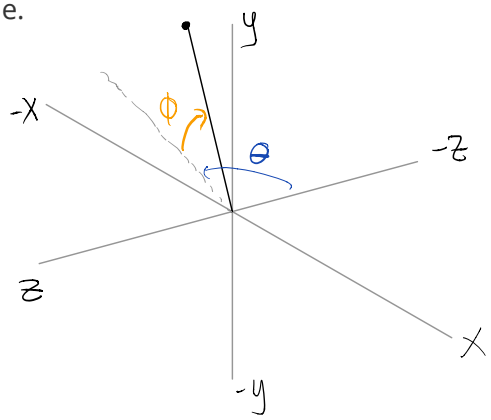
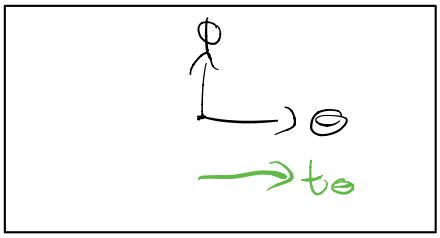
Our goal is to project a planar image on to a spherical surface. Our output image is one where horizontal and vertical coordinates correspond to **angles** that parameterize a spherical surface, much like how latitude and longitude coordinates parameterize the surface of the roughly-spherical 3D earth using two coordinates.

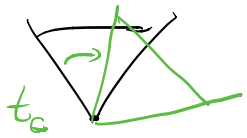
Our spherical coordinate system will use:

- **azimuth** (θ), the angle in the xz plane from the $-z$ axis in the direction of the $-x$ axis), and
- **elevation** (ϕ), the angle towards the $+y$ axis from the xz plane.

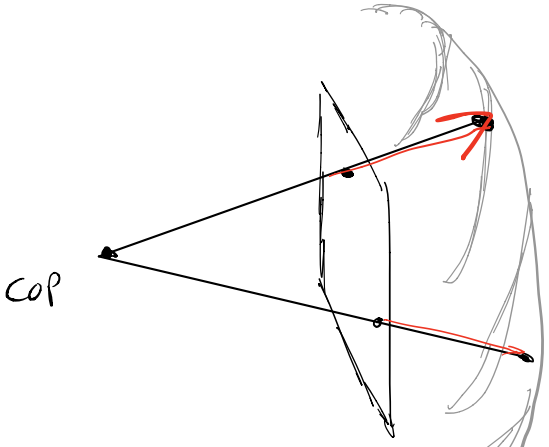


Once the images are in spherical coordinates, the alignment can be done using a translational motion model, because translation in spherical coordinates corresponds to rotation in Euclidean space; because the images are assumed to be taken with the same Center of Projection, the relationship among the cameras can be fully described by a rotation around that COP.



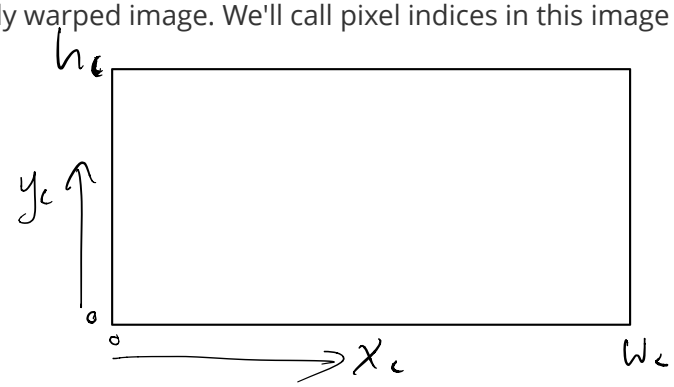


If we were doing forward warping, we'd start in the image coordinates and project onto the surface of the sphere.



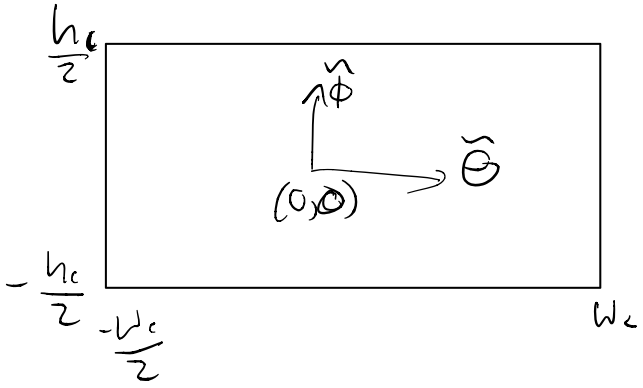
Because inverse warping allows us to use interpolation to handle non-integer outputs of transformation matrices, we need to go the opposite way: given a position in the output image, find the input image pixel coordinates of the value that projects there. This amounts to a series of coordinate transformations, not all of which are linear (so we can't just do it with a single matrix).

1. Start with the spherical output image, where we have an h_c-by-w_c array representing the spherically warped image. We'll call pixel indices in this image x_c and y_c:



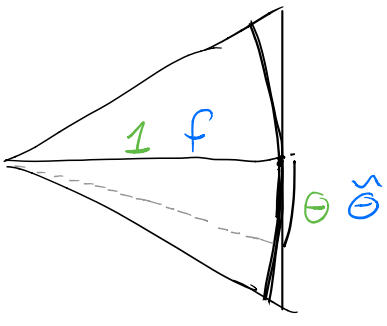
2. Start by moving the image so the origin is at the center:

$$\tilde{\theta} = x_c - \frac{w_c}{2}$$
$$\tilde{\phi} = y_c - \frac{h_c}{2}$$



2. Now convert $(\tilde{\theta}, \tilde{\phi})$ into actual angles. We'll assume our output image represents just the range of horizontal and vertical angles needed to fit the projection of all our image pixels. The focal length of hte camera is f, so we'll position the sphere so that it touches the image at the origin, therefore has radius f. The angle represented by the distance from 0 (the image center) to a coordinate $\tilde{\theta}$ is the arc length subtended by $\tilde{\theta}$ pixels in our output image. That arc lenghth can be calculated based on the following ratio equality:

$$\frac{\theta}{2\pi} = \frac{\tilde{\theta}}{2\pi f}$$

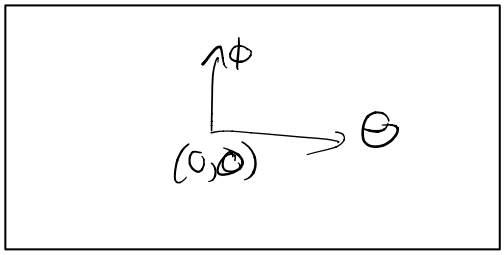


Think of this as the ratio of the length of the arc to the circumference of the sphere in a unit sphere (left side) and a sphere with radius f (right side). Rearranging to solve for θ, we get:

$$\theta = \frac{\tilde{\theta}}{f}$$

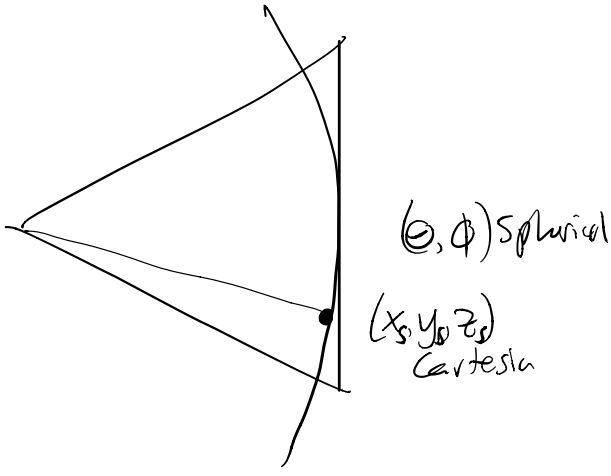
Combining the above two steps, our angular coordinates are:

$$\theta = \frac{x_c - \frac{w_c}{2}}{f}$$
$$\phi = \frac{y_c - \frac{y_c}{2}}{f}$$



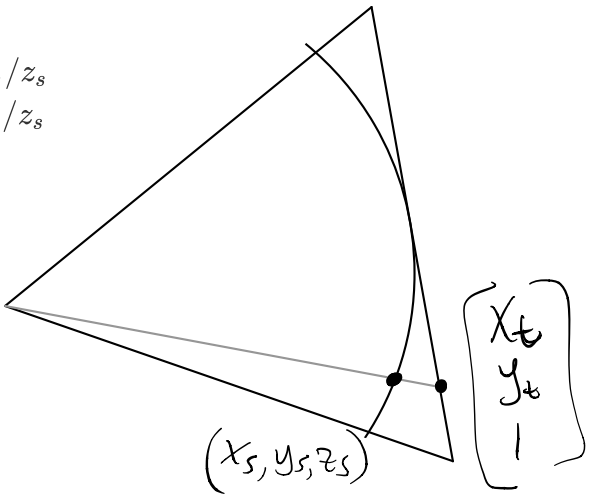
3. We have the angles represented by a particular pixel index in our output image. Now let's convert that into 3D cartesian coordinates and figure out where that point projects into the actual camera's image plane. The spherical-to-cartesian conversion, which can be derived using a bit of trigonometry, is:

$$x_s = \sin\theta\cos\phi$$
$$y_s = \sin\phi$$
$$z_s = \cos\theta\cos\phi$$

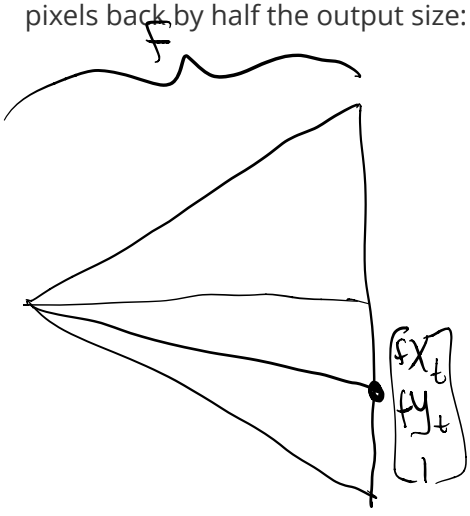


4. Now we have a 3D point, and we want to know where it lands on the image plane. The image plane in our 3D coordinate system lives at a distance 1 in the $-z$ direction from the COP (origin). To project it onto that plane, we simply scale it by $1/z$ to get the point where that projection ray intersects with the $z = -1$ plane.

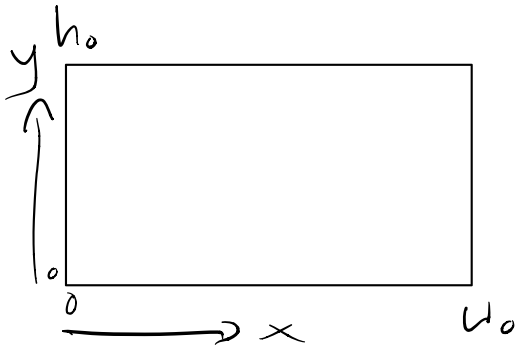
$$x_t = x_s/z_s$$
$$y_t = y_s/z_s$$
$$z_t = 1$$



5. Now that we're on the image plane in 3D, we need to get back to pixel coordinates. The first step is to get into the 2D homoegeneous coordinate system where the image plane is actually at a distance f and the origin in the corner. So we'll replace the $z = 1$ with $w = f$ and shift all the pixels back by half the output size:

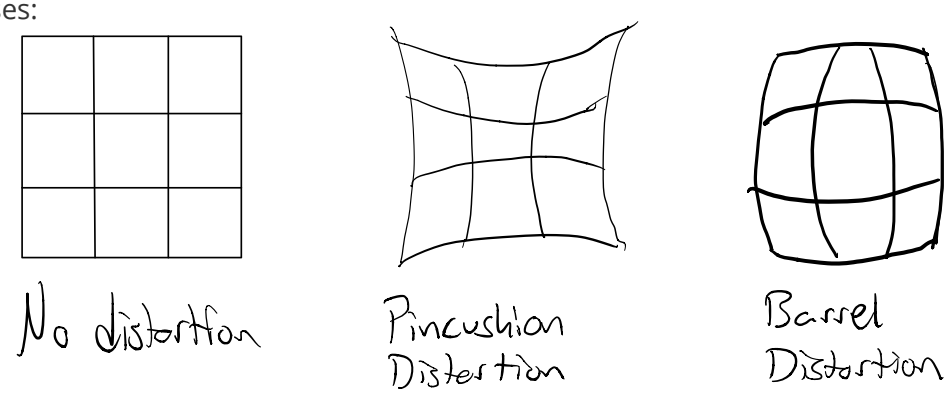


$$x = fx_t + \frac{w_o}{2}$$
$$y = fy_t + \frac{h_o}{2}$$
$$w = 1$$



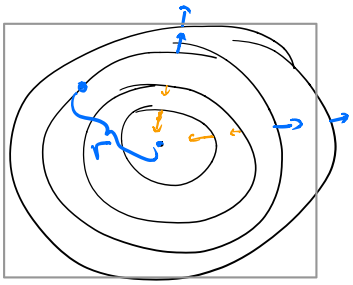
which finally gets us to pixel coordinates in the image plane.

6. There's one more issue we need to handle, which is called **radial distortion**. This is a place where the pinhole camera model breaks down significantly enough in practice that we need to expand our model a little. Radial distortion is a kind of nonlinear warping that occurs due to the realities of lenses:



We'll model these types of distortion with the assumption that pixel positions are nudged inward or outward radially by an amount that is a predictable function of the distance (r) to the center of the image. In particular, a common choice for modeling this is to assume that the distortion is a quadratic in r^2:

Parameters - fit from an img of a checkerboard.



$$x_d = x_t(1 + k_1 r^2 + k_2 r^4)$$
$$y_d = y_t(1 + k_1 r^2 + k_2 r^4)$$

As implied by the notation, this distortion is applied on the x_t, y_t coordinates, which live on the image plane 3D, but before we go into 2D homegeneous camera coordinates. r^2 can be calculated directly as $x_t^2 + y_t^2$, with no need to take a square root.