using similar technology as an educational tool for trainee mechanics.

Of course, while offering these new and emerging applications for projections technology, it's imperative not to overlook the traditional use for projectors. A common application for projection technology would be public presentations. Whether for education or for business, projectors are a common commodity to have. So with the evolution of projection technologies, so too can you expect the evolution of presentation setups. Sukthankar *et al* [40] offer a novel and easy way to calibrate an arbitrarily placed projector by an arbitrarily placed camera, aiding quick set up times and versatility. Johnny Chung Lee, [23], presents a low cost interactive white board that promotes collaboration on projected documents and content. To go one step further; using the holographic technologies referred to above, one could imagine teleconferencing with projected holograms of colleagues who are situated on the other side of the planet.

2.6 Research and Development Applications

This section will deal with the numerous actual implementations and hypothetical implementations of projection technology in the real world. This section discuss the applications and their potential, rather than their technological makeup.

Figure 2.5: 6th Sense Technology, MIT. Reprinted from [30]

2.6.1 Pervasive Media

Pervasive Media spans a wide gamut of new media endeavours. Projection technology plays an incredibly important role in many of these areas, whether it's social and pervasive gaming, augmented reality and architecture, intelligent spaces, information communication and advertising, or even the fine arts. Whether it is a 30,000 ANSI lumens projector for mass communication, or a *pico-projector* for mobile applications, projection technology may provide an ideal solution.

Massachusetts Institute of Technology (MIT) has invested heavily in the research and development of pervasive technologies. The MediaLab's *6th Sense* [30] technology combines mobile phone, digital camera and pico-projector technology into an individual *smart assistant*-like device. The camera in the device acts as both an interaction input, but also as a standard image capturing device. The user uses hand gestures to operate, while the attached projector, displays the information upon a convenient projection surface (Figure 2.5). Through a mobile web connection via the phone, the 6th Sense can also use feature recognition capabilities (camera input) to recognise objects and retrieve relevant data from the world wide web and display it upon the object, creating an augmented reality internet.

Figure 2.6: SenseAble City, MIT. Reprinted from [34]

Another research area in MIT is the *SENSEable City Laboratory* [34] where the researchers seek to present relevant real-time information to city dwellers. Using state-of-the-art sensor technology, mobile devices and embedded displays, the laboratory researches and develops new and innovative means to

Figure 2.7: Architectural Projections. Reprinted from [45]

present real-time information. An example of this is their *EyeStop* [13] development to be unveiled in Florence, Italy in November, 2009. These varying devices (Figure 2.6) will present interactive services, community information and entertainment.

These pervasive, public technologies present an ideal opportunity for marketers and advertisers. With affordable projection technologies together with well calibrated projection software, interactive marketing in the physical world is now becoming a reality. *Reuters Media Group* [32] reported in 2008 of the partnership of a leading international advertising company and high-end projector manufacturer to create advertisements that would be projected within the London Underground train stations. This form of advertising may well prove to be a more economically viable way of marketing products. Less paper based posters, together with fewer aesthetically displeasing billboards may create a more positive reaction than the large paper based solutions of old. Even more interesting still, is the prospect of turning advertisements *off* between certain times.

However, with new computer vision technology, the ethical use of such technology will be under severe scrutiny. *New Scientist* [35] reports of fears of such potentially "Orwellian" technology. Technology that can record crowd numbers, locations of interest, a person's gender or even age may have an unsettling effect. The question of whether the technology is a *novel interaction* or a *surveillance* tool must be asked.

2.6.2 Architecture

With modern interactive graphics and graphics hardware devices, real-time graphics can be applied and projected within the physical environment, creating ever changing, augmented buildings and spaces. Bimber and Raskar [5] have extensively documented methods of creating *spatially augmented* spaces, applying real-time graphics to change the dimensions of the built environment, creating more internal space or to vary the texture or colour of an exterior wall. Figure 2.7 displays examples of alternative architectural projections.

Figure 2.8: Performance Art Examples. Reprinted from [25]

2.6.3 Performance Art

For years it has been expected that computer generated special effects were confined to cinema or television screen. Only in special circumstances, such as large theme parks, would these effects be seen in the physical world. The advancement in projection technology, specifically bulb brightness and energy efficiency, coupled with the evolution of holographic materials [46], [38] means that it is now common to see projection technology in theatre, opera and live musical productions (Figure 2.8).

2.7 Summary

A non-exhaustive list of potential applications of projection technology has been presented. This list provided both actual applications of new age projection technology, and proposed hypothetical employments of such apparatus in a commercial setting. This dissertation will now present a design specification for a versatile projection calibration framework. The design of which is based on the state-of-the-art discussed in this Chapter.

Chapter 3

System Design

This Chapter will provide an overview of the design steps taken to create and implement the calibration framework. The overall goal of the framework will be presented, before an introduction to the individual components of the calibration framework is made.

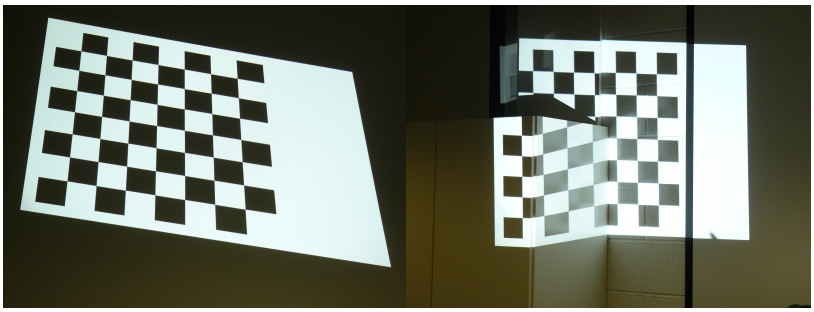
3.1 System Requirements

This calibration system is a framework that can easily calibrate a set of projectors to display large-scale image or video sequence with a camera device. The camera will be located at the audience's *sweet-spot* [5], viewing the projection from their perspective. This system will automatically correct for *keystoning* artifacts that occur when a projector is not aligned orthogonally to the projection surface. The system will also calibrate a series of projectors to display a coherent image on a non-planar, or free form projection surface, thus creating a virtual projection canvas.

3.1.1 System Goals

The goal of this system is to solve for the following discrete steps:

1. Calibrate a set of camera systems to eliminate lens distortion
2. Calibrate the projector:
   (a) Automatically correct keystoning artifacts if the projection surface is planar (Figure 3.1 (a)) using inverse homography function
   (b) Extract 3D surface geometry to remap image (Figure 3.1 (b)) using Gray-code structured light
3. Divide projected image frame across n projectors
4. Prewarp image in real time, exploiting accelerated graphics hardware



(a) Keystone Distortion (b) Geometric Distortion

Figure 3.1: Calibration state machine

3.2 System Design

The calibration system will need to employ underlying image processing, camera input and graphics functionality. Therefore, *Openframeworks*, to be discussed, is availed of to supply solutions. With a design based on previous research and state-of-the-art, this system has been structured as follows:

Camera Calibration

The camera calibration will utilise Zhang's [49] calibration model to compensate for lens distortion. This method has been used and tested extensively, making it a robust solution for this system. It is included in OpenCV's functionality, and will be straightforward to implement.

Automatic Keystoning

The first projection calibration issue that this system will address is the potential problem of image keystoning on a planar projection surface caused when the projection device is not orthogonally aligned with the surface. As a solution, this system will replicate methods used in Sukthankar *et al* [39] [40]. The system will calculate the homography between camera and projector by projecting a calibration target on the projection surface. The camera can capture this and using the same functionality used in the camera calibration step, solve the camera-projector homography.

Sukthankar *et al* reported very good results from their implementation in terms of setup time and accuracy, while also succeeding in being far more intelligible than alternative but similar systems. Unlike Sukthankar *et al*, however, this implementation will assume that the camera is located at the point of the audience. This may permit a more straightforward setup, without adding any excessive setup complexity.

Depth from Structured Light

When projecting upon a non-planar surface, it will be necessary to assess the geometric make up of that surface. Structured light is ubiquitously used for this purpose, and there are numerous implementations

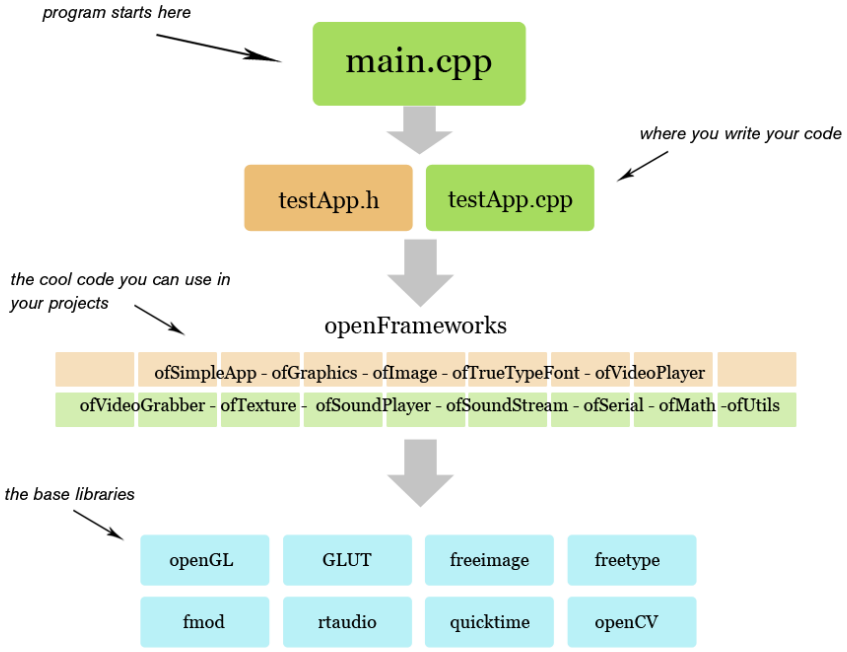


Figure 3.2: Technical architecture of *Openframeworks* [24]

of it in similar applications. For this system, we will adopt Gray-coded binary ([22], [18]) for it's robustness over plain binary code.

Tardif *et al* [42] *et al* provide a framework of which this system will adhere to. As Tardif *et al* utilise plain binary coded light, a comparison can be made between the two methods. Both of these systems are calibrated offline first, before the main application is run.

3.2.1 System Architecture

The system architecture is built upon *openframeworks*, a C++ based, cross platform framework with a particular bias for *creative coding*. Openframeworks (OF) is composed of a number of application programming interfaces (APIs) and libraries, and wraps up their main functionality for ease of use.

The core libraries the OF framework cover graphics programming, sound processing, vision and image processing, typography and video playing or sequence grabbing. The core libraries specific to this work are *OpenGL*, for graphics programming and graphics card accelerated performance. *OpenCV* for image processing capabilities and Apple's *Quicktime* for video playback and webcam access.

Figure 3.2 displays the technical architecture of OF. The core API frameworks create the lowest level, upon which OF's own wrappers sit. The classes that are added to this framework as a part of this research sit on top of these again, between the second and third layer. These classes are as follows (figure 3.3):

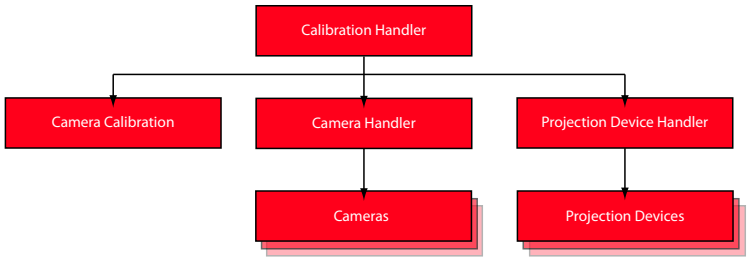


Figure 3.3: Calibration Framework

The *calibration handler* is responsible for the management of the calibration process for both projectors and cameras. This class is responsible for the initialisation of the camera and projector handlers, and in turn, the cameras and projectors themselves. The calibration handler is also responsible for finding the homographies between its subclasses, and the loading and saving of data to XML files.

The *camera handler* looks after each of the cameras in the system. This handler will detect how many cameras are currently connected, and store the intrinsic and extrinsic parameters of each camera. The camera handler provides easy access to any of the connected camera devices. There is one camera handler per application. Parameters specific to each camera are stored in the *camera* object class. These refer to the camera's intrinsic and extrinsic matrices.

The *camera calibration* class is responsible for the calibration process of the camera, as discussed in Chapter 4. The user will take images of a calibration target, and rectify the radial distortion caused by the lens. There is one camera calibration class per camera object.

3.2.2 System State-Machine

The system is modeled as a simple state-machine, with the user opting for a specific state to complete a certain task. The state-machine is designed so that the user goes through the relevant procedure to ensure that the projection devices are calibrated.

At runtime, the application starts in the *neutral state*. At this point nothing happens, it is merely an opening screen for the application. The user can then navigate to the *camera calibration state* (Chapter 4) where the calibration of each camera takes place. The camera calibration screen displays an input from each camera device. If the cameras' frusta overlap, then the cameras can be calibrated in unison, or otherwise individually.

The *projector calibration state* (Chapter 5) looks after the projector calibration with regards the camera. The projector-camera homography is calculated at this stage if the projection surface is planar, otherwise the 3D acquisition of the projection surface is taken using structured light (Chapter 6).

The display image will have to undergo a remapping so that it will appear rectified and coherent from the perspective of the camera. These remapping parameters will be passed from the previous state and transform the display image accordingly in real-time.

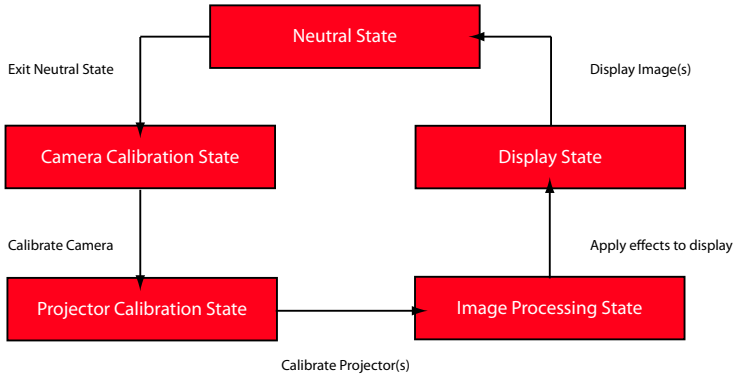


Figure 3.4: Calibration state machine

3.3 Evaluation

The system will be tested against numerous metrics. In the first instance, visual inspection can be used to get a good idea as to how the system has performed. This will not be enough in itself, however, and other tests have been devised to assess the system's performance. The evaluation of the system is discussed in Chapter 7.

3.3.1 Calibration Accuracy

Using camera image space, the accuracy of the calibration can be measured by assessing the alignment of the the projected image. Sukthankar *et al* [39] [40] claim that their system is accurate to ± 3 pixels, but on a camera resolution of 160x120 pixels. Projectors offer digital keystoning rectification, but typically within $\pm 12^{\circ}$. This sets the goal for calibration accuracy on a planar surface.

3.3.2 Object Tracking Consistency

While the image may be nicely aligned on both planar and free-form surfaces, there may be unwanted warping present within the image caused by inaccurate point interpolation, inaccurate geometry extraction or misalignment between two projectors. To test for this, an object tracking test is performed to assess the variance of points within the projected image. No other previous paper has used such a metric to test their system.

3.3.3 Performance Benchmarks

In this test, the performance of CPU based operations will be compared to GPU accelerated alternatives. OpenCV offers functionality to warp images based on a 3x3 matrix while running on the CPU. This warping can be changed to a 4x4 matrix and used within OpenGL to warp the projection matrix, utilising accelerated hardware.

Chapter 4

Implementation: Camera Calibration

This Chapter will provide an overview into the research and development that went into the camera calibration component of this projector calibration framework. The framework uses Zhang's [49] method of camera calibration for its ease and accessibility through OpenCV. Additional information has been referenced from Bradski and Kaehler [7].

4.1 The Pinhole-Camera Model and Calibration

4.1.1 Projective Geometry

To assess the 3D world via a camera, we adopt the pin-hole camera model. The pin-hole camera's aperture is a single point, without a lens, which means that everything viewed will be both in focus, and is thus not subjected to distortion caused by a lens. This camera model maps a 3D coordinate in *World* space to a 2D coordinate on the *camera image plane*. In mathematical terms, we refer to World coordinates as $\mathbf{M} = [X, Y, Z]^T$ and the 2D image plane as $\mathbf{m} = [u, v]^T$. For projective transformations, the homogeneous vectors for both 2D and 3D coordinates are used. This creates new vectors $\tilde{\mathbf{m}} = [u, v, 1]^T$ and $\tilde{\mathbf{M}} = [X, Y, Z, 1]^T$ for image and world coordinates respectively. We now model the projective mapping between $\tilde{\mathbf{m}}$ and $\tilde{\mathbf{M}}$ as:

$$s\tilde{\mathbf{m}} = A \begin{bmatrix} R & t \end{bmatrix} \tilde{\mathbf{M}} \tag{4.1}$$

Where s is an arbitrary scale factor. A is referred to as the camera's *intrinsic matrix*, while matrix $[R, t]$ is considered the *extrinsic matrix* of the camera.

$$A = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

The intrinsic matrix, equation 4.2, contains variables specific to the camera and do not rely on the 3D

space sampled. In equation 4.2, α and β signify the scale factors on the u and v axes in the image, this is related to the closeness of the viewed object. γ, then, represents skewness of the two image axes. Coordinate u_0 and v_0 is the image's principle point (the centre of radial distortion).

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

The intrinsic matrix is often represented as in equation 4.3. The camera's focal points, f_x and f_y, are represented in pixel values, while the image's optical centre, or principal point, is also represented in pixel values c_x and c_y.

The extrinsic matrix $[R, t]$ is so called as it describes the camera's external parameters rotation (R), and translation (t) in real world coordinates from a particular point. This value is dependent on the 3D space sampled. Its expanded notation is:

$$\begin{bmatrix} R & t \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_1 \\ R_{21} & R_{22} & R_{23} & t_2 \\ R_{31} & R_{32} & R_{33} & t_3 \end{bmatrix} \tag{4.4}$$

These expressions can be simplified further with the assumption that the model plane (calibration pattern) is along an axis where $Z = 0$. Therefore Z, now 0, can be disregarded. We can then assume the following from equation 4.1:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A \begin{bmatrix} r_1 & r_2 & r_3 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.5}$$

then becomes:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{4.6}$$

By analysing equation 4.6, it is clear that $\tilde{\mathbf{m}}$ is related to $\tilde{\mathbf{M}}$ by a 3x3 matrix. This matrix is referred to as a *homography* (**H**). As this is a *square* matrix, it is invertible and is a characteristic which is utilised in Chapter 5. We can finally state:

$$s\tilde{\mathbf{m}} = \mathbf{H}\tilde{\mathbf{M}} \tag{4.7}$$

Where **H** is:

$$\mathbf{H} = A \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \tag{4.8}$$

The camera's intrinsic parameters are subject to two constraints given a single homography. As a

homography has 8-degrees of freedom, only two constraints can be obtained from the six extrinsic parameters (3 rotation vectors and 3 translation vectors). These are as follows:

If $\mathbf{H} = \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix}$, then from 4.8, we get:

$$\begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix} = \mathbf{sA} \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \tag{4.9}$$

Where **s** is a scale factor and rotation vectors r_1 and r_2 are orthonormal. This then leads to the two constraints on the intrinsic parameters:

$$h_1^T \mathbf{A}^{-T} \mathbf{A}^{-T} h_2 = 0 \tag{4.10}$$

$$h_1^T \mathbf{A}^{-T} \mathbf{A}^{-T} h_1 = h_2^T \mathbf{A}^{-T} \mathbf{A}^{-1} h_2 \tag{4.11}$$

4.1.2 Lens Distortion

Using a camera which has a lens, we can be expected to encounter both a dominant radial distortion and slight tangental distortion. Radial distortion occurs as the rays further from the lens centre are bent more than those closer to the centre. Tangental distortions are related to the physical assembly process of the camera. Figure 4.1 provides a visual example of lens distortion.

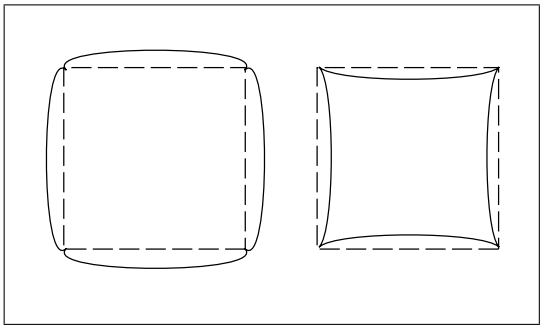


Figure 4.1: Camera Calibration: Barrell (left) and pillow (right) lens distortions

The distortion of the lens can be modeled by a Taylor series. The first two terms, k_1 and k_2, are usually enough to model the distortion, however, if large distortions are present (from a fish-eyed lens) then the third term, k_3, can also be used to extend its accuracy. The following is the mathematical model of distortion as of Zhang (1998).

$$\breve{x} = x + x[k_1(x^2 + y^2) + k_2(x^2 + y^2)^2] \tag{4.12}$$

$$\breve{y} = y + y[k_1(x^2 + y^2) + k_2(x^2 + y^2)^2] \tag{4.13}$$

Where (u, v) are the ideal, undistorted coordinates in the image plane, and $(\breve{u},\breve{v})$ are the distorted image coordinates, and (x, y), $(\breve{x},\breve{y})$ are the normalised, undistorted and distorted image coordinates,

respectively. k_1 and k_2, as mentioned, are the radial distortion coefficients. With equation 4.2, we can deduce that:

$$\breve{u} = u_0 + \alpha\breve{x} + \gamma\breve{y} \tag{4.14}$$

$$\breve{v} = v_0 + \beta\breve{x} \tag{4.15}$$

And taking $\gamma = 0$:

$$\breve{u} = u + (u - u_0)[k_1(x^2 + y^2) + k_2(x^2 + y^2)^2] \tag{4.16}$$

$$\breve{v} = v + (v - v_0)[k_1(x^2 + y^2) + k_2(x^2 + y^2)^2] \tag{4.17}$$

The webcams used in this project do have lenses and will therefore be subjected to lens distortion. We will calibrate them to simulate a pin-hole camera model. The camera and projector will then be treated as an uncalibrated stereo pair, whereby the relative positioning of each device is unknown. The rectification of the lens distortion is paramount in this implementation, the calculation of the extrinsic matrices are not important. This next section will present Zhang's method of camera calibration.

4.2 Camera Calibration: Zhang's Method

In the first instance, the calibration target (the chessboard pattern) is applied to a rigid surface. The feature points of the calibration target are assumed to lie on a single plane (the model plane). If the feature points do not reside on the same plane, the camera will attempt to rectify the image with inaccurate feature positions, resulting in distortion artifacts that could corrupt the intrinsic and extrinsic parameters, and hence what should be a corrected image. Then, by moving either the calibration target with respect to the camera, or vice-versa, a series of images of the calibration target from numerous orientations is captured. The movement need not be known. The feature points of the calibration target must be detected before the five intrinsic parameters and six extrinsic parameters are calculated. This is calculated by a *closed-form solution*. The radial distortion coefficients can then be estimated by solving the linear least-squares which is then refined by a *maximum likelihood estimation*.

4.3 Camera Calibration: OpenCV

Vision framework, *OpenCV*, covers all relevant functions for the successful calibration of a camera. The camera calibration component uses an 8x8 chessboard pattern (figure 4.2) as the planar object to detect, as discussed. The following functions are then called in succession to acquire the camera's intrinsic, extrinsic matrices and distortion variants.

1 `cvFindChessBoardCornerGuesses()`

This function estimates the location of the features upon the chessboard. This function looks for the

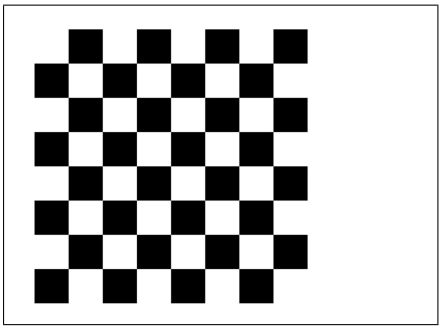


Figure 4.2: Camera Calibration: Calibration Pattern

possibility of a calibration target being present in the current image. For this framework, we use an 8x8 chessboard, with 7x7 internal corners. The function finds features where a black square meets another black square and the locations are approximately stored. For a more accurate detection, the following is used:

2 `cvFindCornerSubPix()`
Once the location of the chessboard's features have been estimated, this function then iteratively refines the location of the feature points to sub-pixel accuracy.

3 `cvDrawChessboardCorners()`
This function simply draws a graphical marker upon the location where a feature is deemed to have been found. This step can be used for the user to visually confirm the result.

4 `cvUnDistortOnce()` Finally, this function is called to redraw the image, compensating for both the tangental and radial distortions, passed as parameters as a part of the intrinsic matrix. Each pixel from the input image is mapped based on the distortion values and are linearly interpolated.

The results of the `cvFindCornerSubPix()` function step can be seen in figure 4.3. The final correction of the image can be seen in Figure 4.4

Figure 4.3: Camera Calibration: Feature identification

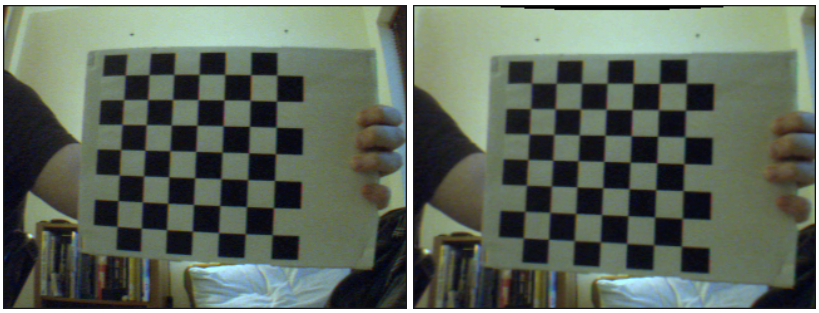


Figure 4.4: Camera Calibration: A non-calibrated image (left) and a calibrated image (right)

4.4 Camera Calibration: Framework

The framework is developed using both OpenCV and Apple Quicktime and is capable of handling numerous camera inputs, using Quicktime to enumerate each device at runtime. The application will then display each available camera input, allowing the user to calibrate the cameras. The calibration method proposed by Zhang, and presented above, will be carried out. The user will have to calibrate the cameras individually, if their view frusta do not overlap. Otherwise, both cameras can be calibrated in unison. The following code outlines how the camera devices are located using the Quicktime API. Code examples can be viewed in Appendix A where a screen shot of the camera calibration screen can also be seen. This is the first part of the calibration process.

For convenience, the application offers the user the opportunity to save the camera calibration parameters to an XML file. It was found to be a nuisance to have to re-calibrate each camera at runtime. This is acceptable as the distortion and intrinsic matrices of the camera do not depend on the scene being viewed. Using OpenCVs' `cvSave()` function call, each of the cameras' distortion coefficients and intrinsic matrix values are saved to a file and can be loaded at runtime, using `cvLoad()`, when the application is next executed. The next chapter will deal the calibration of the projector image with respect to one of the camera devices.

4.5 Summary

This Chapter presented the theory behind the camera calibration step used in this framework. Zhang's calibration method was described and its role and implementation in this framework discussed. Zhang's calibration method is fully supported within OpenCV and easily carries out the necessary functionality necessary for this dissertation. The camera calibration stage of the process is needed to make sure that the lens distortion of the camera is rectified and does not add its own warp to the projection rectification. The extrinsic parameters of the camera are not required for this implementation.

Chapter 5

Implementation: Projector Homography and Calibration

This Chapter will detail the process required to calibrate a projector from the perspective of a camera. By calculating the geometric relationship between the camera and the projector, the projector will undergo an *automatic keystoning* process. The camera is located at the audience's sweet spot, simulating their perspective of the projected image on the projection surface.

5.1 Homography: Overview

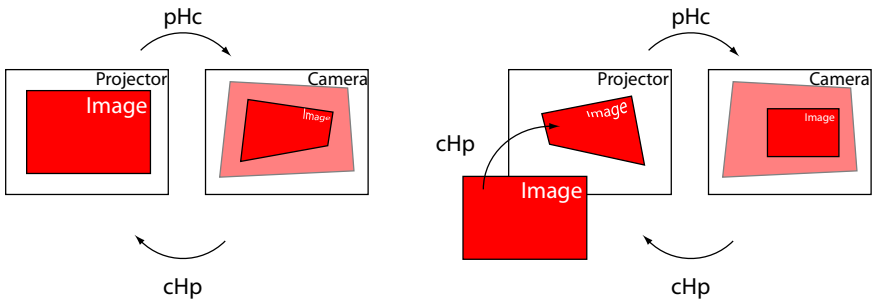


Figure 5.1: Overview of the relationship between camera, projector and projection surface

The relationship between an arbitrarily positioned projector, whose position and orientation are not known, and an arbitrarily positioned camera, whose position and orientation are also unknown, can be determined as long as the camera image contains the projected image on the projection surface. As the

camera can identify the projected image on the projection surface, a mapping between the ideal, projected image and that in the camera image plane can be deduced. This mapping is simply a perspective transform described as:

$$(u, v) = \left(\frac{h_{11}s + h_{12}t + h_{13}}{h_{31}s + h_{32}t + h_{33}}, \frac{h_{21}s + h_{22}t + h_{23}}{h_{31}s + h_{32}t + h_{33}} \right) \tag{5.1}$$

Where (s, t) is a pixel in projection space, and mapped to a pixel in camera image space (u, v). This transform can expressed in matrix form with homogenous coordinates:

$$\begin{bmatrix} \alpha u \\ \alpha v \\ \alpha \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} s \\ t \\ 1 \end{bmatrix} \tag{5.2}$$

Homography, H, is a 3x3 matrix, consisting of $h_{11} \dots h_{33}$, and α is a scaling factor. However, $h_{11} \dots h_{33}$ are each unknown variables of the homography H and must be solved for. In this instance equation (5.2) is expanded as follows:

$$\begin{bmatrix} \alpha u \\ \alpha v \\ \alpha \end{bmatrix} = \begin{bmatrix} h_{11}s & h_{12}t & h_{13} \\ h_{21}s & h_{22}t & h_{23} \\ h_{31}s & h_{32}t & h_{33} \end{bmatrix} \tag{5.3}$$

If we accept $\alpha = [h_{31}s, h_{32}t, h_{33}]$, we can create two simultaneous equations for u, v as follows:

$$u(h_{31}s + h_{32}t + h_{33}) = h_{11}s + h_{12}t + h_{13} \tag{5.4}$$

$$v(h_{31}s + h_{32}t + h_{33}) = h_{21}s + h_{22}t + h_{23} \tag{5.5}$$

We now have two linear equations with nine unknowns, as before. While there are nine unknowns in this equation, it can be assumed that each coordinate lies upon the same unknown plane. We can finally state that for n feature points detected, equation(5.4) can finally be written as a $2n$x9 matrix and solved using QR decomposition, a common method used to solve a least-squares function:

$$\begin{bmatrix} s_1 & t_1 & 1 & 0 & 0 & 0 & -u_1s_1 & -u_1t_1 & -u_1 \\ 0 & 0 & 0 & s_1 & t_1 & 1 & -v_1s_1 & -v_1t_1 & -v_1 \\ s_2 & t_2 & 1 & 0 & 0 & 0 & -u_2s_2 & -u_2t_2 & -u_2 \\ 0 & 0 & 0 & s_2 & t_2 & 1 & -v_2s_2 & -v_2t_2 & -v_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ s_n & t_n & 1 & 0 & 0 & 0 & -u_ns_n & -u_nt_n & -u_n \\ 0 & 0 & 0 & s_n & t_n & 1 & -v_ns_n & -v_nt_n & -v_n \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ \vdots \\ h_{23} \\ h_{33} \end{bmatrix} = 0 \tag{5.6}$$

As homogenous coordinates are used, it can assumed that there are 8-degrees of freedom as $|\vec{h}| = 1$. H can now be solved with four pixel correspondences, where each correspondence provides two constraints u, v ([37], [40]). More correspondences add robustness to the homography estimation. The 3x3

matrix is first estimated using *least-squares*, explained above, and is refined further using the *Levenberg-Marquardt* function fitting algorithm [17].

5.2 Automatic Keystone Correction Framework

This framework reuses techniques utilised in the *camera calibration* framework described in Chapter 3. To obtain the required feature points for the homography calculation, the calibration target (chessboard pattern) is reused to map pixel correspondences in the camera image space to the projection image space. In this setup, the projector is placed in an arbitrary position, while the camera is positioned at the sweet-spot of the audience, simulating their point of view.

5.2.1 Camera - Projector Homography

The homography between the camera and projector are calculated by the following steps:

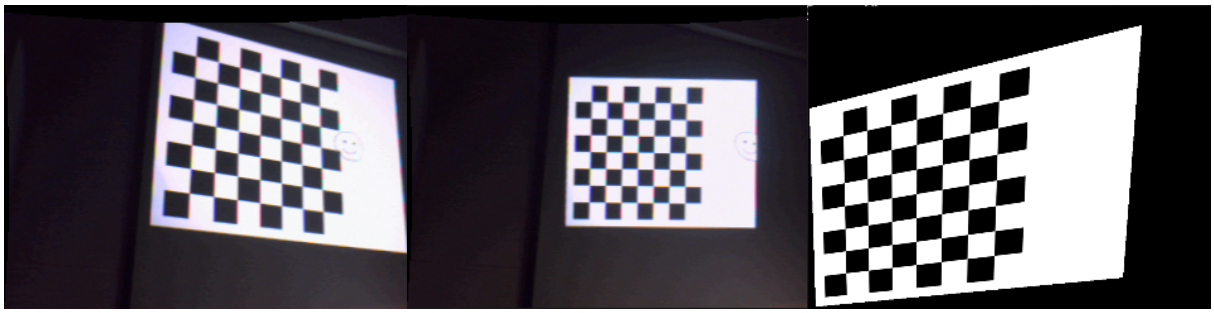
1. Project a calibration target (chessboard) onto the projection surface.
2. Detect the feature points to subpixel accuracy and store.
3. Use found feature points to determine 3x3 homgraphy between camera and projector via the display screen.
4. Apply the inverse of this matrix transformation to the projection image image.

The feature points are detected as before, locating the 7x7 internal squares of the calibration target. These points are then saved and passed to the OpenCV `cvFindHomography()` function, where the 3x3 homography matrix that maps the screen space to the camera image space is determined using the mathematical techniques described above. Once the mapping function has been successfully obtained, we can now apply the inverse homography transform to the ideal projection image, thus pre-warping it to appear undistorted to the camera's perspective. The sequence is performed as follows:

1 `cvFindChessBoardCornerGuesses()`
: As described in Chapter 4, this function finds the 7x7 internal corners and stores their coordinates in the camera image domain.

2 `cvFindCornerSubPix()`
: Again, once the location of the chessboard's features have been estimated, this function then iteratively refines the location of the feature points to sub-pixel accuracy.

3 `cvFindHomography()`
: By passing the initial source points of the calibration target along with the destination points (the ones uncovered by the previous function), this function is capable of finding the homography 3x3 transform that maps the source points to the destination points. RANSAC is used to find five random



(a) Uncalibrated, keystoned image (b) Rectified image (c) Pre-warped projected image that is projected in (b)

Figure 5.2: Camera devices perspective of the uncalibrated projection, the rectified image and the image projected to appear correct from the camera device's point of view

corresponding pairs to estimate the homography where a least-squares function assesses the quality of the outcome. The homography is further refined by applying a function fitting Levenberg-Marquadt method.

4 `CvWarpPerspective()` This function is used to apply a 3x3 warp function to an image. The particular image in question is passed, along with a destination image and the 3x3 matrix. The warp is carried out on the source image and the result is saved in the destination image. The points in the image are linearly interpolated. If only four points are available to identify, `cvGetPerspectiveTransform()` can be used instead.

It is at this point that the use of OpenGL and GPU acceleration can be exploited. Instead of passing two image files to the `cvWarpPerspective()` function, the newly found homography, which is only computed once, can be converted from its 3x3 *row-major* to the OpenGL standard 4x4 *column-major* structure as follows:

$$\begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} becomes \begin{bmatrix} h_1 & h_4 & 0 & h_7 \\ h_2 & h_5 & 0 & h_8 \\ 0 & 0 & 1 & 0 \\ h_3 & h_6 & 0 & h_9 \end{bmatrix}$$

This new 4x4 matrix can then be multiplied with the OpenGL scene's projection matrix to warp the scene and achieve real-time rates. Figure 5.2 displays the keystoned and rectified projections and also the pre-warped image projected to create the rectified projection. These outputs were warped manually.

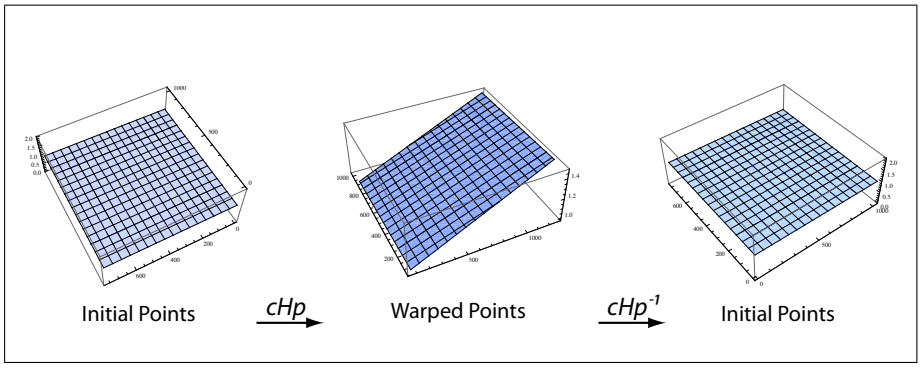
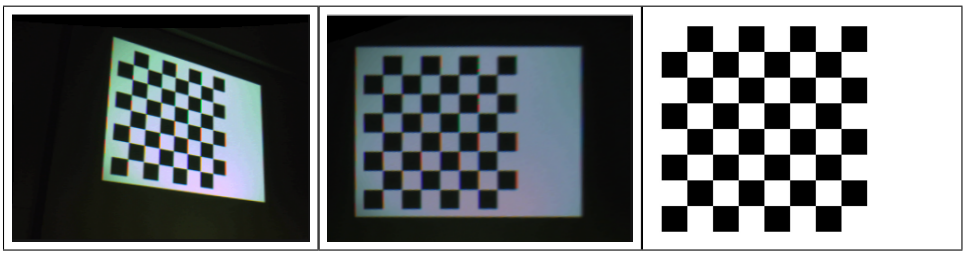


Figure 5.3: A warp transform applied to 4 points can be rectified by applying the inverse transform

5.3 Erroneous Results

Problems were encountered at this stage of the project. In this implementation, the camera device is located at the audience's location (audience sweet-spot) and therefore one transformation between the projected image and the camera device should suffice to rectify the image to appear coherent to the audience. In Sukthankar[39][40], the projection screen's orientation must be taken into account in the rectification function as that implementation expects the audience to be orthogonal to the projection surface and the camera to be arbitrarily located elsewhere. In this dissertation, the screen's orientation should not play a part in the rectification of this dissertation's calibration framework.

Figure 5.3 demonstrates that it is possible to rectify an image, which has been subject to a perspective warp, by applying the inverse warp function. There then may have been the possibility that the homography function calculated may be incorrect. Figure 5.4 displays the projection from the perspective of the camera device (a), and the rectification test to make sure the the homography inverse function was correct. By visual inspection, it is clear to see that the rectified image, (b), is a very similar to the calibration target (c).



(a) Warped image seen by camera (b) Rectified image in camera plane (c) Calibration target

Figure 5.4: A warp transform applied to 4 points can be rectified by applying the inverse transform

The homography transform was rigorously tested using Wolfram Mathematica 6 to confirm that the transformation under the 3x3 matrix was in fact correct. Code samples and graphical outputs from this

notebook can also be found in Appendix B, where the theory and the implementation are fully documented in Appendix B.2 and the manual image warping function used instead is documented in Appendix B.3.

5.4 Automatic Keystone Correction Framework - Multi Projectors

Due to time constraints, the successful implementation of the multi-projector component for this aspect of the system was not carried out. Had more time been available, its implementation would be relatively straightforward. The exact process outlined above would be carried out by each projector sequentially for each n projectors. The projection of the of the images from each projector could be performed in two ways. First, if the aim of the system was to eradicate the potential of full occlusion by an object, then each projector would project the exact same image, onto the exact same area. Instead of one full occlusion, there would be n $\frac{1}{n}$ shadows, where n is the number of projectors. Alternatively, if the goal was a large scale display, the image to be projected would be divided by creating n *framebuffer objects*. Each projector would then be responsible for the projection of the image data in its designated framebuffer object. The photometric calibration needed to correct colour artifacts is beyond the scope of this project.

This calibration process is successful for the automatic keystoning of a projection image on a planar projection surface. This technique can not be applied to a free-form, or non-planar surface, as it cannot be taken for granted that the feature points will lie on the same plane. To solve for projector calibration for free-form surfaces, another technique must be investigated and this is discussed in Chapter 6.

5.5 Summary

This Chapter has provided a theory for automatically correcting keystone correction caused by a misaligned projector from the perspective of a camera located at the audience's sweet-spot. Mathematics has been presented to confirm initial hypotheses, however these steps were not successfully implemented. Details of the errors encountered are documented, as too are attempted solutions. To overcome this problem, a manual warping tool was created.

Chapter 6

Implementation: Structured Light

This Chapter will outline the implementation of the structured light component of this dissertation. It will start by providing a general outline of the method and why it is used, before an explanation of how the method is implemented into this dissertation is detailed.

6.1 Structured Light: Overview

Structured light's purpose in 3D depth extraction is the creation of a dense number of feature point correspondences with which the camera device can detect. A known light pattern, or series of known light patterns, is projected into the scene and encode each pixel in the camera image plane with a colour value. The most common case of structured light is the projection of black an white stripes that continue to grow finer with each successive step. The black and white patterns then identify each pixel location in the camera image domain, giving each pixel a 0 or 1 value for each pattern projected. Over a series of projected patterns, each pixel accumulates a codeword which then corresponds to a pixel in projected image space. To assist an arbitrary projector-camera setup, both vertical and horizontal patterns are projected. The two main encodings of structured light are binary encoding and the more robust Gray-code encoding.

6.1.1 Binary and Gray-code Structured Light

Decimal	Binary	Gray-code
0	000	000
1	001	001
2	010	011
3	011	010

As was discussed briefly in Chapter 2, Gray-coded binary is preferred to binary coded light for robust encoding. Unlike in binary code, the boundary of each stripe does not occur in the same place more than once. In binary code, the boundary will remain in the same position for the duration of the projections, meaning that the pixels located at these locations become encoded incorrectly thus corrupting the

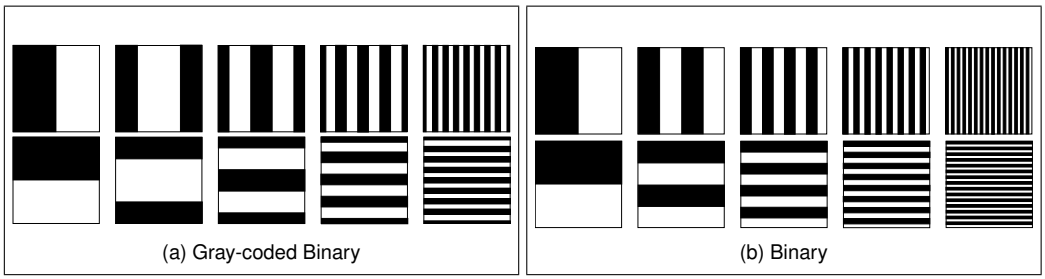


Figure 6.1: Vertical and Horizontal structured light patterns

integrity of the correspondences. This is not the case with Gray-code. Gray-code also has a relationship of $O(log_2((n))$ to the number of necessary patterns to project onto a scene. In the case of a 1024x768 XGA projector, only 20 images are needed to uniquely encode each pixel to be projected. Figure 6.1 presents an example of both Gray-coded binary and binary coded light patterns.

6.2 Structured Light: Implementation

The implementation of the structured light process in this dissertation is broken into separate components. First, the current pattern must be projected into the scene. This is followed by the image capture by the camera, where the camera can see all of the projector's frustum on the projection screen. Then, the images must be thresholded to binary values so that 1 or 0 values can be attributed to the pixel in question. A mapping construction is then created between the projected image and the camera image plane. This is followed by an inverse mapping and finally the mage construction, where the image is pre-warped in real-time using OpenGL and the GLSL shading language to utilise the GPU.

6.2.1 Pattern Projection

The fundamental part of the structured light step is the correct projection of the binary pattern. First, the amount of patterns to be projected must be known. As a 1024x768 image needs to encoded, enough patterns to encode 768,432 pixels must be chosen. Gray-code projection is of the order $O(log_2((n))$, therefore 20 patterns are needed to uniquely encode each projected pixel, as $2^{20} = 1,048,576$. In this implementation, where an arbitrary camera-projector can be expected, ten vertical patterns and ten horizontal patterns are projected into the scene.

To set this up, decimal numbers $N_d = [0 \dots 1023]$ are converted to their binary equivalent $N_b = [2^0 ... (2^{10} - 1)]$. These binary values are then used to convert directly to their Gray-code equivalent by the following algorithm [27]:

- The binary code number has n-bits, so the Gray-code number will have n-bits.
- The most significant bit (MSB) is the same in both binary and Gray-code

- The second MSB in the binary number is calculated by adding the MSB in the binary number to the second MSB in the Gray-coded number, the sum is noted and any carry ignored
- Working from left to right, from MSB to least significant bit (LSB), the n-th bit (counting from right to left) in the binary number is formed from summing the $(n+1)$th bit in the binary number with the nth bit in the Gray coded number. The corresponding sum is noted and any carry ignored.

We now have our 10-bit Gray-code sequence, $N_g = [0...(2^{10} - 1)]$. This sequence is then then used to draw the necessary stripes. The width of the stripe is dependent on the on the highest value, in this case 1024. 1024 stripes are drawn, and their colour is dependent on the current bit-value to be projected. If the bit-value is 0, then a black stripe is projected, if the bit-value is 1, a white stripe is projected. The horizontal stripes are projected first, and this rendering algorithm is simply repeated with the x, y values swapped at draw time.

6.2.2 Image Capture

The image acquisition component's role is to capture an image of each pattern projected up the projection surface. Memory is allocated for 20 images to be stored initially (this is reviewed and later changed) and at each point when the pattern is projected, the image is captured and converted to grayscale for later thresholding. The pattern will only be incremented when the pattern has been successfully captured by the camera. 20 grayscale images have now been stored in memory and await thresholding.

6.2.3 Image Thresholding

To attribute each pixel in camera image space with a 1 or 0 value, thresholding must be carried out. For each of the 20 images saved in grayscale, these images are individually thresholded according to a suitable threshold value. It was found, however, that for the images representing the LSBs, the resolution of the camera device was not high enough and the integrity of the thresholded values, as a result, were clearly in doubt. This would certainly have had detrimental consequences on the final calibration results. The thresholded outputs can be seen in Figure 6.2.

At this point a better form of thresholding the images had to be implemented. To do this, Tardif *et al* [42] present a more robust form of thresholding. In this thresholding step, both the structured light patterns and their respective inverse patterns are projected into the scene. So if a bit projected initially was a 0 (ie black) then the inverse pixel value will be a 1 (ie white). The images are captured as before. The ten vertical images are follows by ten horizontal patterns, this is then followed by their inverses. To do this, $n \times 2$ patterns are projected, and therefore 40 images are stored in this implementation rather than the initial 20.

The images are each stored in grayscale as before and the thresholding occurs by subtracting the inverse pattern by from its converse. This happens for each of the 20 images to result in 20 new difference images. These values store the difference of the images from -255 to +255. We would expect to see the pixels covered by the projection frusta to have the most extreme values of $\pm$ $\sim$255, with the pixels not covered by the frusta to have a much smaller value, typically between $\pm$50.

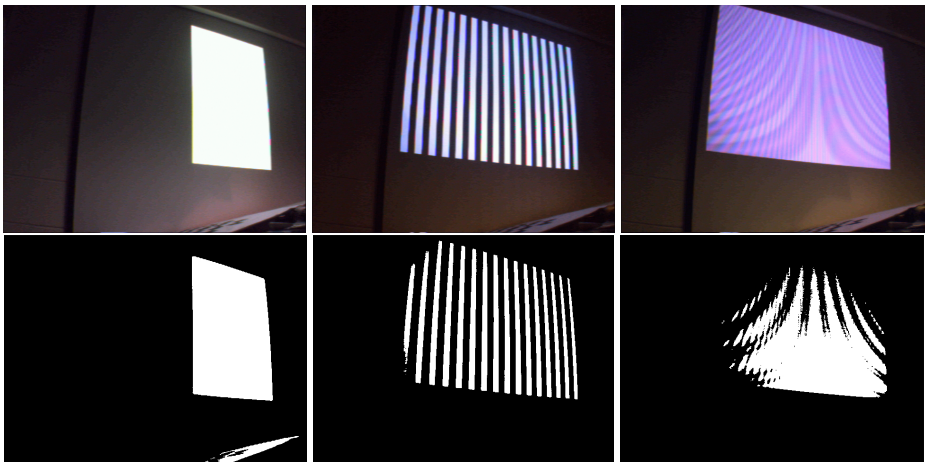


Figure 6.2: Bit pattern 10, 5 and 1 respectively. The colour image capture can be seen on top, and its corresponding thresholded image is beneath.

The threshold value is calculated empirically by studying the distribution of the pixel values in the difference images and the threshold value of ± 100 was chosen.

```
1    if (pixel_value(u,v) > T) pixel_value(u,v) = 1;
2    if (pixel_value(u,v) < -T) pixel_value(u,v) = 0;
3    else rejected
```

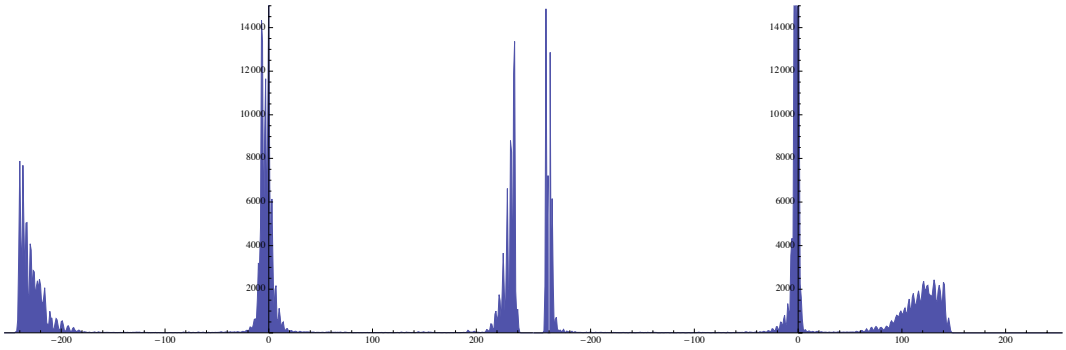
The distribution histograms can be seen in Figure 6.3. This technique provided a far more robust thresholding response than the initial implementation. Problems encountered included blooming effects from the white stripes which lightened the black areas and corrupted the outcome, this was greatly alleviated with the Tardif *et al* implementation. This method also excluded any outliers (pixels not included within the projection frustum), as light pixels in the surrounding areas could be erroneously encoded. The distribution histograms generated by this implementation echoed their outcome almost exactly.

6.2.4 Mapping Construction

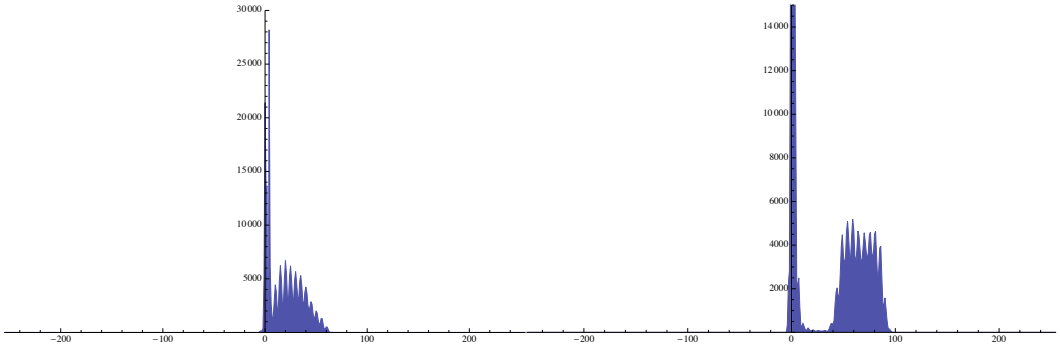
The mapping, R between projector and camera space, is constructed by concatenating the ten horizontal bits, from MSB to LSB, to the ten vertical bits, MSB to LSB and it takes the form:

$$R(u,v) = [R(u,v)^v_{n-1} \dots R(u,v)^v_0], [R(u,v)^h_{n-1} \dots R(u,v)^h_0]$$

Where R is the mapping for a particular pixel (u,v) and n is the number of patterns projected. As discussed before, however, the LSBs lose integrity as the frequency of of the stripes increase in later patterns. As a result, these bits cannot not be counted in the final mapping between projector and camera pixels as the risk of error will be too high. Figure 6.4 displays the sudden fall-off of usable bits after



(a) Bit 10 - Vertical Distribution (b) Bit 10 - Horizontal Distribution



(c) Bit 1 - Vertical Distribution (d) Bit 1 - Horizontal Distribution

Figure 6.3: Distribution of bit values of bit 10 (top) difference images and bit 1 (bottom)

thresholding. We know they are unusable as their value falls under the threshold permitted (ie -100 $\geq$ $pixel\_value(u, v) \leq$ +100).

As can be seen from Figure 6.4, the quantity of usable bits declines rapidly after the 7th pattern projection. Knowing this, the subsequent bits are set to 0, creating a new mapping function R':

$$R'(u, v) = [R(u, v)^v_{n-1} \dots R(u, v)^v_{n-n'} 0 \dots 0], [R(u, v)^h_{n-1} \dots R(u, v)^h_{n-n'} 0 \dots 0]$$

Where bits $n - n'$ are set to 0 and n' is the number of unusable pixels. This creates a *down-sampled* mapping (R') of pixels from projector to camera image space. Instead of dealing with 1024x768 points, the new mapping R', rebuilt with 7 bits creates a grid in the dimension of 128x96 points. These points can be used as texture coordinates to essentially texture camera image space. Figure 6.4 is, again, a near exact result of the same step carried out in Tardif *et al* [42].

6.2.5 Inverse Mapping

The first stage of the inverse mapping will differ from Tardif *et al*'s implementation. As Gray-code structured light has been adopted for this implementation, it will be necessary to convert from Gray-code to binary

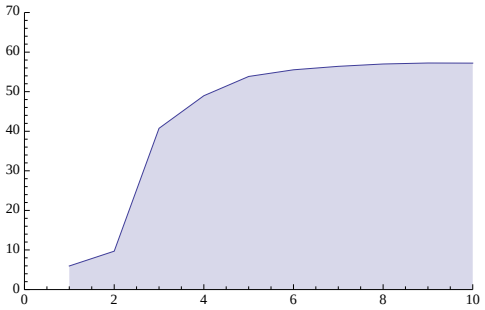


Figure 6.4: Percentage of usable pixels based on the different stripe widths. The number of usable pixels falls sharply after bit 4

code at this point so that the correspondences can be made. This step is simply the reverse of what was implemented to convert binary to Gray-code in the projection stage. The mapping R, and hence R^{-1}, from projection image space to camera image space is now ready to be computed.

For this step, the projector image plane (s, t) and the camera image plane (u, v) are normalised as they are of different resolutions, but are of the same aspect ratio. Like in [42], a set of pixels within the camera image plane, $S_c(s, t)$ are defined where the thresholding step implemented previously eradicates any pixels that do not lie within the projection frusta. An alpha map is created, masking the outliers. A set of pixels $S_p(s, t)$ represents the down-sampled pixels in the projection image.

$$S_c(s, t) = \{(u, v)|R^{-1}(u, v) = (s, t)\}$$

Then, for each down-sampled position in the projection image space, the point's corresponding position is located in camera image space. This is carried out by matching the down-sampled pixel's index in projection space to its now binary code-word equivalent in the camera image domain. A mapping R will describe the relative positions of each pixel where R is the displacement of a pixel between $S_p(s, t)$ and $S_c(s, t)$.

This mapping R is essentially a displacement map between $S_p(s, t)$ and $S_c(s, t)$. This will then stored as a 2D texture, where the s-displacement is stored in the R channel and the t-displacement stored in the G channel of the texture as the projection image will be warped. This texture is to be used in the image construction, discussed next.

6.2.6 Image Construction

The image construction is carried out using OpenGL and GLSL to utilise the GPU in the computer system. This will allow real time pre-warping of the projection image, upholding the fasted FPS rate possible. The displacement map, stored as a texture, is passed to the vertex shader where a 2D texture look-up is carried out to fetch the values stored in the R and G channel of the texture respectively. OpenGL normalises parameter values as they are passed to the respective GLSL shaders. Therefore, the values are pre-normalised before being passed, and then a multiplication by 1024 in the R channel value (s-

displacement) and 768 in the G channel (t-displacement) to un-normalise the displacement vectors. This technique displaces the vertices accordingly, but also linearly interpolates the intermediate points in a single, accelerated step. The GLSL vertex shader is detailed in Appendix C

The next stage of the image construction is the scaling and the translation of the image so that it will occupy the largest *to-scale* rectangle within the projection area on the projection surface. This *similarity* transformation, S, will fit the projected image to the largest possible rectangle, of the same aspect ratio, that can reside within the projection frustum from the perspective of the camera.

Figure 6.5 presents the prospective results of this step. The former displays a wireframe and plain image before a warp is applied while the latter displays the wireframe and plain image of the texture after it experiences a random displacement.

6.2.7 Geometric Calibration - Multi-Projectors

As in Chapter 5, the multi-projector component of this implementation was not carried out. Like before, and stated in Tardif *et al* [42], the exact process for a single projection calibration is repeated for each n projectors in the multi-projector system. The division of the image across n projectors is as stated in Chapter 5. If the system needs to be robust against occlusions, then each projector will cast redundant images. If a large scale display is required, the image is split across n framebuffer objects and projected by its respective projector.

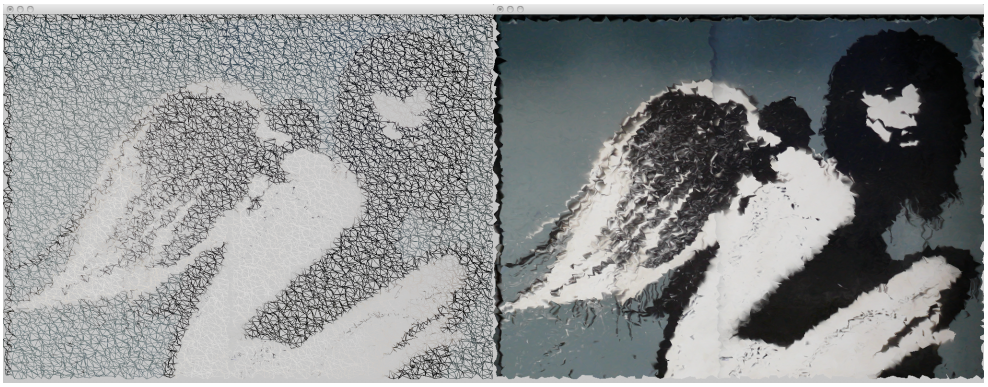
6.2.8 Completed to Date

Due to time constraints experienced, this component of the system was not fully implemented. This calibration system can successfully project, capture and threshold the difference images while the structure to pre-warp the image via the GPU is in place. The only element left to implement is the inverse mapping stage. The implementation presented above is at this stage theoretical and would be subject to change. Although the component was not fully implemented, the intermediary results displayed in Figures 6.3 and 6.4, as mentioned, reflect results shown in Tardif *et al* [42] and provide promising outlook for further development.

6.3 Summary

This Chapter presented the implementation for the geometrical calibration component of this dissertation. Gray-code structured light was opted for robust encoding of the camera image plane. The encoded pixels in the camera image plane correspond to the those in the projection image plane and a displacement map is created to warp the projection image to texture the camera image plane. While not entirely complete, intermediary results are hugely promising.

(a) No warp and wireframe (b) No warp image



(c) Warp on wireframe (d) Warped image

Figure 6.5: Random displacement map applied to vertices on plane

Chapter 7

System Evaluation

This Chapter will present the system's performance based on a number of specifically designed tests. The first performance based metric that must be assessed is the automatic keystoning calibration of the system, using a single projector and projecting onto a planar projection surface. This test will track an object as it moves horizontally across a screen. This will then be extended to test this automatic calibration performance across two projectors upon a planar surface.To gauge the effectiveness of the calibration, the results will be compared to those taken before the calibration step takes place. Metrics can also be compared to other implementations of projector calibration systems. To assess the Gray-code based geometric calibration component for non-planar, or free form surfaces, the same experiments can be carried out, being compared to pre-calibration results.

The performance of the system will also be measured. Benchmark testing will be utilised to measure the performance increase between the calibration being solely carried out on the central processing unit (CPU) versus the graphics processing unit (GPU).

7.1 System Hardware

This project was developed using two XGA (1024 x 768) DELL 2400MP (3000 ANSI Lumen) projection devices, UniBrain Fire-i firewire webcams and developed on an Apple MacBook Pro, running a 2.4GHz Intel Core 2 Duo with 4GB of RAM. The computer also has access to NVidia 9600M GT and NVidia 9400M graphics cards.

7.2 Single Projector - Planar Surface

7.2.1 Object Tracking Consistency

The object tracking test will aim to assess the correctness of the orientation and the consistency of the internal warping of a calibrated, pre-warped projection. This test will plot the trajectory of a ball as it moves

across the screen, calculating its centre point and storing the (u, v) coordinates as seen on the camera image plane. The line plot ought to be linear, as it is a planar surface, and the standard deviation from the v axis should be 0. If result is otherwise, it represents a warp of some description present in the projected image with respect to the camera.

For comparison, the ideal projection's object tracking plot will be displayed along with the recorded uncalibrated plot and the newly calibrated plot. This will give an immediate, observational feedback as to how the system performed. The standard deviation, $\sigma(v)$, of the line plots produced from this test will measure how often the plot strays along the v axis. If there is incorrect or inaccurate point interpolation within the image, the standard deviation will highlight this. In addition, the difference between the starting point and the finishing point, *Diff*(v), will be recorded as will the rate of the change (*RoC*) of the plot, giving a full overview of the plot's characteristics. The ideal projection will exhibit 0 on each of these metrics and it is the goal of the calibrated projection to return values that are as close to 0 as possible.

The hypothesis is that the uncalibrated projection will exhibit a warp directly related to the respective positioning of the camera, projector and projection surface. The standard deviation with regards the v coordinate in the image plane will be non-zero as will the rate of change. The calibrated projection should exhibit characteristics close to, or identical to, that of the ideal projection.

7.2.2 Object Tracking Consistency: Result

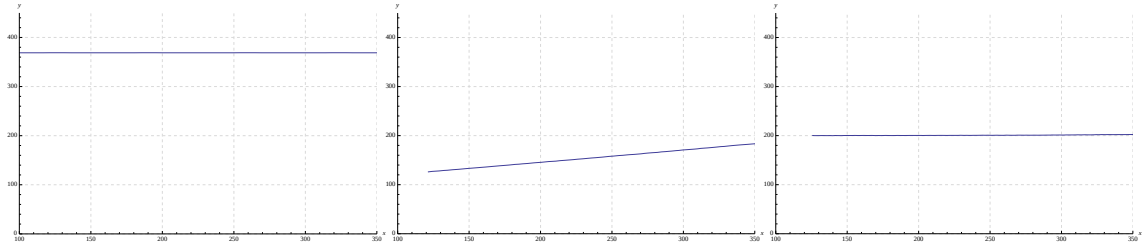
The resulting plots of the test are displayed in Figure 7.1. Here it is evident that the uncalibrated projection image, (b), exhibits a strong distortion with respect to the ideal projection, (a). The uncalibrated plot reveals a large standard deviation of 26.9455 and a rate of change of 0.25657. In comparison to this, the calibrated, pre-warped image, (c), displays far more satisfactory results. The standard deviation is negligible at 0.0212659 as too is the rate of change, 0.00965629, as hypothesised. These results are detailed in Table 7.1.

Status	**RoC**	**Diff** (v)	$\sigma(v)$
Ideal	*0.0*	*0.0*	*0.0*
Uncalibrated	0.25657	90.528	26.9455
Calibrated	0.00965629	2.481	0.0212659

Table 7.1: Object tracking results

7.3 Performance Benchmarks

The aim of the performance benchmarks are to compare the performance of the projection system running on the central processing unit (CPU) via OpenCV, against the system running on the graphics processing unit (GPU), using OpenGL. The metric in this case are the *frames per second* (FPS) rendered during runtime.



(a) Ideal Projection (b) Uncalibrated Projection (c) Calibrated Projection

Figure 7.1: Results for the *object tracking* test

The hypothesis is that there should be a noticeable increase in the FPS rate when the image or movie is warped using OpenGL rather than OpenCV. The setup of the test is as follows: The projector output *mirrors* the computer display, so the projector displays the exact content showing on the computer screen. During runtime, the current FPS rate, paired with the current elapsed time in seconds and milliseconds (*sec:millisec*), is printed to the console providing a very dense plot of the frame rate of the application. The output is plotted using Wolfram Mathematica 6.

For the CPU setup, the OpenCV `cvWarpPerspective()` function is called to apply the warp to an image. In this function, a source image and transformation matrix is passed along with specific flags. The transformation matrix is carried out upon the image, linearly interpolating points, and stored in a destination image parameter which is also passed. The transferring of data from one image type to another can expect to cause a lag in performance.

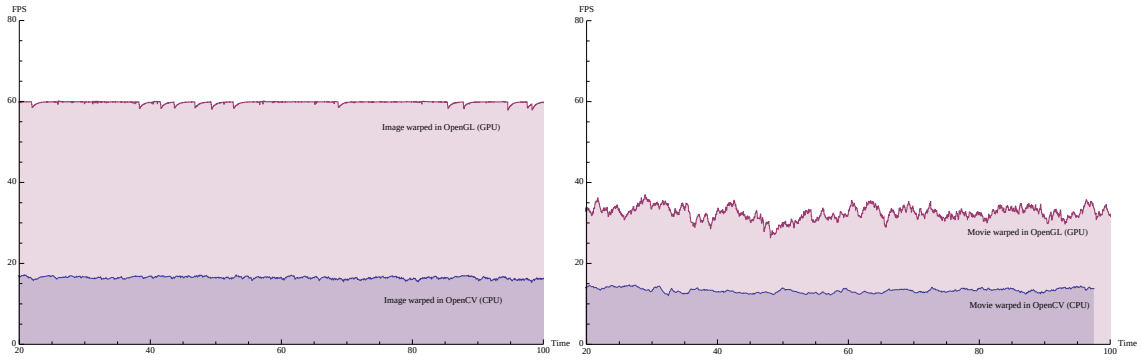
To measure the performance using OpenGL, the homography, which is a projective transform, is multiplied by the scene's projection matrix. The 3x3 homography transform must be converted from its *row-major* makeup to a 4x4 *column-major* matrix, as described in Chapter 5. This is subsequently multiplied with the scene's projection matrix to warp the OpenGL scene. As there is no swapping data from one image to another and the OpenGL pipeline is accelerated by the GPU, we can expect huge gains from this approach.

7.3.1 Rendering Image Files

The first test compares the rendering of a single image which undergoes a warp according to a calculated homography. The test is run for 100 seconds and the output plots from 20 seconds to 100 seconds. This test should perform at, or close to, interactive rates. As there is no image change or data transfers, the update overhead is negligible other than the warping function call.

7.3.2 Performance Benchmark: Result 1

As can be scene from Figure 7.2 (a), the image warped with OpenGL pipeline renders at real-time rates, seldom deviating from the 60 FPS rate we would hope for. This is a vast improvement over the OpenCV



(a) Rendering an image with OpenGL (red) versus OpenCV (blue)

(b) Rendering a video with OpenGL (red) versus OpenCV (blue)

Figure 7.2: Result for benchmark test 1(left) and 2 (right)

rendering method which averages at 16 FPS, barely a third of the rate of the OpenGL implementation.

7.3.3 Test 2: Rendering Movie Files

The second test will examine the system's performance while playing a video file. In this implementation the metric is still FPS rate over a period of 100 seconds. The graph plots the performance between 20 and 100 seconds. In this test we can expect a drop in the performance of each implementation. As the video file has to be updated at every frame, an overhead is created. Even though the video will run at 24FPS, it is updated at each frame to check if a new frame must be rendered. We still expect a superior performance running the video though the OpenGL pipeline than that of OpenCV. The result is present in Figure 7.2 (b).

7.3.4 Performance Benchmark: Result 2

As expected, a drop in performance is evident in both outputs. A large drop is evident in the OpenGL rendering while a smaller performance drop is noticeable in the OpenCV implementation. Regardless of the drop, the GPU accelerated version still renders at a rate comfortably higher than general video frame rate ($\sim$ 24FPS). A table displaying a comparison can be seen in Table 7.2.

Method	Media	Ave FPS
OpenCV (CPU)	Image	16.5119
OpenCV (CPU)	Video	14.0555
OpenGL (GPU)	Image	59.6745
OpenGL (GPU)	Video	32.9909

Table 7.2: OpenCV versus OpenGL benchmark

7.4 To Be Evaluated

7.4.1 Single Projector - Free-form Surface

As the Gray-code based geometric calibration step was not fully implemented, it was not possible to test the accuracy of this calibration step. Had it been implemented fully, the very same tests would be carried out to assess the overall calibration and the integrity of the linear interpolation of warped points within the image. However, much of the implementation is complete and though it can not be tested by these means presented, the intermediary results presented in Figure 6.3 and 6.4 display the exact characteristics as those found in [42], which is promising for further development.

7.4.2 Multi Projector Evaluation

As discussed in Chapters 5 and 6, the multi-projector component of this system was not implemented due to time constraints. However, the testing of the multi-projector component would be identical to that of the single projector tests.

Chapter 8

Conclusions and Future Work

This dissertation attempted to create a flexible framework that would calibrate a multi-projector system using a camera device. First, the camera device must under go lens distortion rectification so that the camera device does not contribute its own warp in the the calibration process. Zhang's camera calibration model is used to implement this step.

Keystoning, which occurs when a projector is not correctly aligned to the projection surface, is corrected by calculating the warp function a known image undergoes and applying the inverse transform to correct the image so that it is square and coherent to the viewer. This is done by using the projector-camera (PROCAM) model. The camera device is located at the audience's sweet-spot, and by rectifying the image within camera image space, the image will be rectified for the audience.

A more challenging task is to successfully calibrate the projection when projecting upon a non-planar, or free-form, surface. In this situation, the robust Gray-code structured light is used to encode the scene in the camera image plane. By analysing the correspondences, a displacement map is created between the projected image and camera image plane. Using this displacement map, the image can be pre-warped and linearly interpolated to essentially *texture* the camera image plane, rectifying the image from the viewers perspective. These steps have been further enhanced by utilising the GPU for real-time rendering performance.

While the implementation of the calibration framework is not complete, the preliminary results attained from tests are very promising. First, the warping rectification test shows, with negligible standard deviation ($\sigma(v)$ = 0.0212659), that the linear interpolation method used to rectify the perspective warp works very well. Though this warp was applied manually, it is expected that similar results will be attained by automatic calibration once the inverse homography error, encountered in Chapter 5, is resolved.

The use of OpenGL and the utilisation of the GPU to apply the warp transform, rather than the CPU, hugely improves the performance of the system. By using the GPU to render an image, the system runs at an interactive framerate of $\sim$60FPS, 3.614 times faster than using the CPU. Rendering video files, too, presents favourable results on the GPU, offering speeds 2.347 times faster than those achieved on the CPU. While the video file did not run at interactive rates, an average framerate of 32.9909FPS mean that the system can comfortably achieve video framerates of $\sim$24FPS.

8.1 Future Work

8.1.1 Multi-projector Calibration

Due to time constraints, the multi-projector component of this calibration system was not implemented. The respective multi-projector calibration development for planar surfaces and free-form surfaces has been discussed in Chapters 5 and 6.

8.1.2 Photometric Calibration

When projector frusta overlap, the intersecting areas on the projection screen are subject to higher colour intensity than those areas not intersecting. This creates artifacts that must be resolved to achieve a uniform projection. Additionally, if projecting upon a surface which does not have uniform texture or colour, it is also probable that inconsistencies, caused by the underlying colour or texture, will affect the final colour make-up of the final projection. These issues can be solved with photometric calibration.

Using a single camera device provides an ideal opportunity to assess the outcome of the projected images on the projection surface. The single camera is not subject to varying quantisation results, as would be expected if a number of camera devices were to be used, and so can be used to calculate a standard colour blending function to rectify the photometric artifacts of the projection.

There is extensive research done in this area. Bimber and Rakesh [5] use alpha blending on each projected image to create a uniform projection across a multi-projector system. Chen *et al* [11], Juang and Majumder [19] and Song and Cham [36] all offer separate solutions to the problem of accurate photometric calibration of a PROCAM system.

8.1.3 Framework Extension

Further to the above additions to the calibration system, the addition of a robust event-handling graphical user interface (GUI) would greatly aid the workflow of the system. *Qt* [26] is such an interface that would allow the operator greater control of the entire calibration system.

The implementation of the image processing state, presented in Chapter 3, would permit the operator to add different effects to the image or video currently playing. These effects could be created using GLSL where the GPU would be used to accelerate rendering times.

Appendix A

Camera Calibration Results

A.1 Enumeration of Camera Devices

```
1
2   int deviceCount = 0;
3   int availableDeviceCount = 0;
4
5   for(int i = 0 ; i < (*deviceList)->count ; ++i)
6   {
7     SGDeviceName nameRec;
8     nameRec = (*deviceList)->entry[i];
9     SGDeviceInputList deviceInputList = nameRec.inputs;
10
11    int numInputs = 0;
12    if( deviceInputList ) numInputs = ((*deviceInputList)->count);
13
14    if(nameRec.flags != sgDeviceNameFlagDeviceUnavailable)
15    { deviceCount++; availableDeviceCount++; }
16    else
17      deviceCount++;
18  }
```

A.2 Finding Chessboard Patterns

```
1  bool ofCvCameraCalibration::findCorners(const IplImage* bw, vector<ofxPoint2f>& points) const
2  {
3    IplImage* templmg = cvCreateImage( csize, IPL_DEPTH_8U, 1 );
4    CvPoint2D32f* p = new CvPoint2D32f[nCornersX*nCornersY];
5    int nCorners;
6
7    int bPerfect = cvFindChessBoardCornerGuesses( bw, templmg, NULL, cvSize(nCornersX, nCornersY),
         p, &nCorners );
8
9    cvFindCornerSubPix(bw, p, nCorners, cvSize(5,5), cvSize(-1,-1),
10   cvTermCriteria(CV_TERMCRIT_ITER, 100, 0.1) );
11
12   //convert back
13   for( int i=0; i<(nCornersX*nCornersY); ++i )
14     points.push_back( ofxPoint2f(p[i].x,p[i].y) );
15
16   cvReleaseImage( &templmg );
17   delete p;
18
19   if( !bPerfect ) {
20     cout << "Did not find expected number of points...\n";
21     return false;
22   } else {
23     return true;
24   }
25 }
```

A.3 Calibrate the Camera

```
1
2  void ofCvCameraCalibration::calibrateCamera(const int                    nImages,
3                        const vector<ofxPoint2f>&  _screenPoints,
4                        const vector<ofxPoint3f>&  _worldPoints,
5                        ofxVec4f&                  _distortionCoeffs,
6                        ofxMatrix3x3&              _camIntrinsics,
7                        vector<ofxVec3f>&          _transVectors,
8                        vector<ofxMatrix3x3>&      _rotMatrices
9                        ) const
10 {
11
12   //assume all points have been for all images
13   int* nCorners = new int[nImages];
14   for( int i=0; i<nImages; ++i )
15     nCorners[i] = nCornersX*nCornersY;
16
17   CvPoint2D32f* screenP = new CvPoint2D32f[nImages*nCornersX*nCornersY];
18
19   for( int i=0; i<nImages*nCornersX*nCornersY; ++i ){
20     screenP[i].x = _screenPoints[i].x;
21     screenP[i].y = _screenPoints[i].y;
22   }
23
24   CvPoint3D32f* worldP = new CvPoint3D32f[nImages*nCornersX*nCornersY];
25   for( int i=0; i<nImages*nCornersX*nCornersY; ++i ) {
26     worldP[i].x = _worldPoints[i].x;
27     worldP[i].y = _worldPoints[i].y;
28     worldP[i].z = _worldPoints[i].z;
29   }
30
31   CvVect32f  distortion = new float[4];
32   CvMatr32f  camera_matrix = new float[9];
33   CvVect32f  translation_vectors = new float[3*nImages];
34   CvMatr32f  rotation_matrices = new float[9*nImages];
35
36   cvCalibrateCamera( nImages,      //Number of the images.
37           nCorners,      //Array of the number of points in each image.
38           csize,       //Size of the image.
39           screenP,       //Pointer 2D points in screen space.
40           worldP,      //Pointer 3D points in real space
41           distortion,      //output: 4 distortion coefficients
42           camera_matrix,     //output: intrinsic camera matrix
43           translation_vectors,  //output: Array of translations vectors
44           rotation_matrices,  //output: Array of rotation matrices
45           0 );         //intrinsic guess needed
46
47   PrintIntrinsics( distortion, camera_matrix );
48
```

```
49    //convert coefficients
50    _distortionCoeffs.set( distortion[0], distortion[1], distortion[2], distortion[3] );
51
52    //convert camera matrix
53    for( int i=0; i<9; ++i)
54      _camIntrinsics[i] = camera_matrix[i];
55
56    //convert translation vectors
57    for( int ilmg=0; ilmg<nlmages; ++ilmg ) {
58          _transVectors.push_back( ofxVec3f(translation_vectors[ilmg*3],
59                        translation_vectors[ilmg*3+1],
60                        translation_vectors[ilmg*3+2]) );
61    }
62
63    //convert rotation matrices
64    for( int ilmg=0; ilmg<nlmages; ++ilmg )
65    {
66       _rotMatrices.push_back( ofxMatrix3x3(rotation_matrices[ilmg*9],
67                         rotation_matrices[ilmg*9+1],
68                         rotation_matrices[ilmg*9+2],
69                         rotation_matrices[ilmg*9+3],
70                         rotation_matrices[ilmg*9+4],
71                         rotation_matrices[ilmg*9+5],
72                         rotation_matrices[ilmg*9+6],
73                         rotation_matrices[ilmg*9+7],
74                         rotation_matrices[ilmg*9+8]) );
75    }
76
77    CvVect32f  translation_vectors2 = new float[3];
78    CvMatr32f  rotation_matrices2 = new float[9];
79
80    CvMat* matRotMat;
81    CvMat* matTranMat;
82
83    float* focalLength = new float[2];
84    focalLength[0] = camera_matrix[0];
85    focalLength[1] = camera_matrix[4];
86
87    CvPoint2D32f principalPoint;
88    principalPoint.x = camera_matrix[2];
89    principalPoint.y = camera_matrix[5];
90
91  }
```

A.4 Camera Calibration Screenshot

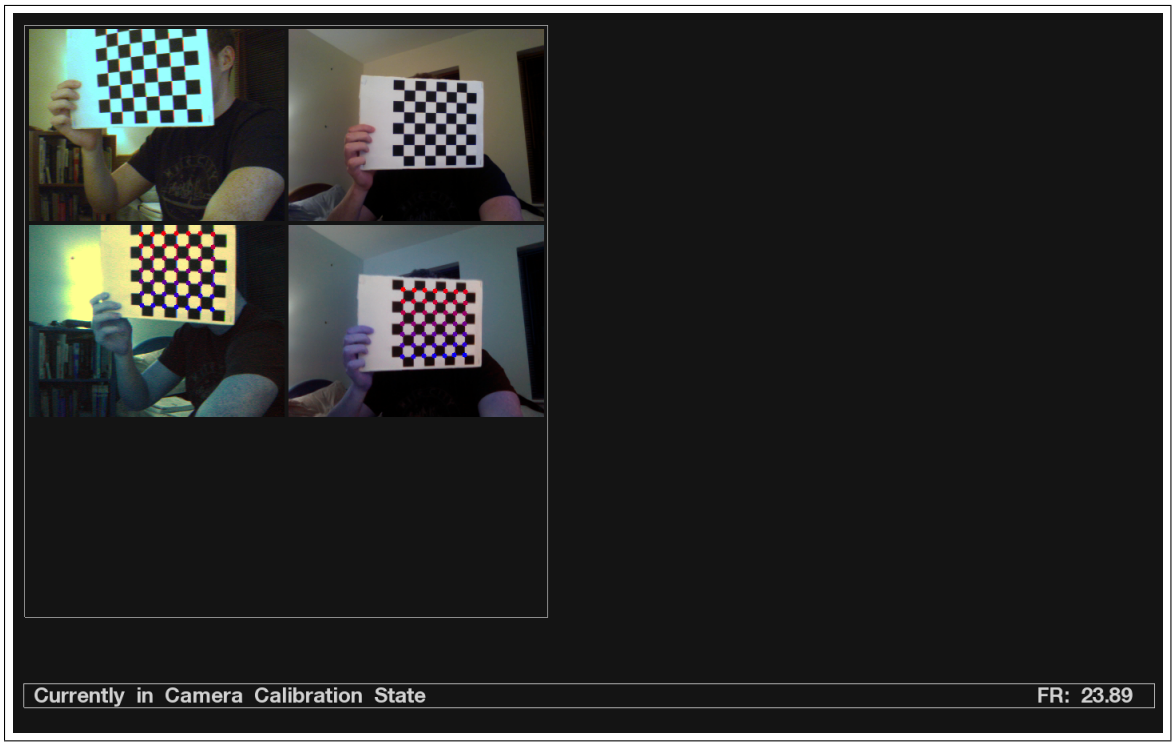


Figure A.1: Camera Calibration: Screenshot of camera calibration screen (number of camera devices = 2)

Appendix B

Projector-Camera Homography Issues

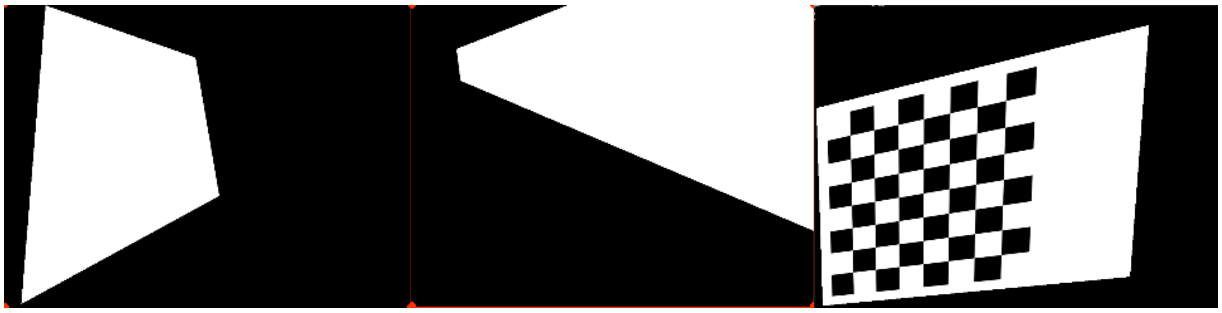
B.1 Getting Camera Projector Homography

```
1  void ofxCalibrationHandler::getCameraProjHomography()
2  {
3    cameraProjHomography = cvCreateMat(3, 3, CV_32F);
4
5    // Temp variables for
6    bwImg2   = cvCreateImage( csize, IPL_DEPTH_8U, 1 );
7    IplImage* tempImg = cvCreateImage( csize, IPL_DEPTH_8U, 1 );
8    cvCvtColor( tempHolder.getCvImage(), bwImg2, CV_RGB2GRAY );
9
10   CvPoint2D32f* p = new CvPoint2D32f[49];
11
12   CvMat* mSrcPoints = cvCreateMat(49, 2, CV_32F);
13   CvMat* mImgPoints = cvCreateMat(49, 2, CV_32F);
14   CvMat* inverse = cvCreateMat(3, 3, CV_32F);
15
16   CvPoint2D32f* srcPointsArr = new CvPoint2D32f[49];
17   CvPoint2D32f* srcPointsArrFour = new CvPoint2D32f[4];
18
19   for(int i=0; i<49; i++)
20   {
21     srcPointsArr[i].x = (int)(i mod 7)*20;
22     srcPointsArr[i].y = (int)(i/7)*20;
23   }
24
25   int nCorners = 0;
26   int found = cvFindChessBoardCornerGuesses(  bwImg2, tempImg, NULL, cvSize(7, 7), p, &nCorners
         );
27   if(!found) printf("Corners not found - \d\n",nCorners);
28
29   cvFindCornerSubPix(bwImg2, p, nCorners, cvSize(5,5), cvSize(-1,-1), cvTermCriteria(
         CV_TERMCRIT_ITER, 100, 0.1) );
```

```
30    cvSetData(mImgPoints, p, sizeof(CvPoint2D32f));
31    cvSetData(mSrcPoints, srcPointsArr, sizeof(CvPoint2D32f));
32    cvFindHomography(mSrcPoints, mImgPoints, cameraProjHomography, CV_RANSAC, 1.0, NULL);
33    cvInvert(cameraProjHomography, inverse, CV_LU);
34
35    cvReleaseImage(&tempImg);
36    cvReleaseMat(&mSrcPoints);
37    cvReleaseMat(&mImgPoints);
38
39    projectorCalib.projectionHomography = cameraProjHomography;
40 }
```

B.2 Erroneous Results

In Chapter 5 it was stated that problems calculating the correct homography function, and inverse homography function, were encountered. While the homography function seemed to correct the image within the camera image plane (Figure 5.4 (b)), when applied to the projection image, by visual inspection the results were incorrect. Figure B.1 (a) and (b) display a polygon (1024 x 768) undergoing the same homography transform that corrected the image in Figure 5.4 (b). As can be seen from the results below, the images are clearly incorrect with regards the correct pre-warped image B.1 (c).



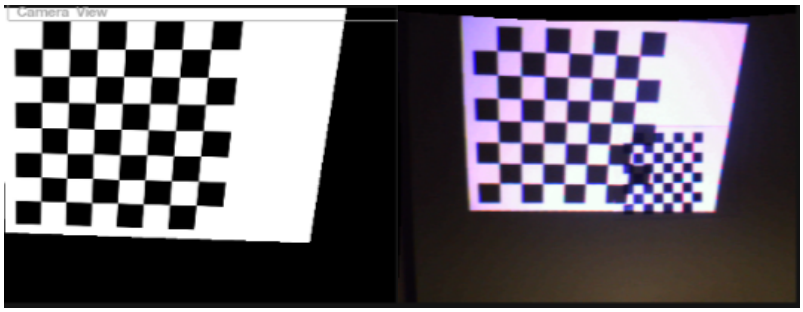
(a) Homography reapplied (b) Inverse homography applied (c) Correct Warp

Figure B.1: Incorrect results from seemingly correct transform matrix

After testing the system in both Matlab and Wolfram Mathematica 6, it is clear that both 2-dimensional and 3-dimensional transforms can be rectified by applying the inverse of the transform (Figure 5.3). This may suggest that the initial theory of a direct keystone rectification by applying the inverse transform to pre-warp the projected image is incorrect. Figure B.2 displays the most hopeful result of any tests carried out. This replicates the function used in Sukthankar *et al*'s [39], whereby:

$$pHs = cHs^{-1}.cHp$$

However, this dissertation is yet to mathematically prove that this function is correct, and therefore



(a) Camera-screen homography used (b) From camera's perspective

Figure B.2: Better results using Sukthankar *et al*'s [39] implementation

cannot be assumed to be the solution to this problem. It must be repeated at this stage that Sukthankar *et al* have a different implementation to this dissertation.

It was because of this issue with the system that consequent components of the system were not satisfactorily implemented. To satisfy this calibration problem temporarily, a manual warping function was created which is discussed next.

B.3 Manual Warping for Calibration

To continue with the development and to carry out tests that would gauge the system's performance, a function to allow the manual warping of an image was created. As only four points are necessary to compute a homography, four control points, one located at each of the image's four corners, are used to manually warp the image to be projected. Using the camera device to assess the position and orientation of the projected image, the image can be pre-warped to appear coherent to the camera device, and thus the audience. The user simply clicks and drags on the four controls points to actively warp the image (Figure B.3). The homography transform gets calculated every frame and applies the 3x3 transform to pre-warp the image. This process is carried out using the OpenGL implementation for speed and efficiency.

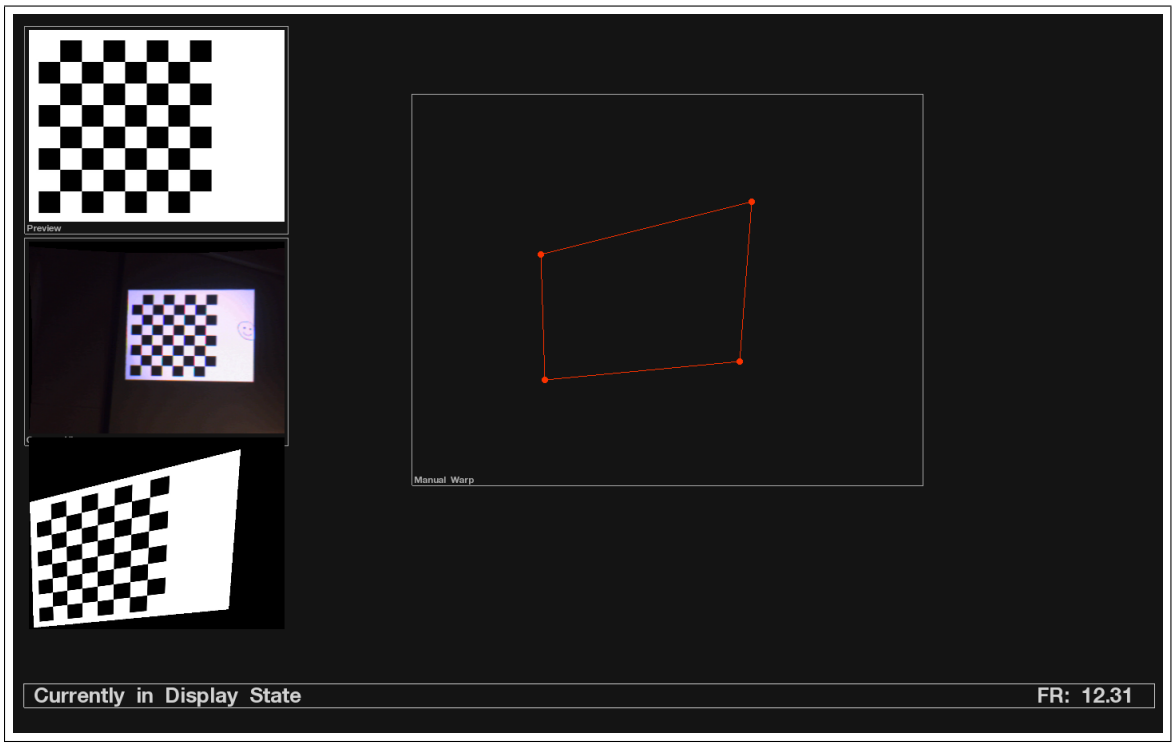


Figure B.3: Manual Warp Interface

Appendix C

Structured Light

C.1 GLSL Vertex Shader - (Displacement Map)

```
1  uniform sampler2D disMap;
2
3  void main()
4  {
5    vec4 pos;
6    vec4 newPos;
7
8    pos = gl_ModelViewProjectionMatrix * gl_Vertex;
9
10   float dX = texture2DLod( disMap, gl_MultiTexCoord0.xy, 0.0 ).x;
11   float dY = texture2DLod( disMap, gl_MultiTexCoord0.xy, 0.0 ).y;
12
13   dX *= 1024.0;
14   dY *= 768.0;
15
16   gl_FrontColor = gl_Color;
17   gl_TexCoord[0] = gl_MultiTexCoord0;
18   gl_Position = vec4(pos.x+dX, pos.y+dY, pos.zw);
19
20   //gl_Position = ftransform();
21 }
```

Bibliography

[1] D. Akca, F. Remondino, D. Novák, T. Hanusch, G. Schrotter, and A. Gruen. Performance evaluation of a coded structured light system for cultural heritage applications. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 6491 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, jan 2007.

[2] M. Ashdown, M. Flagg, R. Sukthankar, and J.M. Rehg. A flexible projector-camera system for multi-planar displays. In *CVPR04*, pages II: 165–172, 2004.

[3] J. Batlle, E. Mouaddib, and J. Salvi. Recent progress in coded structured light as a technique to solve the correspondence problem: A survey. *PR*, 31(7):963–982, July 1998.

[4] Oliver Bimber, Anselm Grundhöfer, Stefanie Zollmann, and Daniel Kolster. Digital Illumination for Augmented Studios. *Journal of Virtual Reality and Broadcasting*, 3(8), December 2006. urn:nbn:de:0009-6-6302, ISSN 1860-2037.

[5] Oliver Bimber and Ramesh Raskar. *Spatial Augmented Reality: Merging Real and Virtual Worlds*. A K Peters, Ltd., July 2005.

[6] BMW. Bmw augmented reality. Accessed 12 August 2009 at: http://www.bmw.com/, 2009.

[7] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly, Cambridge, MA, 2008.

[8] Jonathan Carr. Atlas editorials. Accessed 2 September 2009 at: http://www.atlaseditorials.com/2008/09/12/back-to-the-future-part-ii-predictions/, 2008.

[9] Li-Wei Chan, Wei-Shian Ye, Shou-Chun Liao, Yu-Pao Tsai, Jane Yung jen Hsu, and Yi-Ping Hung. A flexible display by integrating a wall-size display and steerable projectors. In *UIC*, pages 21–31, 2006.

[10] Han Chen, Rahul Sukthankar, Grant Wallace, and Kai Li. Scalable alignment of large-format multi-projector displays using camera homography trees. In *Proceedings of IEEE Visualization*, pages 339–346, 2002.

[11] Xinli Chen, Xubo Yang, Shuangjiu Xiao, and Meng Li. Color mixing property of a projector-camera system. In *PROCAMS '08: Proceedings of the 5th ACM/IEEE International Workshop on Projector camera systems*, pages 1–6, New York, NY, USA, 2008. ACM.

[12] Daniel Cotting and Markus Gross. Interactive environment-aware display bubbles. In *UIST '06: Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 245–254, New York, NY, USA, 2006. ACM.

[13] MIT Eyestop. Senseable city. Accessed 12 August 2009 at: http://senseable.mit.edu/eyestop/, 2009.

[14] M. Fiala. Automatic projector calibration using self-identifying patterns. In *ProjCam05*, pages III: 113–113, 2005.

[15] Christian Frueh and Avideh Zakhor. Capturing 2.5d depth and texture of time-varying scenes using structured infrared light. *3D Digital Imaging and Modeling, International Conference on*, 0:318–325, 2005.

[16] R. Furukawa and H. Kawasaki. Dense 3d reconstruction with an uncalibrated stereo system using coded structured light. In *ProjCam05*, pages III: 107–107, 2005.

[17] Willow Garage. Opencv 1.1 c reference. Accessed 2 September 2009 at: http://opencv.willowgarage.com/documentation/index.html, 2009.

[18] Jens Gühring. Dense 3-d surface acquisition by structured light using off-the-shelf components. In *Proc. Videometrics and Optical Methods for 3D Shape Measurement*, pages 220–231, 2001.

[19] Ray Juang and Aditi Majumder. Photometric self-calibration of a projector-camera system. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 0:1–8, 2007.

[20] M. Kimura, M. Mochimaru, and T. Kanade. Projector calibration using arbitrary planes and calibrated camera. In *PROCAMS07*, pages 1–2, 2007.

[21] Daisuke Kondo, Yuichi Shiwaku, and Ryugo Kijima. Free form projection display and application. In *PROCAMS '08: Proceedings of the 5th ACM/IEEE International Workshop on Projector camera systems*, pages 1–2, New York, NY, USA, 2008. ACM.

[22] Johnny C. Lee, Paul H. Dietz, Dan Maynes-Aminzade, Ramesh Raskar, and Scott E. Hudson. Automatic projector calibration with embedded light sensors. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 123–126, New York, NY, USA, 2004. ACM.

[23] Johnny Chung Lee. Wii remote projects. Accessed 12 August 2009 at: http://johnnylee.net/projects/wii/, 2009.

[24] Zach Lieberman and Theodore Watson. Openframeworks. Accessed 10 August 2009 at: http://www.openframeworks.cc/about.

[25] Mesmer. Video and projection design and implementation. Accessed 2 September 2009 at: http://www.mesmer.co.uk/, 2009.

[26] Nokia. Products: Qt - a cross-platform application and ui framework. Accessed 6 September 2009 at: http://qt.nokia.com/products/, 2009.

[27] University of Sheffield. Gray-code to binary conversion. Accessed 2 September 2009 at: http://www.shef.ac.uk/physics/teaching/phy107/solvegray.html, 2009.

[28] T. Okatani and K. Deguchi. Autocalibration of a projector-camera system. *PAMI*, 27(12):1845–1855, December 2005.

[29] Jordi Pagès, Joaquim Salvi, Rafael García, and Carles Matabosch. Overview of coded light projection techniques for automatic 3d profiling. In *ICRA*, pages 133–138, 2003.

[30] MIT Media Lab Pranav Mistry. 6th sense. Accessed 1 August 2009 at: http://www.pranavmistry.com/projects/sixthsense/, 2009.

[31] Ramesh Raskar, Michael Brown, Ruigang Yang, Wei chao Chen, Greg Welch, Herman Towles, Brent Seales, Henry Fuchs, and Greg Welch Herman Towles. Multi-projector displays using camera-based registration, 1999.

[32] Reuters. Digital projection international and cbs outdoor partner for cross track projection on the london underground. Accessed 2 September 2009 at: http://www.reuters.com/article/pressRelease/idUS190805+03-Mar-2008+MW20080303, Mar 2008.

[33] C. Rocchini, P. Cignoni, C. Montani, P. Pingi, and R. Scopigno. A low cost 3d scanner based on structured light, 2001.

[34] MIT SENSEable City Lab. Senseable city. Accessed 12 August 2009 at: http://senseable.mit.edu/, 2009.

[35] Tom Simonite. Innovation: When advertising meets surveillance. Accessed 2 September 2009 at: http://www.newscientist.com/article/dn17424-innovation-when-advertising-meets-surveillance.html, July 2009.

[36] Peng Song and Tat-Jen Cham. A theory for photometric self-calibration of multiple overlapping projectors and cameras. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Workshops*, page 97, Washington, DC, USA, 2005. IEEE Computer Society.

[37] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing: Analysis and Machine Vision*. CL-Engineering; 2 edition, September 1998.

[38] Dimensional Studios. Musion eyeliner. Accessed 1 August 2009 at: http://www.eyeliner3d.com/, 2009.

[39] R. Sukthankar, R.G. Stockton, and M.D. Mullin. Smarter presentations: Exploiting homography in camera-projector systems. In *ICCV01*, pages I: 247–253, 2001.

[40] Rahul Sukthankar, Robert G. Stockton, and Matthew D. Mullin. Automatic keystone correction for camera-assisted presentation interfaces. In *IN THE PROC. OF INTERNATIONAL CONFERENCE ON MULTIMODAL INTERFACES*, pages 607–614, 2000.

[41] Jay W. Summet, Matthew Flagg, James M. Rehg, Gregory D. Abowd, and Neil Weston. Gvu-procams: enabling novel projected interfaces. In *MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia*, pages 141–144, New York, NY, USA, 2006. ACM.

[42] J. Tardif, S. Roy, and M. Trudeau. Multi-projectors for arbitrary surfaces without explicit calibration nor reconstruction. In *3DIM03*, pages 217–224, 2003.

[43] J. P. Tardif, S. Roy, and J. Meunier. Projector-based augmented reality in surgery without calibration. In *Engineering in Medicine and Biology Society, 2003. Proceedings of the 25th Annual International Conference of the IEEE*, volume 1, pages 548–551 Vol.1, 2003.

[44] Jean-Philippe Tardif and Sebastien Roy. A mrf formulation for coded structured light. In *3DIM '05: Proceedings of the Fifth International Conference on 3-D Digital Imaging and Modeling*, pages 22–29, Washington, DC, USA, 2005. IEEE Computer Society.

[45] Pablo Valbuena. Official website. Accessed 2 September 2009 at: http://www.pablovalbuena.com/, 2009.

[46] Vizoo. Advanced video design. Accessed 12 August 2009 at: http://www.vizoo.com/, 2009.

[47] Eric W. Weisstein. "gray code." from mathworld–a wolfram web resource. Accessed 12 August 2009 at: http://mathworld.wolfram.com/GrayCode.html, 2009.

[48] Li Zhang, Brian Curless, and Steven M. Seitz. Rapid shape acquisition using color structured light and multi-pass dynamic programming. In *In The 1st IEEE International Symposium on 3D Data Processing, Visualization, and Transmission*, pages 24–36, 2002.

[49] Zhengyou Zhang and Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:1330–1334, 1998.

[50] Jin Zhou, Liang Wang, Amir Akbarzadeh, and Ruigang Yang. Multi-projector display with continuous self-calibration. In *PROCAMS '08: Proceedings of the 5th ACM/IEEE International Workshop on Projector camera systems*, pages 1–7, New York, NY, USA, 2008. ACM.