Bard College
Bard Digital Commons

Senior Projects Spring 2019 | Bard Undergraduate Senior Projects

Spring 2019

Geometric Correction for a Spherical mirror projection on a Nonplanar Surface

Methuen J. Bell-Isaac
Bard College

Follow this and additional works at: https://digitalcommons.bard.edu/senproj_s2019

Part of the Other Computer Engineering Commons

This work is licensed under a Creative Commons Attribution-Noncommercial-No Derivative Works 4.0 License.

Recommended Citation

Bell-Isaac, Methuen J., "Geometric Correction for a Spherical mirror projection on a Nonplanar Surface" (2019). *Senior Projects Spring 2019*. 106.
https://digitalcommons.bard.edu/senproj_s2019/106

This Open Access work is protected by copyright and/or related rights. It has been provided to you by Bard College's Stevenson Library with permission from the rights-holder(s). You are free to use this work in any way that is permitted by the copyright and related rights. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself. For more information, please contact digitalcommons@bard.edu.

Geometric Correction for a Spherical Mirror Projection on a Nonplanar Surface

A Senior Project submitted to
The Division of Science, Mathematics, and Computing
of
Bard College

by
Methuen Jelani Bell-Isaac

Annandale-on-Hudson, New York
May, 2019

Abstract

This paper discusses an approach for removing distortion from an image projected on a non-planar surface. With a single projector setup in a spherical mirror projection system, it becomes possible to preserve image features. The approach takes advantage of the configuration of the surface, specifically, the geodesic dome in this project. The configuration acts as a mold so that a warp mesh can be designed to match the surface configuration. Points in an image are then mapped to their corresponding point on the destination multi-planar surface represented by the mesh. The removal of distortion brings us a step closer to automating processes like this and paves a way for experimenting with applications in an immersive environment.

Contents

Abstract **iii**

Dedication **vii**

Acknowledgments **ix**

1 Introduction **1**
1.1 Background . . . 1
1.2 Motivation . . . 2
1.3 Past Work . . . 3
1.4 Setup . . . 4

2 Computing Image Transformations **7**
2.1 Geometric Transformations . . . 7
2.1.1 Translations . . . 7
2.1.2 Rotations . . . 9
2.2 Homography . . . 9
2.3 Affine Transformations . . . 11

3 Mesh Construction and Data Handling **13**
3.1 Software . . . 13
3.1.1 OpenCV for Python . . . 13
3.1.2 Numpy . . . 14
3.1.3 Processing (Java) . . . 14
3.1.4 JavaScript Object Notation . . . 14
3.2 Construction Process . . . 15
3.2.1 Node Class . . . 15
3.2.2 Placing Nodes . . . 16
3.2.3 Storing Node and Triangle Information . . . 16
3.2.4 Manual Distortion Removal . . . 17

4 Mesh Warping **21**
4.1 Selecting a Test Image 21
4.2 Data Preparation 22
4.3 The Masked Image 23
4.3.1 Region of Image 23
4.3.2 Masking 24
4.4 Warp and Reconstruction 24
4.4.1 Piecewise Affine Transformations 25
4.4.2 Combining the Images 26

5 Results **29**
5.1 Output Image 29
5.2 Analysis of Geometric Correction 30

6 Conclusion and Future Work **37**

Appendices **39**

A Python and Processing **39**
A.1 Execute_warp.py 39
A.2 dome_mesh.pde 42

Bibliography **49**

Dedication

Dedicated to my mother.

Acknowledgments

I want to thank my friends and family who have supported me through the years.

I want to thank my best friend Rachael Fritz for her unwavering support and encouragement.

I finally want to thank my advisor Keith O'Hara. He has been an inspiration to me during my time at Bard College. His consistent devotion to helping his students amazes me. He has done more than advise me in my academic journey, whether that be giving encouragement when I was struggling, or challenging me to optimize my potential.

1
Introduction

1.1 Background

The human brain's ability to analyze our environment through our eye, is an unbelievable feat. Our brain is able to receive input from what our eyes see and makes conclusions in nanoseconds. Computer Vision is the field of study that attempts to mimic the human brain's natural ability of vision. Our brains are excellent tools for feature detection and research in computer vision has successfully allowed computers to replicate many of the same actions. 3D modeling, face detection and motion capture are just a few applications of computer vision and emphasize the possibilities that are present because of computer vision applications.[2]

Augmented reality has become an emerging technology that has several practical uses. For example, the medical industry has seen a rise in AR interest due to the ability for this technology to enhance medical training. AR has provided the ability to interact with virtual objects through the use of cameras and screens. One method that has seen a lot of application at planetariums is a projector-camera system. These systems in planetariums operate using multiple projections and a costly setup. When planetariums perform demos, the use of multiple projectors allows them to correctly calibrate their cameras with the nonplanar surface in view, by using a collection of

planar projections to cover the nonplanar surface. This is quite trivial on a smaller scale and expensive when attempting to use multiple projectors.

However, to reduce cost and increase feasibility of setting up projector-camera systems in casual spaces, it is effective to design this setup using a single projector. This type of system can allow for one to interact with a projected screen based on the input from what the camera is observing. The projection of a screen on a planar surface is required to begin this process, and there are multiple examples of this system functioning correctly when the projection is on a planar object in real space. This is due to the lack of distortion that needs to be accounted for since the projection is planar and the surface it is projected on is also planar.

1.2 Motivation

The one who provided the geodesic dome is electronic arts professor Ben Coonley. This was initially built for an exhibit in the exhibition *Dreamlands: Cinema and Art 1905-2016 at the Whitney Museum of American Art*. He hopes the dome can be used as a tool to explore different things that could be done in the visual arts using a nonplanar surface, such as this dome. The Bard College computer science department took it upon themselves to find a way to configure a projector-camera system that could utilize the dome to create an immersive environment. An immersive environment setup with a projector-camera system could potentially be useful in running applications that can be interacted with without the use of any headset or other peripheral. However, this requires that the camera be calibrated appropriately and a method must be found that can automatically find point correspondences between what the camera sees and the actual projection. With much work done on projector-camera systems on planar surfaces, executing the same setup for a nonplanar surface is still a subject that requires more exploration.

1.3 Past Work

This project is the third edition in a line of two previous senior projects written by Kai Malowany and Darren Tirto.

Malowany's project *Beyond Homographies: Exploration and Analysis of Image Warping for Projection in a Dome* is focused on discussing multiple approaches for image warping, specifically for a geodesic dome. The concepts in this work are useful in this project for expanding upon the methods that have been explored and directly relate to the image warping method discussed in Section 4.4.1.

Tirto's project *Projector-Camera Calibration for a Multi-Planar Dome* discusses methods of removing distortions from a projection on a dome through projector-camera calibration. Tirto used ray-plane intersection to find the correspondences between the points in the physical space and camera view. Though Tirto was able to estimate a projector matrix, he was not satisfied with the accuracy and noted that he was constrained by the fact that he was not able to use more planes than the camera could image to accurately calibrate the entire space. Tirto's discussion of how this could be improved using the physical configuration of the dome inspires the methods in Section 3.2.

Paul Bourke, a researcher that has dedicated a significant amount of time to the study of using a spherical mirror for projection, has suggested that a mesh be used to overlay an image to create a homography utilizing the points of the mesh.[3] Paul Bourkes research has found to be extremely helpful in understanding the multiple approaches to projection in a dome. Bourkes papers have given insight into single projector systems and have set a foundation for the continuation of this project.

1.4 Setup

The environmental setup to house the spherical mirror projection is a geodesic dome made of wood and cardboard. The dome is constructed from 40 triangles, fifteen equilateral triangles with the dimensions 35” x 35” x 35” and 25 isosceles triangles with the dimensions, 31” x 31” x 35”. The projector being used is an Infocus IN3136a 720p projector. The projector is oriented to project off a planar mirror and then onto a spherical mirror to cover the surface area of the dome.

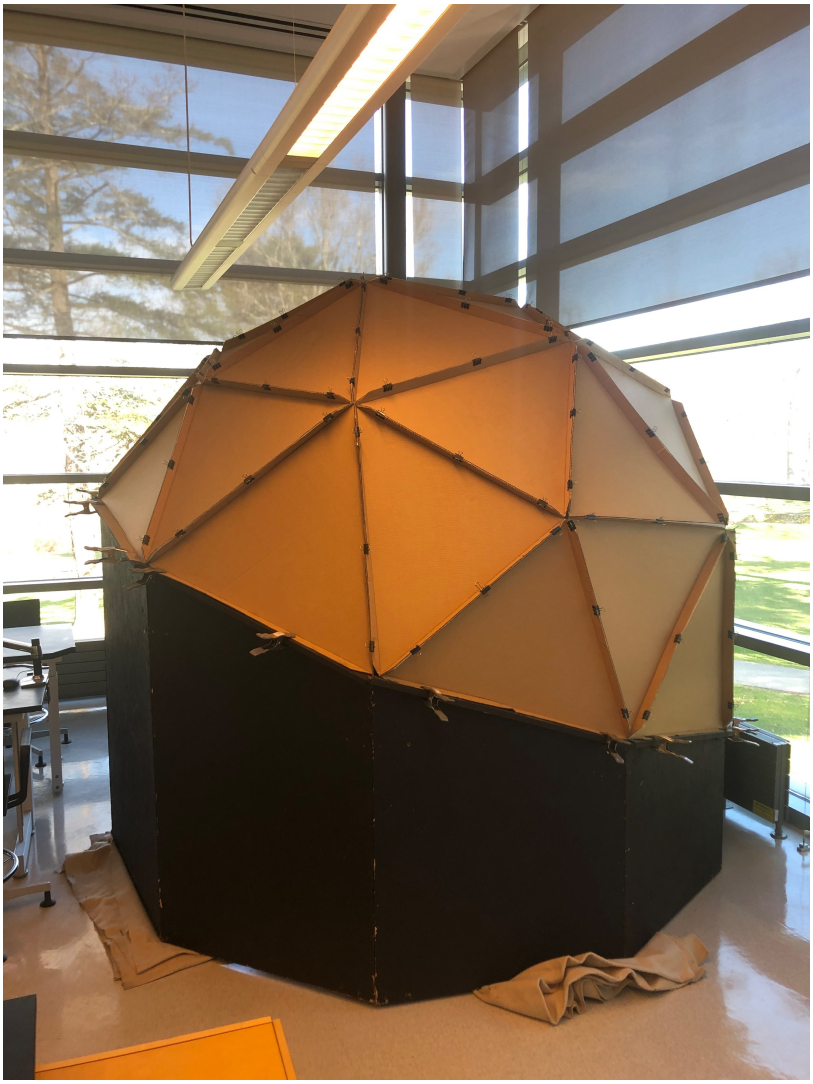


Figure 1.4.1. Geodesic dome exterior

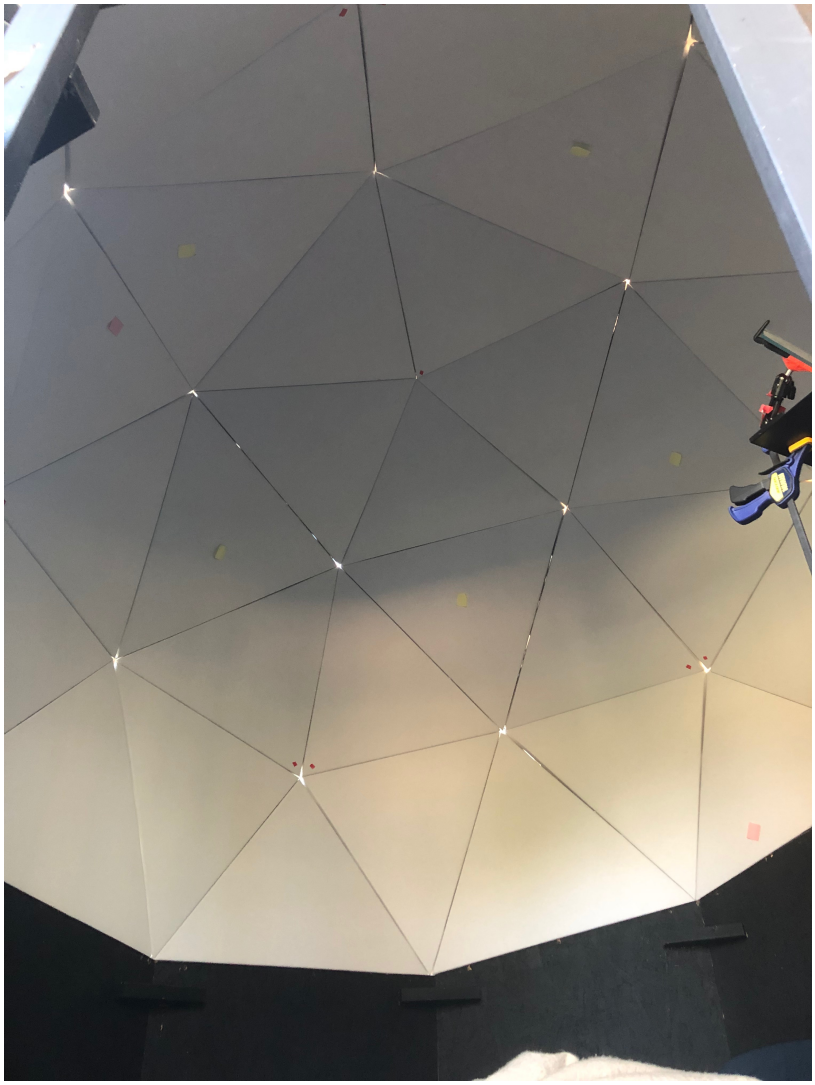
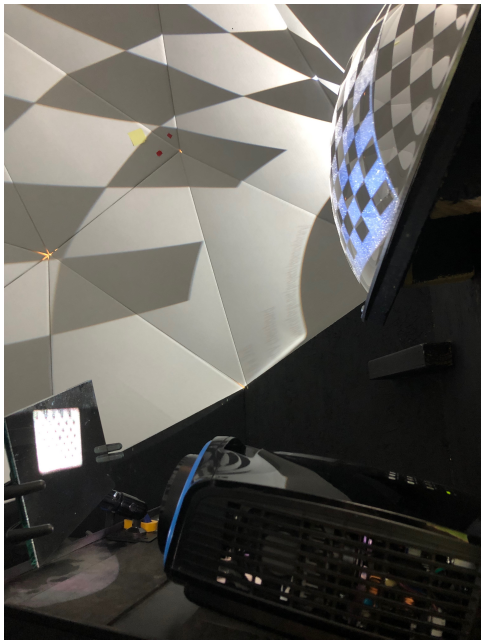


Figure 1.4.2. Geodesic dome interior.

(a) The reflection off the spherical mirror distributes the projection so that it covers a larger surface area of the dome. Consequently this adds distortion to the image.



(b) Note how the projection reflects off the planar mirror and onto the spherical mirror.

Figure 1.4.3. The interior setup of the spherical mirror and the Infocus IN3136a 720p projector.

2
Computing Image Transformations

2.1 Geometric Transformations

Image processing consist of many elements, and geometric transformations are one of the most essential. The positions of pixels in an image can be modified using image transformations, outputting new images that reveal a visual representation of a specific transformation. This section will discuss the details of some different geometric transformations and how they are applied to the individual points in an image.

Before we continue with this section, it is important to note that the vectors that are shown in the following section will be a homogeneous representation of coordinates. Homogeneous coordinates introduces a third dimension that is not normally present so that lines and planes can be represented at infinity.

For example, if a coordinate is normally denoted by (x, y) its homogeneous coordinate would be (x, y, w) where $w \neq 0$. Homogeneous representation helps perform projective transformations using matrices.

2.1.1 Translations

Translation is a geometric transformation that uses addition to change the position of the specified point.

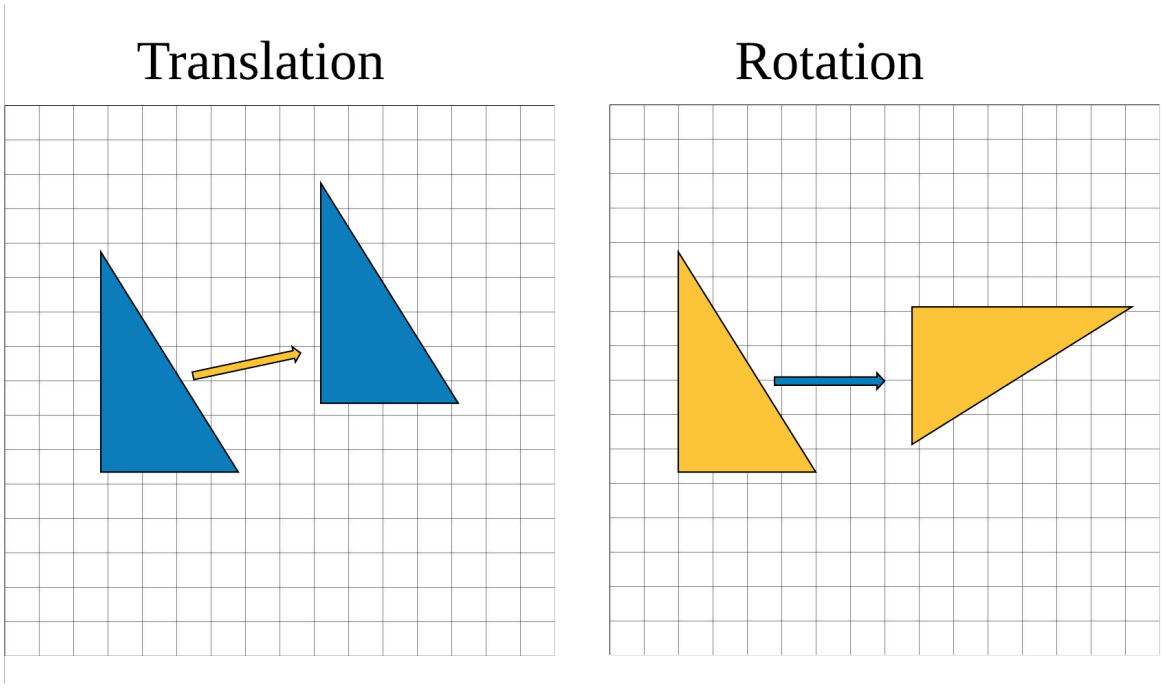


Figure 2.1.1. A simple representation of two image transformations on a right triangle. The translation example has shifted the triangle by some factor after applying a translation vector to the triangles vertices. The rotation example shows a 90 degree rotation.

Let vector $\mathbf{u}$ be defined by

$$\mathbf{u} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Let the translation vector $\mathbf{t}$ be defined by

$$\mathbf{t} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

If the translation is applied to the vector $\mathbf{u}$, the output vector $\mathbf{v}$ will be

$$v = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

representing a new position in 2D space. The identity matrix coupled with the translation vector produces a 3x3 matrix that can now use matrix multiplication to apply the translation transformation. With the help of translation all the pixels in an image can be shifted the same amount when the same translation vector is applied to each pixel in the image.

2.1.2 Rotations

Rotation is another transformation defined by an angle θ which is then applied to a vector.

Let vector $\mathbf{u}$ be defined by

$$\mathbf{u} = \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

and output vector $\mathbf{v}$ be defined by

$$\mathbf{v} = \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

where

$$x_2 = x_1 \cos(\theta) y_1 \sin(\theta)$$

and

$$y_2 = x_1 \sin(\theta) + y_1 \cos(\theta)$$

.

In matrix format, this rotation can be represented by a 2 x 2 unitary matrix.

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

When the rotation is applied to u, we get the following matrix representation of this transformation.

$$\mathbf{v} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} u$$

2.2 Homography

For the purpose of truly understanding how image transformation functions, it is important to grasp the idea of homography. Homography is a 2D projective transformation that maps points in one plane to another. [1]

Homography is useful for executing several different processes including rectifying, registering, and warping images. These processes are examples of taking points from one plane and mapping them to a set of points on another plane. Assume there is some image **M** of a piece of paper on a table and another image **N** that has that same piece of paper in it but the image is from a different perspective than image M. A homography transform will be required to display an image where the paper in each image are aligned with each other and seem as though they were captured from the same perspective.

Let x_1, y_1 represent a point on the plane (piece of paper) in image M and let x_2, y_2 represent the same corresponding point on the paper, but in image N. To find the homography matrix that every point can be applied to, this equation is utilized shown using the euclidean system.

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = H \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

By finding H every point on the plane in image N can map to its appropriate point in image M and correctly align the images. This method of calculating H is called Direct Linear Transformation[1] and requires four or more points to find H due to the 8 degrees of freedom.

$$\mathbf{H} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}$$

Executing the equation above using homogeneous coordinates is shown below.

$$\begin{bmatrix} x_2 \\ y_2 \\ w_2 \end{bmatrix} = H \begin{bmatrix} x_1 \\ y_1 \\ w_1 \end{bmatrix}$$

This results in the following values for x_2, y_2 and w_2.

$$x_2 = x_1 h_1 + y_1 h_2 + w_1 h_3$$

$$y_2 = x_1 h_4 + y_1 h_5 + w_1 h_6$$

$$w_2 = x_1 h_7 + y_1 h_8 + w_1 h_9$$

2.3 Affine Transformations

An affine transformation is the process of executing a matrix multiplication and then a vector addition. In other words, it is a linear transformation followed by a translation. This allows for the expression of rotations, translations and scale operations, all using the affine transformation. Essentially these affine transformations represent relations between images and the transform is represented by a 2×3 matrix. An example transformation is shown below.

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & \Delta x \\ a_{10} & a_{11} & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

The output vector on the left-hand side of the equation represents an affine transformation from $\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$ to $\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$.

3
Mesh Construction and Data Handling

This chapter will focus on the process of developing the mesh and preparing it for warping. A mesh is an object made up of a network. Understanding how the mesh is rendered and used is crucial to comprehending image reconstruction and the affine transformations that will be discussed in later sections.

3.1 Software

To allow for experimentation and the opportunity for the field of computer vision to continue to grow, open source software becomes a great asset for the public. This section briefly discusses the software utilized in this project and how significant the functions involved are to the success of this project.

3.1.1 OpenCV for Python

OpenCV is a library that provides open source tools for computer vision applications. In 1999, Gary Bradsky started OpenCV at Intel and was later joined by Vadim Pisarevsky. The two managed the OpenCV team at Intel and successfully developed a library that supports multiple computer vision algorithms for a numerous number of problems. [6]

OpenCV-Python is the library specifically built to support the Python programming language. The Python implementation is ideal for the purpose of this project because of its readability and lack of complexity.

3.1.2 Numpy

The Numpy Python library is a package that is extremely useful when working with matrices, images and and any subject related to scientific computing. The Numpy array is the most useful object in the library and is the primary tool utilized in this project. The Numpy array object will help simplify performing several operations such as matrix multiplication and solving equation systems. These operations will be crucial for warping images by applying affine transformations.

3.1.3 Processing (Java)

Processing is a programming language that provides functions and types which have several practical uses relating to visual arts. The documentation is broad and when utilized correctly, it is quite simple to design and develop projects with different kinds of geometrical shapes, with various potential functionality. Processing is based in Java and so variables and data structures are declared the same way. With the goal of designing a mesh with movable nodes that should also have their x and y values stored for later use, processing was clearly the best choice for being able to efficiently build the mesh. Processing can also use physics to simulate real world environments or even develop complex games.

3.1.4 JavaScript Object Notation

JavaScript object Notation or JSON has been around since 1996 but did not become very popular till around 2001. Douglas Crockford is credited with the discovery of JSON format and JSON has since trumped XML as the Gold Standard for transporting data. The ability for JSON to completely surpass other formats is because of how easy it is to learn to use as well as visually pleasing and simple to write. It is for these reasons that we decide to use JSON objects to record and utilize the data needed to successfully perform multiple affine transformations.An example is shown in Figure 3.1.1.

```
{
"0": {
    "vertex1": 0,
    "v2ertex": 1,
    "vertex3": 3
    },
"1": {
    "vertex1": 0,
    "vertex2": 4,
    "vertex3": 3
    }
}
```


Figure 3.1.1. A JSON object with two nested JSON objects representing triangles in the mesh. The name value pairs in the nested objects refer to the nodes that form that specific triangle.

3.2 Construction Process

The triangle mesh is developed in Processing where object-oriented programming principles become ideal for conceptualizing the warp mesh function. We want the mesh to represent the non-planar surface that is being projected on, however the mesh must be altered according to the desired configuration.

3.2.1 Node Class

The construction of the mesh begins by rendering nodes into a processing sketch. It is beneficial to understand the contribution that a Node object makes to this project. The role of this Node class may seem small, but it is the foundation for executing the entire process successfully. These nodes hold information that is necessary for making correspondences between the real world space and the projection.

A Node object is defined by three attributes, an x-coordinate, a y-coordinate and a radius that relates to the actual radius of the ellipse being drawn. With that said, we bring our attention to the methods of this Node class. The first method is a draw function that does not return any value, but executes the rendering of a Node object. The draw function calls ellipse() from the

processing documentation which renders an ellipse at some specified point. The other method within the Node class is contains() which takes two parameters x and y and returns a boolean value. The purpose of this function is to determine the difference in location between the mouse and the Node. Details on how exactly this is accomplished is explained in section 3.2.4.

3.2.2 Placing Nodes

To begin rendering the Nodes that make up the mesh into the processing sketch, a window with a black background is created. Note that this window is a result of calling fullScreen() within the processing setup() function, and the device displaying the screen is the projector. To complete the primary goal of populating the window with all the appropriate vertices, we observe the screen that is being projected onto the ceiling of the dome. When the mouse pointer within the display is hovering over the real world vertex on the ceiling, a node is manually placed at that location in the window.

Repeating this process for every real world dome vertex, results in a window that is populated with nodes at different locations corresponding to real world vertices. This process highlights the actual desired mapping from the real world dome coordinate, to a point in the projection.

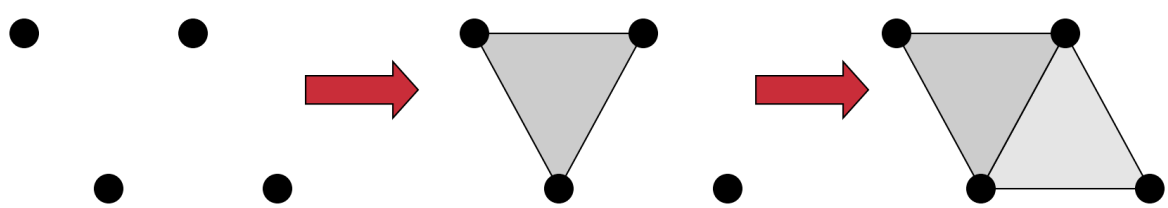


Figure 3.2.1. An example of how the triangles in the mesh are constructed. Nodes are rendered in the scene at various positions in the window. The desired edges are added between nodes to complete the mesh.

3.2.3 Storing Node and Triangle Information

As the nodes for the mesh are rendered, they are also added to an ArrayList of nodes. This allows for an integer to be connected to each node through indexing. Once every node in the window is added to the ArrayList, the nodes corresponding index is rendered above it. With

the nodes and indexes being displayed in the projection, each triangle can be identified by the nodes that make it up. To make a record of each triangle, once a triangle is observed a number is assigned to it and the nodes that make it up are logged. This is completed by creating a JSON object for each triangle, where each object contains three name-value pairs, pertaining to the index of the nodes in the ArrayList. The JSON objects are then saved to a .json file for future use. Each Node is recorded in a similar method, however the name value pairs are the x and y coordinates of that node.

After all the triangle and node data has been collected, edges can be added to the sketch to create a more vivid representation of the mesh. To draw the edges in the sketch, the function `line()`is used and takes the coordinates of the two points it connects as parameters. The .json file containing the triangle objects is parsed and for each triangle, each node making up the triangle is pulled (index). Using the index, the appropriate node is found within the ArrayList and the x and y values of the specified node object are used to render the lines. With the edges displaying in the projection it is clear to see that the projection of the mesh matches the real world configuration of the dome. However, when the projection is shifted away from the mirror, the distortion in the triangles is evident. The triangles are not close in size, in fact they are several different sizes as a result of the distortion from the curved mirror.

3.2.4 Manual Distortion Removal

Now that the mesh matching the configuration of the dome has been found, it is necessary to remove the distortion from this mesh to produce a "uniform mesh". To construct this mesh which is displayed in Figure 3.2.3, the nodes in the mesh were given a dragging functionality. This is accomplished using Processing's functions `mousePressed(), mouseDragged(), and mouseReleased()`. A global variable dragNode is initialized to null and then becomes the targeted node to be dragged. As the mouse is pressed, a check is made on every node to verify that the mouse is within the radius of the node. If this is true, then dragNode is set to that node and it can be dragged while the mouse is being dragged. When the mouse is released, the node's x and y values are set to the mouse x and y values and dragNode is set back to null.

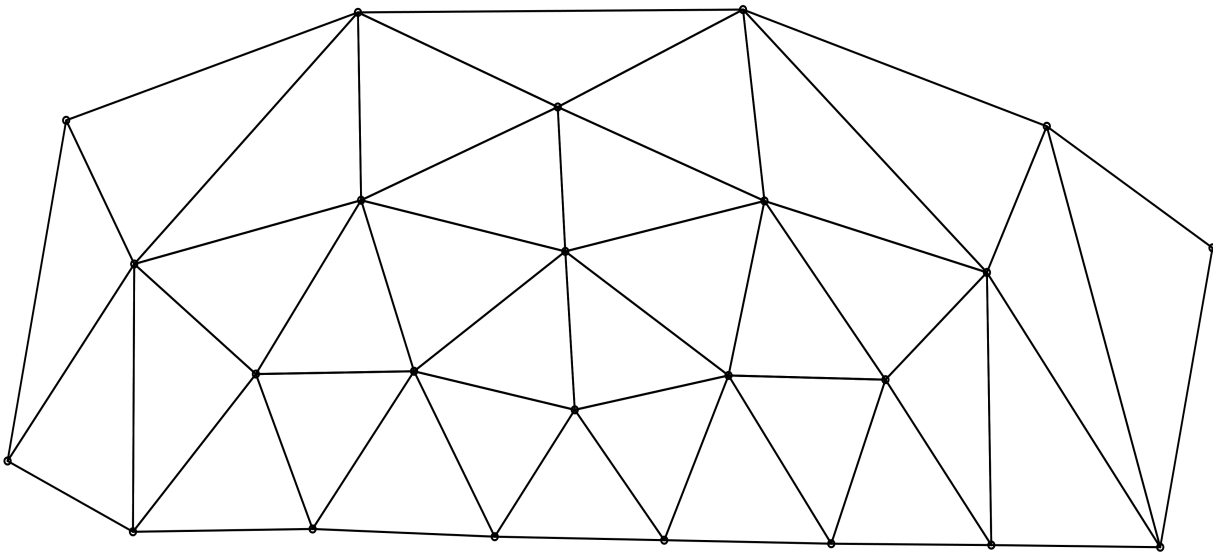


Figure 3.2.2. The mesh that represents the configuration of the dome in the spherical mirror projection. The mesh clearly shows how distorted the projection becomes when reflected off the spherical mirror.

In the real world dome configuration, the triangles are similar in size and not the many different sizes shown in Figure 3.2.2. The "uniform mesh" is constructed by dragging the nodes from the warped mesh so that the triangles are oriented as if the dome configuration was a single plane. Unfortunately this is an estimate and raises cause for error in the actual warping. Now that the triangles are closer in size and arranged as if every triangle was on the same plane, this mesh can be used to divide the original image into the appropriate number of triangles for warping.

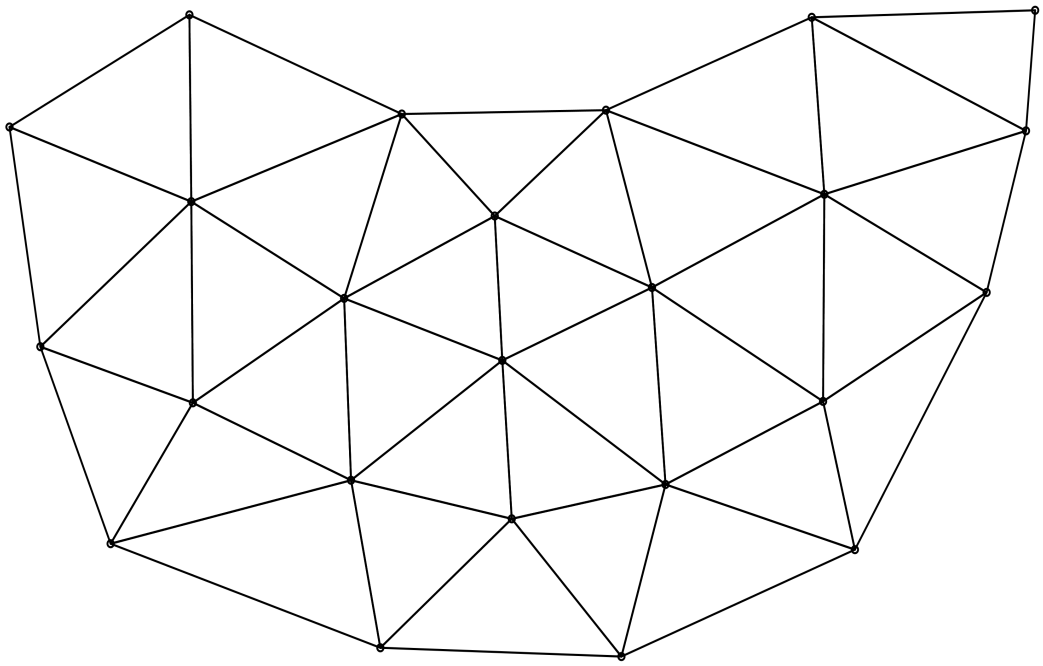


Figure 3.2.3. The mesh that is a representation of the nonplanar surface. The mesh serves as a tool for dividing the original image into the appropriate number of triangles. This mesh is an estimation of how the configuration would appear if it were laid out flat.

4
Mesh Warping

The warp mesh, from the previous chapter, will ultimately produce a warped image and there are several steps that are required to do this, as explained in this chapter. OpenCV will be the primary tool for executing the warp and the use of affine transformations will be prominent in understanding the algorithm.

4.1 Selecting a Test Image

After the warped image is retrieved, how well the algorithm performed must be visible in the projection of the output. To clarify, there must be a clear observation that verifies that the algorithm is functioning as intended. Not every image will contain characteristics that make it clear to identify that distortion was removed. It is important for the image to contain several vertical and horizontal lines so that the distortion will be easy to notice. The checkerboard image that is normally used for camera calibration, is an ideal choice for a test image. The squares in the image are all congruent and so this image is an excellent selection given that the variation in the lengths of the sides and the performance of the algorithm share an indirect relationship.

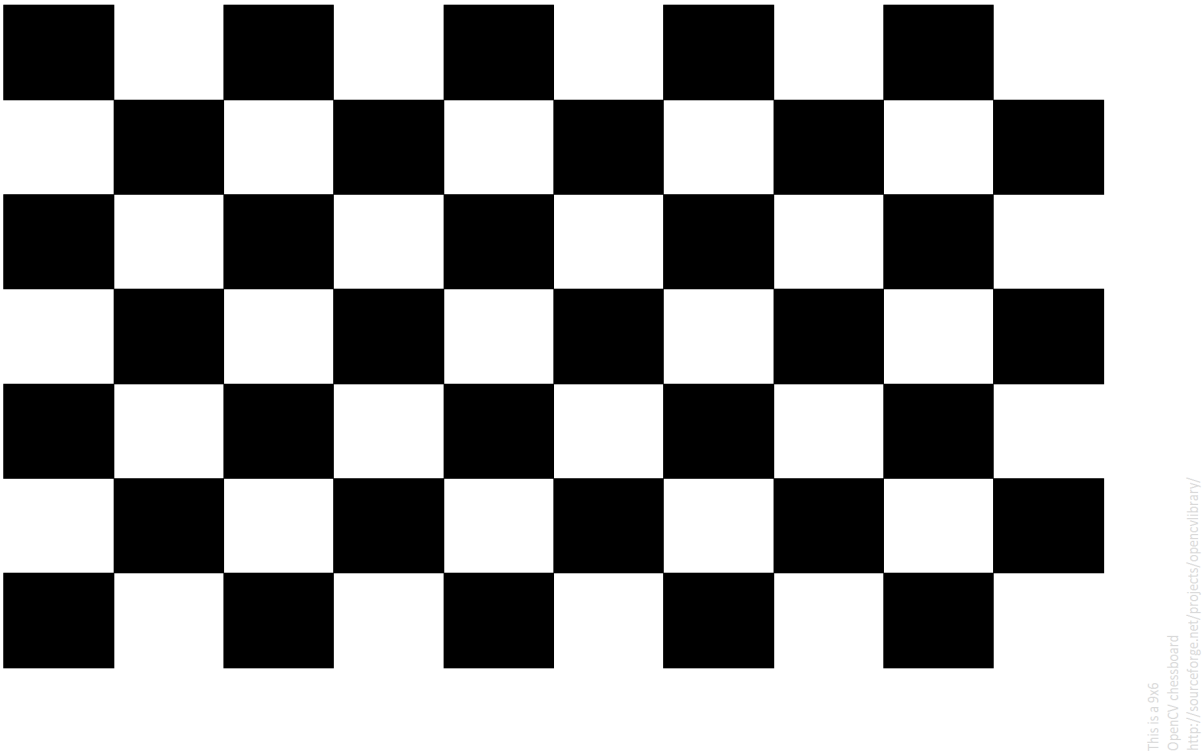


Figure 4.1.1. The checkerboard image used for analyzing the output of the image warping

4.2 Data Preparation

To warp the image, there is a number of things that are needed before that actual warping is executed. Within the Python script that warps the image, the .json files discussed in the previous sections are imported using the Python json library. Once imported, the node and triangle data can be accessed using the name of the variable holding the desired .json file.

Figure 4.3.1. An example of an ROI missing from the original image because it has been removed for warping. Note that this image is strictly to help visualize what happens when an ROI is selected.

4.3 The Masked Image

Isolating specific sections of an image is much simpler using Numpy and OpenCV. These libraries are crucial for masking an image, which is our method of choice for returning a triangular region in the image.

4.3.1 Region of Image

The region of an image or ROI for short, is what allows for the image reconstruction to occur. As discussed in the previous section, when attempting to isolate an image, an ROI is the portion that needs to be extracted. The "uniform mesh" from Section ??? is used to get each individual ROI. The points that make up each triangle are passed to a Numpy array object and this array is used to extract the ROI from the image.

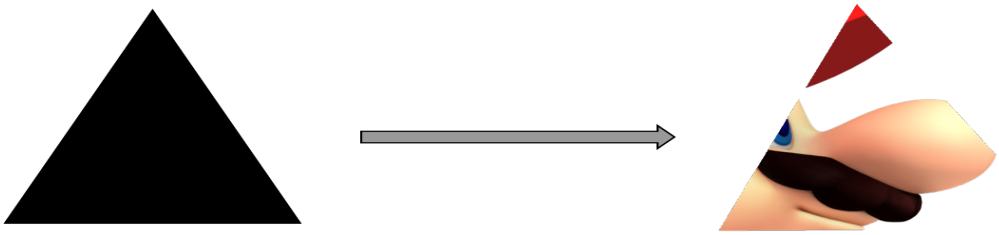


Figure 4.3.2. The points that correspond to the section of the image that is being isolated are used to mask the image. That triangular ROI is then filled with its corresponding location in the image being warped.

4.3.2 Masking

To distinguish the desired ROI from the rest of the image, a mask is applied. The Numpy array object containing the information for the ROI is passed to the OpenCV funciton fillPoly(). This function takes the source image as the first output, the region that is being targeted as the second parameter (In this case a numpy array) and a color for the pixels for the third parameter. The result from running fillPoly() is then passed to another OpenCV function called `bitwise_and()`.

4.4 Warp and Reconstruction

Warping is executed by applying multiple affine transformations to regions specified by the warp mesh. As the individual ROI's are warped, those individual sections must be stitched together to output a completely warped image.

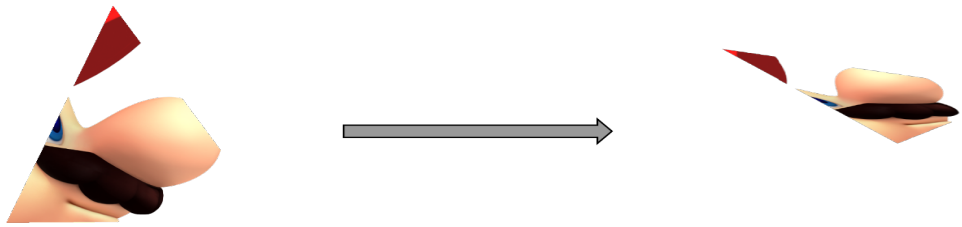


Figure 4.4.1. After the ROI is retrieved, an affine transform is applied to it and the triangles new position is outputted. An affine transformation example is shown in this figure.

4.4.1 Piecewise Affine Transformations

The Affine Transform should now be found for each warped image. We choose to write a function here that will return the warped ROI when successfully executed. We begin by obtaining the affine transform between the ROI and its destination. The points that make up the ROI and the destination points, are determined by the triangle that is being warped. The points are taken as parameters in the OpenCV function getAffineTransform(). This function calculates the affine transform that maps the source points to the destination points,

The affine transform obtained from getAffineTransform() must now be applied to the masked image. The openCV function warpAffine() performs this task for us by taking the image, affine transform, and size of the output as parameters. The destination image shows the transformation from one triangle to another and the distortion can be seen in the output. (Figure 4.4.1)

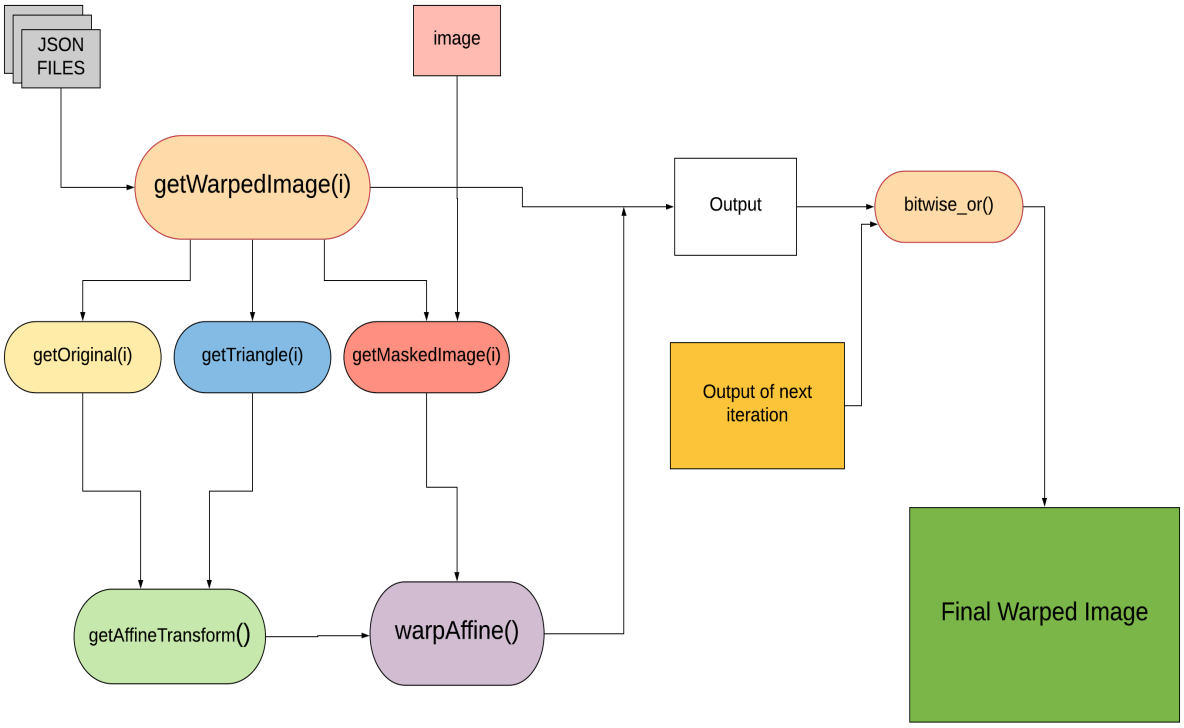


Figure 4.4.2. A visualization of the process that is executed from start to finish of the image warp. This work flow accurately represents the flow of data from one function to the next.

4.4.2 Combining the Images

The final step is to combine all the ROI's so that the full image is reconstructed. This is done by first declaring a variable that will cbe initialized with the first warped image. This will act as the base for the final warped output. Then using a for loop that variable is set to the output of running bitwise_or on itself and the ROI currently being accessed. The final warp is complete when the loop ends and the reconstructed image is saved as a PNG file.

Figure 4.4.3. A snapshot of the image reconstruction process. After each ROI is warped, it is added to its appropriate location in the final warped image.

5
Results

In this section we will analyze the success of attempting to remove distortion from the checkerboard image, as well as a few other images. Observing the output image will reveal how well features from the original image were preserved.

5.1 Output Image

The image that is outputted from the warp and reconstruction of each ROI is expected to have a lot of distortion before it is projected off the spherical mirror. This is expected since the distortion in the image is meant to be corrected after being reflected by the mirror. Before the output image is analyzed, it is crucial to understand what is being searched for to verify the distortion removal. This is done by identifying features in the original image, more specifically, the checkerboard image in this case. The checkerboard is made up of white and black squares, and these squares all share the same length sides. This information is suffice to analyze the output image and decide how effective the warping was. Since the squares all have the same length sides, the squares in the output image are expected to have the same length sides. Preservation of the dimensions of the squares is what will be searched for in the warped image. The lower the variation in aspect ratio of the squares, the less distortion that is present.

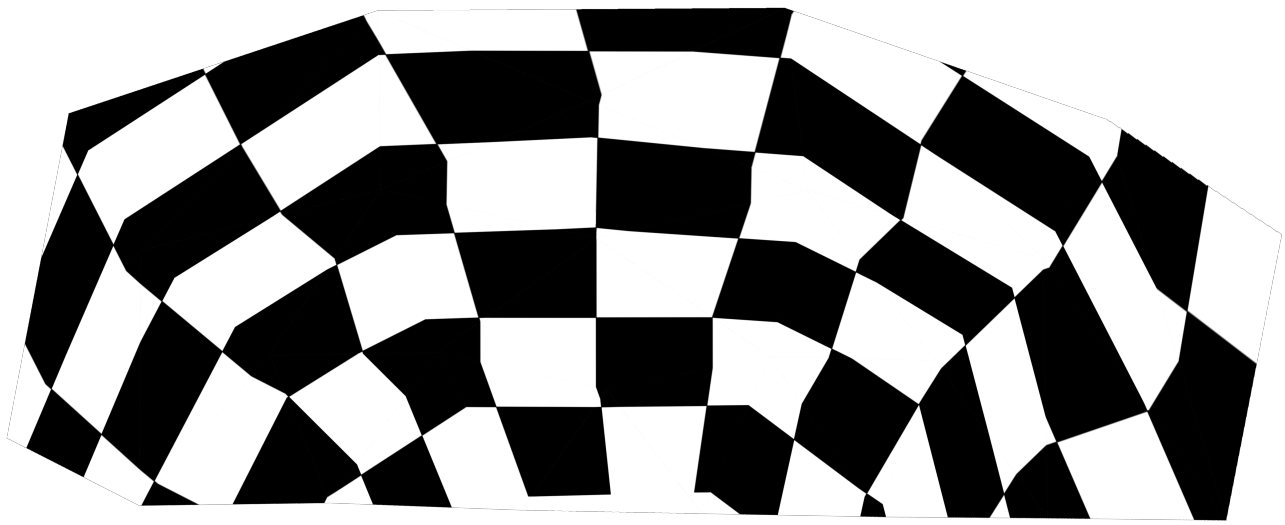


Figure 5.0.1. The output image after the geometric correction is applied to the checkerboard image.

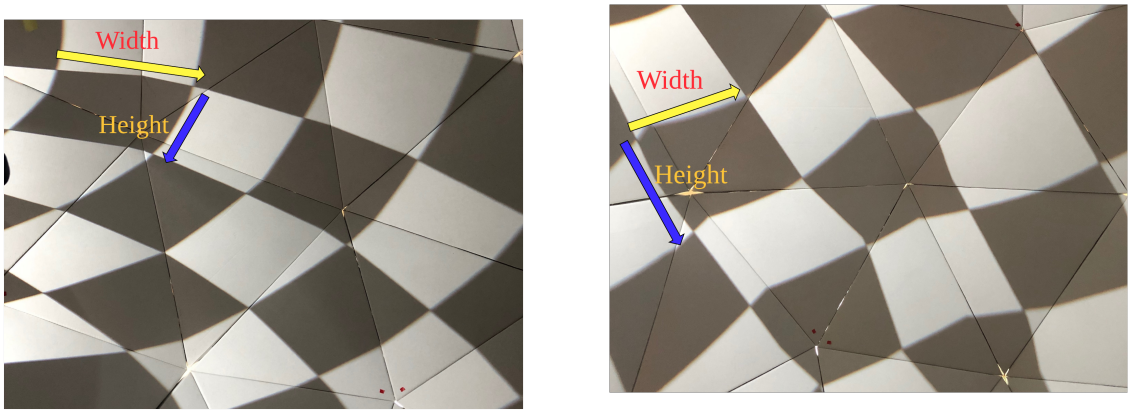
5.2 Analysis of Geometric Correction

To successfully analyze the output image presented in section 5.1, we must compare it to the projection of the original image before the correction was applied. The details of the comparison will be based on the measured widths and heights of 18 black squares from each image. To get the square dimension measurements, first the original image is projected and using a meter stick, the width and height are measured and recorded. This is done 18 times for the Original image as well as the output image. The values obtained from measuring the squares is shown in Figure 5.2.1. These values can help us determine how similar the squares are by calculating the variance among the side lengths. The variance is calculated for the sides of the squares in both images and the results are compared. To calculate the variance we use the equation

$$S^2 = \frac{\sum(x_i - \bar{x})^2}{n-1}$$

The variance for the set of width measurements from the original unwarped image of the checkerboard is 108.56. The variance for the set of height measurements for the same image is 81.79. The calculated variance of the width measurements from the warped image is 48.26 and the variance of the height measurements is 33.67. Fortunately there is a clear significant difference in the variance calculations for the lengths of the squares in the two images. The original unwarped image has a much higher variance among both the width and height measurements. This means that distortion was removed from the original projection during the attempt to preserve the original dimensions. A comparison between two images is shown in figure 5.2.1.

The aspect ratio for each square in both images is also determined using the width and height measurements. Calculating the aspect ratio's of the squares is done my setting up the ratio width:height, for every black square. Once these are found the variance among those values was found as well. The variance for the aspect ratio's of the squares in the original image is determined to be 0.38 while the variance for the ratio's in the corrected image is 0.018. With such a low variance in aspect ratio, it is valid to conclude that distortion was in fact removed after the geometric correction is executed.



(a) Snapshot of the original checkerboard image being projected off the spherical mirror. The distortion in the squares is evident.

(b) The original checkerboard image after the distortion is corrected. The dimensions of the squares are clearly closer in size. Notice the bend in some of the lines that should be parallel.

Figure 5.2.1. A comparison between two snapshots of the same image. (a) Showing the image before it is warped, while (b) shows the same image after the warping is executed

Though distortion was removed from the projection, the horizontal and vertical lines were not all straight in the output image. This is most likely caused by the estimation of the "uniform mesh" and is a direct result of human error when attempting to replicate the domes configuration on a planar surface. The dome configuration has specific dimensions that if represented in the mesh with perfect proportion's, may improve the success of the geometric correction.

To produce another example of removing distortion, the algorithm is applied to a more generic image. This is shown in Figure 5.2.3 and 5.2.4. These test cases aided in confirming the success of geometric correction. The output image clearly shows how the warping of the image caused by the spherical mirror is corrected.

Original Unwarped Image	
Width	Height
60 cm	20 cm
45 cm	35 cm
37 cm	40 cm
52 cm	28 cm
50 cm	35 cm
36 cm	43 cm
29 cm	46 cm
52 cm	40 cm
36 cm	54 cm
46 cm	51 cm
53 cm	40 cm
29 cm	45 cm
42 cm	39 cm
62 cm	36 cm
63 cm	27 cm
46 cm	33 cm
47 cm	52 cm
56 cm	46 cm

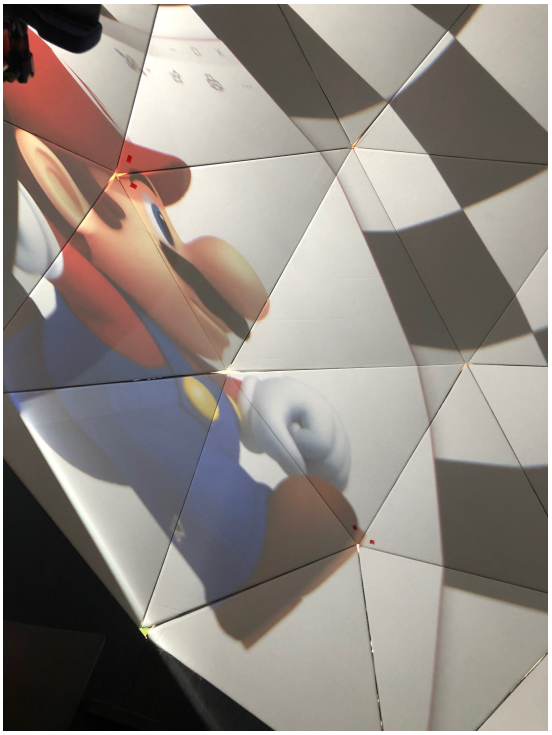
(a) Widths and heights for the squares in the original image

Output Image	
Width	Height
36 cm	33 cm
37 cm	40 cm
37 cm	42 cm
46 cm	52 cm
53 cm	45 cm
50 cm	52 cm
58 cm	53 cm
50 cm	52 cm
47 cm	46 cm
55 cm	54 cm
46 cm	47 cm
48 cm	46 cm
50 cm	37 cm
52 cm	53 cm
36 cm	46 cm
56 cm	46 cm
46 cm	46 cm
52 cm	48 cm

(b) Widths and heights for the squares in the output image

Figure 5.2.2. Black square widths and heights

(a) Full screen view of unwarped image displayed in projection.



(b) Unwarped image displayed in projection

Figure 5.2.3. The projection of a png file before the distortion is removed from the image. Notice the warp that can be seen in the projection of the image.

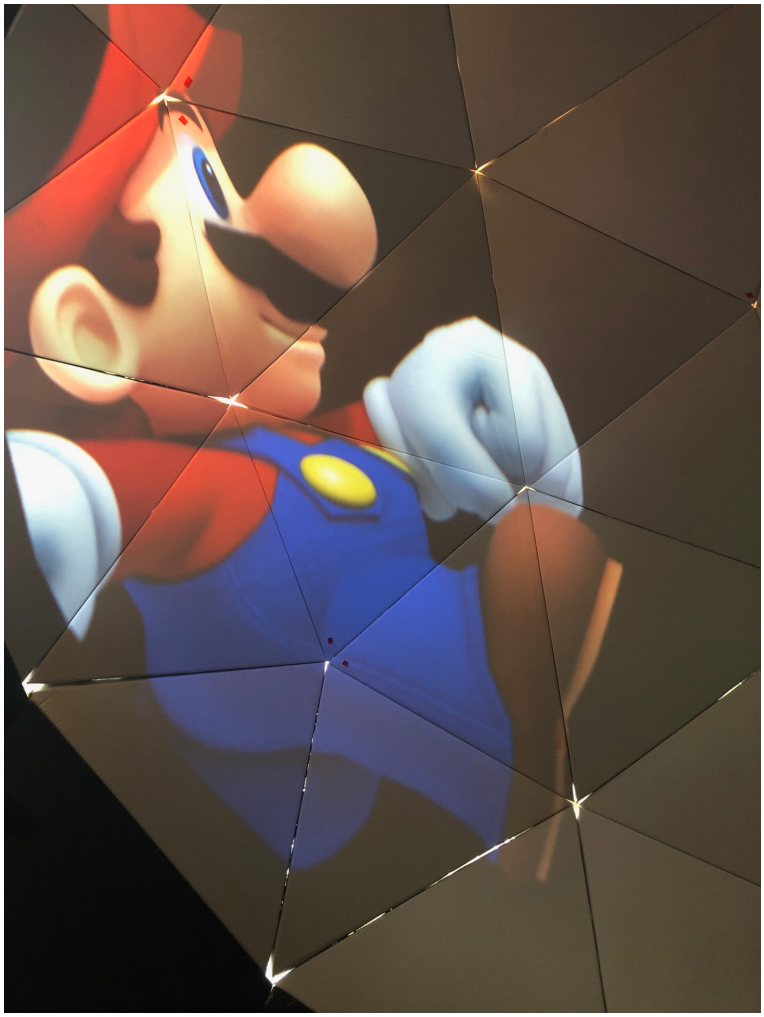


Figure 5.2.4. The projection of the output image after that image is corrected

6
Conclusion and Future Work

This project focused on removing distortion from a spherical mirror projection using affine transformations. The method of using the configuration of the dome was explored and resulted in positive findings. A mesh warp tool was developed in Processing and served its purpose as a way of retrieving points from the projection to make the appropriate correspondences. This mesh was also manipulated to mimic the configuration of the nonplanar view as if it were planar. This crucial step in the geometric correction process can definitely be improved and would greatly benefit the results of this project. A method of properly and accurately mimicking the configuration as if it were a planar surface may remove a significant amount of the distortion left in the output image.

The ultimate goal is to be able to interact with applications in an immersive environment and this project and taken a step forward in this subject. It is my hope that this project can be built upon and that the methods utilized in this work will aid in the automation of geometric correction. It is unfortunate that this project was not able to perfect the process, however, configuration replication has been proven to be a viable option for working with projector-camera systems in multi-planar surfaces. In future work involving projections on nonplanar surfaces using spherical mirror projections, this project should be referenced to help improve calibration in a projector-camera system.

Appendix A
Python and Processing

A.1 Execute_warp.py

```
#
# Geometric Correction Execution file
#
# Methuen Jelani Bell-Isaac
# Bard College Class of 2019

'''
This Python script executes image warping using
affine transformations to remove distortion
from a projection.
'''

import cv2
import numpy as np
import json

image = cv2.imread('mario.png',-1)

# resolution of display that is being projected
dsize = (4800, 2700)

triangles = open('triangles.json')
json_file = open('full_mesh2.json') # destination nodes
original_json = open('test3.json') # source nodes

data = json.load(json_file)
org = json.load(original_json)
```

```
tri = json.load(triangles)

'''
Get the source nodes
return a tuple of x and y value pairs
'''
def getOriginal(pid):
    triangle = tri[str(pid)]
    point_1 = (org[str(triangle["v1"])]["x"] , org[str(triangle["v1"])]["y"])
    point_2 = (org[str(triangle["v2"])]["x"] , org[str(triangle["v2"])]["y"])
    point_3 = (org[str(triangle["v3"])]["x"] , org[str(triangle["v3"])]["y"])

    return point_1, point_2, point_3

'''
Get the destination nodes
return a tuple of x and y value pairs
'''

def getTriangle(pid):

    triangle = tri[str(pid)]
    point_1 = (data[str(triangle["v1"])]["x"], data[str(triangle["v1"])]["y"])
    point_2 = (data[str(triangle["v2"])]["x"], data[str(triangle["v2"])]["y"])
    point_3 = (data[str(triangle["v3"])]["x"], data[str(triangle["v3"])]["y"])

    return point_1, point_2, point_3

'''
Apply masking
return an ROI that has isolated the desired triangle
'''
def getMaskedImage(pid):
    mask = np.zeros(image.shape, dtype=np.uint8)
    p1, p2, p3 = getOriginal(pid)
    roi = np.array([[p1, p2, p3]], dtype=np.int32)
    # fill the ROI so it doesn't get wiped out when the mask is applied
    channel_count = 4  # i.e. 3 or 4 depending on your image
    ignore_mask_color = (255,)*channel_count
    cv2.fillPoly(mask, roi, ignore_mask_color)

    # apply the mask
    masked_image = cv2.bitwise_and(image, mask)

    return masked_image

'''
Get and apply the affine transformation
```

```
Return the warped ROI

'''
def getWarpedImage(pid):
    s1, s2, s3 = getOriginal(pid)
    d1, d2, d3 = getTriangle(pid)
    masked_image = getMaskedImage(pid)
    src = np.float32([[s1[0] ,s1[1]], [s2[0] ,s2[1]], [s3[0],s3[1]]])
    dest = np.float32([[d1[0],d1[1]], [d2[0],d2[1]], [d3[0],d3[1]]])
    affine_transform = cv2.getAffineTransform(src,dest)
    warped_image = cv2.warpAffine(masked_image, affine_transform, dsize)

    return warped_image

final_warp = getWarpedImage(0) # Initialize the final image with the triangle at

counter = 0

'''
Get the warped ROI for each triangle in the mesh and
Combine the warped images to construct the complete warped image
'''
for i in tri:


    img = getWarpedImage(i)

    final_warp = cv2.bitwise_or(final_warp, img)
    print("Triangles left: " + str((len(tri) - counter)))
    counter += 1

print("Image Warp Successful!")
cv2.imwrite('image4.png', final_warp)
```

A.2 dome_mesh.pde

```
ArrayList nodes;
Node dragNode = null;
int mesh_color = 0;

/*
* This sketch is used to develop the warp mesh
* This sketch allows the mesh to be manipulated
* All node information is coming from a .json file
* Unless new nodes are added to the scene
*/

void setup() {
  nodes = new ArrayList();
  fullScreen(2);
  strokeWeight(6);

}


void draw() {

  //JSON object of nodes
  JSONObject data = loadJSONObject("full_mesh.json");

  //JSON object of triangles
  JSONObject tris = loadJSONObject("triangles.json");

  background(255);

  //draw nodes from the "nodes" arraylist as long as it is not empty
  if (nodes.size() > 0) {
    for (int i = 0; i < nodes.size(); i++) {
      Node p = (Node) nodes.get(i);
      stroke(mesh_color);
      p.draw();
    }
}


//Render nodes into the Processing sketch directly from the .json file
for (int i = 0; i < data.size(); i++) {
  JSONObject json = data.getJSONObject(str(i));


  int x = json.getInt("x");
```

```
  int y = json.getInt("y");


  Node n = new Node(x, y, 10);


  if (nodes.size() < data.size()) {
    nodes.add(n);

  }


  //textSize(96);
  //text(str(i), x-30, y-75);
  //fill(0, 255, 255);

}

// Draw the edges for each triangle
for (int i = 0; i < tris.size(); i++) {

  JSONObject shape = tris.getJSONObject(str(i));

  int l = shape.getInt("v1");
  int m = shape.getInt("v2");
  int n = shape.getInt("v3");

  Node a = (Node) nodes.get(l);
  Node b = (Node) nodes.get(m);
  Node c = (Node) nodes.get(n);

  JSONObject v1 = data.getJSONObject(str(l));
  float v1x = a.x;
  float v1y = a.y;
  JSONObject v2 = data.getJSONObject(str(m));
  float v2x = b.x;
  float v2y = b.y;
  JSONObject v3 = data.getJSONObject(str(n));
  float v3x = c.x;
  float v3y = c.y;

  stroke(mesh_color);

  line(v1x, v1y, v2x, v2y);
  line(v1x, v1y, v3x, v3y);
  line(v2x, v2y, v3x, v3y);

}
```

```
}
```

List of Figures

1.4.1 Geodesic dome exterior 4

1.4.2 Geodesic dome interior. 5

1.4.3 The interior setup of the spherical mirror and the Infocus IN3136a 720p projector. 6

2.1.1 A simple representation of two image transformations on a right triangle. The translation example has shifted the triangle by some factor after applying a translation vector to the triangles vertices. The rotation example shows a 90 degree rotation. 8

3.1.1 A JSON object with two nested JSON objects representing triangles in the mesh. The name value pairs in the nested objects refer to the nodes that form that specific triangle. 15

3.2.1 An example of how the triangles in the mesh are constructed. Nodes are rendered in the scene at various positions in the window. The desired edges are added between nodes to complete the mesh. 16

3.2.2 The mesh that represents the configuration of the dome in the spherical mirror projection. The mesh clearly shows how distorted the projection becomes when reflected off the spherical mirror. 18

3.2.3 The mesh that is a representation of the nonplanar surface. The mesh serves as a tool for dividing the original image into the appropriate number of triangles. This mesh is an estimation of how the configuration would appear if it were laid out flat. 19

4.1.1 The checkerboard image used for analyzing the output of the image warping . . . 22

4.3.1 An example of an ROI missing from the original image because it has been removed for warping. Note that this image is strictly to help visualize what happens when an ROI is selected. 23

4.3.2 The points that correspond to the section of the image that is being isolated are used to mask the image. That triangular ROI is then filled with its corresponding location in the image being warped. 24

4.4.1 After the ROI is retrieved, an affine transform is applied to it and the triangles new position is outputted. An affine transformation example is shown in this figure. 25

4.4.2 A visualization of the process that is executed from start to finish of the image warp. This work flow accurately represents the flow of data from one function to the next. 26

4.4.3 A snapshot of the image reconstruction process. After each ROI is warped, it is added to its appropriate location in the final warped image. 27

5.0.1 The output image after the geometric correction is applied to the checkerboard image. 30

5.2.1 A comparison between two snapshots of the same image. (a) Showing the image before it is warped, while (b) shows the same image after the warping is executed 31

5.2.2 Black square widths and heights 33

5.2.3 The projection of a png file before the distortion is removed from the image. Notice the warp that can be seen in the projection of the image. 34

5.2.4 The projection of the output image after that image is corrected 35

Bibliography

[1] Jan Erik Solem, *Programming Computer Vision with Python*, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2012.

[2] Richard Szeliski, *Computer Vision: Algorithms and Applications*, Springer, Salmon Tower Building, New York City, 2010.

[3] Paul Bourke, *Using a spherical mirror for projection into immersive environments*, `http://paulbourke.net/papers/graphite2005/`.

[4] Darren Tirto, *Projector-Camera Calibration for a Multi-Planar Dome*. Senior Thesis, 2018.

[5] Kai Malowany, *Beyond homographies: Exploration and analysis of image warping for projection in a dome*. Senior Thesis, 2017.

[6] OpenCV, *Introduction to OpenCV-Python TutorialsUsing a spherical mirror for projection into immersive environments*, `https://docs.opencv.org/3.4.2/d0/de3/tutorial_py_intro.html`.

[7] Opencv, *Geometric Transformations of Images*, `https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html#geometric-transformations`.