Image Warping for Geodesic Dome Projection with Spherical Mirror

Henry Chang, Prof. Keith O'Hara

June-August 2019

Motivation

In order to give students a realistic and active experience of using remote sensing robots, we need a system of straightforward interfaces to control robots remotely and collect feedback wirelessly. Thus, the goal of the system is to provide an inexpensive, easy-to-use, configurable robot interface (e.g. for drones) in more humble settings than places like large museum planetarium installations. This image warping project, as part of developing this larger system for robotics education, aims to create an immersive projection environment for spherical views such as the night sky, outer space and physical world (these views would be collected by robots with wide angle cameras).

Introduction

Optics-wise,the center of the dome is the best spot for a complete and undistorted view of the projection, some domes come with a spherical mirror that could be put near the peripheral of the dome to assist projection and save space for the viewer. The round dome surface as well as a spherical mirror distort the projected image. Thus, pre-warping the image is required for the projected image to look as desired on the dome surface. The challenge is that spherical mirror projection breaks the symmetry of the projection environment and complicates the pre-warping process. I used light ray tracing to find the correspondence between the image plane and the projection surface,

and therefore the mapping between the pre-warped pixels and the original pixels. I compare the results of using forward and backward light ray tracing.

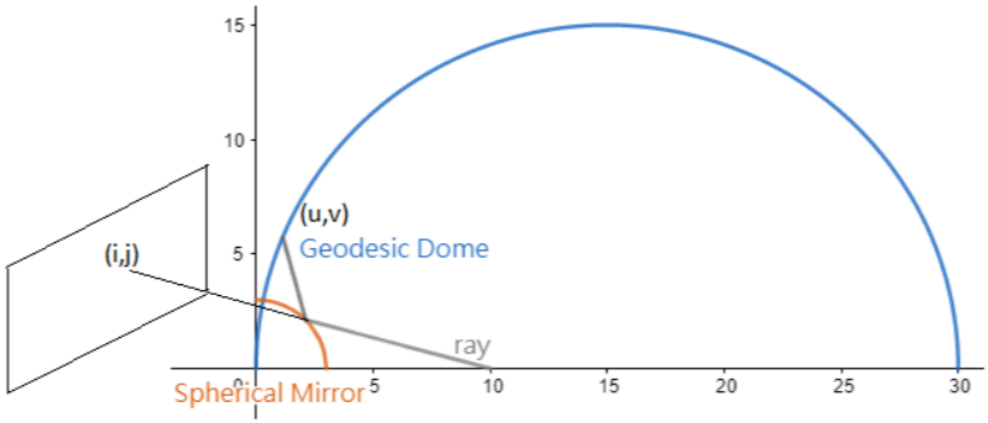
Problem

To generate the pre-warped image, two things need to be known: 1) knowing where each pixel should land on the dome to display the correct image 2) knowing where each pixel would land on the dome after projection.

(1) is done by a 2d circle to a 3d hemisphere mapping, of which the key point is that for any point in 2d, the ratio of its distance from the circle center corresponds with the ratio of its 3d counterpart's angle to the z-axis (assuming both have center at origin and have the same radius). We also assumed that the viewer's eye's are near to the center of the dome, so light ray travel the same distance from the dome to the viewer.
(2) is more complicated, and involves light ray tracing.

Light Ray Tracing

Light ray tracing is a common technique in computer graphics to generate desired images because it follows the physics of optics, and is known as first principle for solving projection problems. I approach the problem using light ray tracing, utilizing both directions: forward tracing is tracing the light in the direction of the actual projection and backward the opposite. With forward tracing, where each pixel location would be projected on the dome can be found; with backward tracing, given a location on the dome, we know which pixel contributes to the projection there. Forward tracing gives the projection function while backward gives its inverse, which is what we need for pre-warping. However, the difficulty is that backward tracing involves finding the spot where each light ray is reflected off the spherical mirror – a hard optics optimization problem that can only be solved numerically (takes time) but not analytically. On the other hand, forward tracing only involves simple math (law of reflection) and could be done efficeintly, but can easily shoot light rays out of ROI (region of interests) and cause calculation chaos. Forward tracing also requires interpolation afterwards to find the final pre-warping.



This is a diagram of our projection environment. The vertical is the z-axis, horizontal the x and in/out of page direction is the y-axis.

Forward tracing gives the projection function $F(i,j) = (u,v)$, for each (i,j) in the image plane. Backward tracing gives the inverse projection function $B(u,v) = (i,j)$, for each (u,v) on the projection plane. To generate the pre-warped image, we need to either set $warpedImage[(x,y)] = originalImage[F(x,y)]$ or $warpedImage[B(x,y)] = originalImage[(x,y)]$.

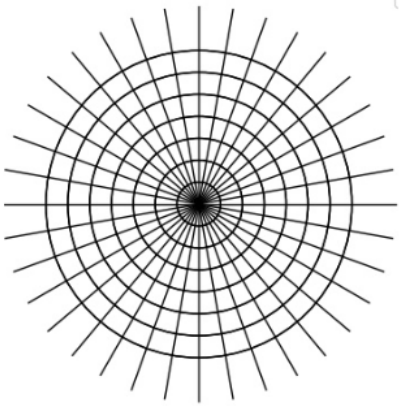
Backward tracing is essentially the problem: given two points, the projector coordinate (pinhole model) and the a point on the dome, where is the light reflected off the mirror? This turns out to be a optimization problem that can be solve numerically but not analytically [2]. As explained in [1], according to Fermat's principle, light travels the shortest distance possible. The problem becomes finding the reflection point such that the distance the a light ray travels from the projector to mirror, then to the point on the dome is the shortest possible. Solving backward tracing is mathematically hard and computationally inefficient. However, for each projection setting, light always travel the same way and the mapping is reusable for multiple images. Thus, if we have a particular setting that will remain the same for a long time, we can find prewarping with only backward tracing and reuse it. Same applies for forward warping.

Algorithm Design

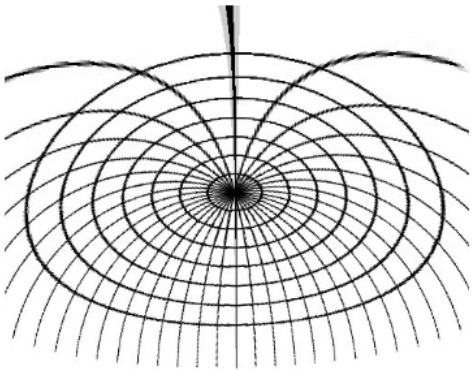
The pre-warping algorithm is essentially putting the image plane and the projection environment into the same coordinate system to perform light ray tracing and find the mapping between the image plane and projection plane. I use forward and backward tracing to find these mappings. Backward and forward tracing are intuitive coordinate transformations, with each step describe below:

Forward: image to projection plane (translation and rotation) $\rightarrow$ image plane to sphere (pinhole model, projection center and projection plane)$\rightarrow$ sphere to dome (law of reflection)$\rightarrow$ dome to circle(3d to 2d) $\rightarrow$ 2d to image
Backward: image ROI to 2d circle (translation)$\rightarrow$ circle to sphere/dome (2d to 3d) $\rightarrow$ dome to mirror (optimization: minimize) $\rightarrow$ image to projection plane(pinhole projection) $\rightarrow$ image plane to image

Once we have found the correspondences between (u, v) and (i, j). The warping is done with setting $warpedImage[(x, y)] = originalImage[F(x, y)]$ or $warpedImage[B(x, y)] = originalImage[(x, y)]$. A demonstration of a regular polar mesh and its pre-warped version that if projected, will give back the regular mesh. (Given the regular mesh, my program will generate the pre-warped mesh.)



Regular Mesh



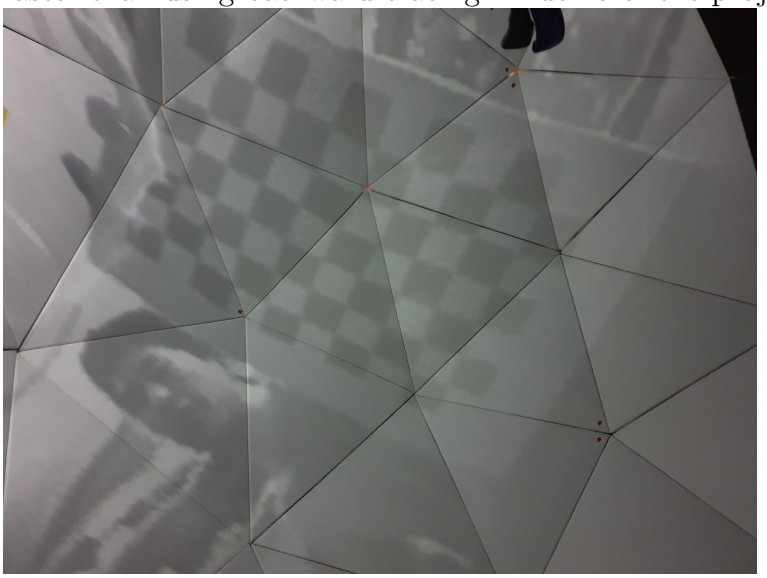
Pre-Warpped Mesh

The pre-warping program's implementation is done in Python, Processing and OpenGL. Python provides available optimization packages that can be used for backward tracing. Processing offers an interface for visualization of the projection environment and for easy implementation of OpenGL, which utilizes GPU to accelerate the pre-warping. The program files and documentation can be found at link (click).

Result and Analysis

Image size	20x20	50x50	100x100	200x200	500x500	1080x1080	2160x2160
Forward Runtime(s)	0.5	3.4	12.95	46	342	1487	x
Backward Runtime(s)	x	x	x	x	10	x	156

As we can see from the runtime result, the computational work done is proportional to the size of the image–the average work done to find the correspondence for a pixel is the same regardless of image size. However, we see the advantage of using forward tracing because it can find the warp 30 times faster than using backward tracing. A demo of the projection on the dome:



References

[1] Paul Bourke. Using a spherical mirror for projection into immersive environments (mirrordome). *Proceedings of the 3rd international conference*

on Computer graphics and interactive techniques in Australasia and South East Asia.

[2] David Eberly. Computing a point of reflection on a sphere.