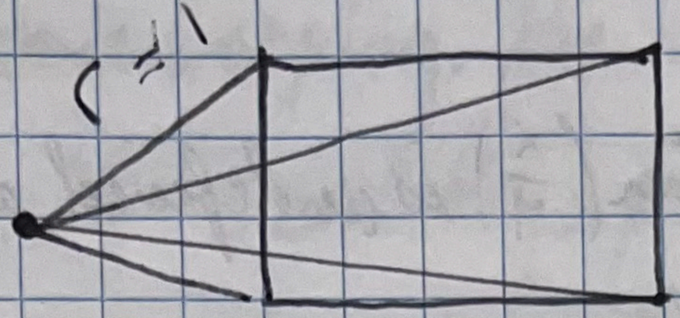
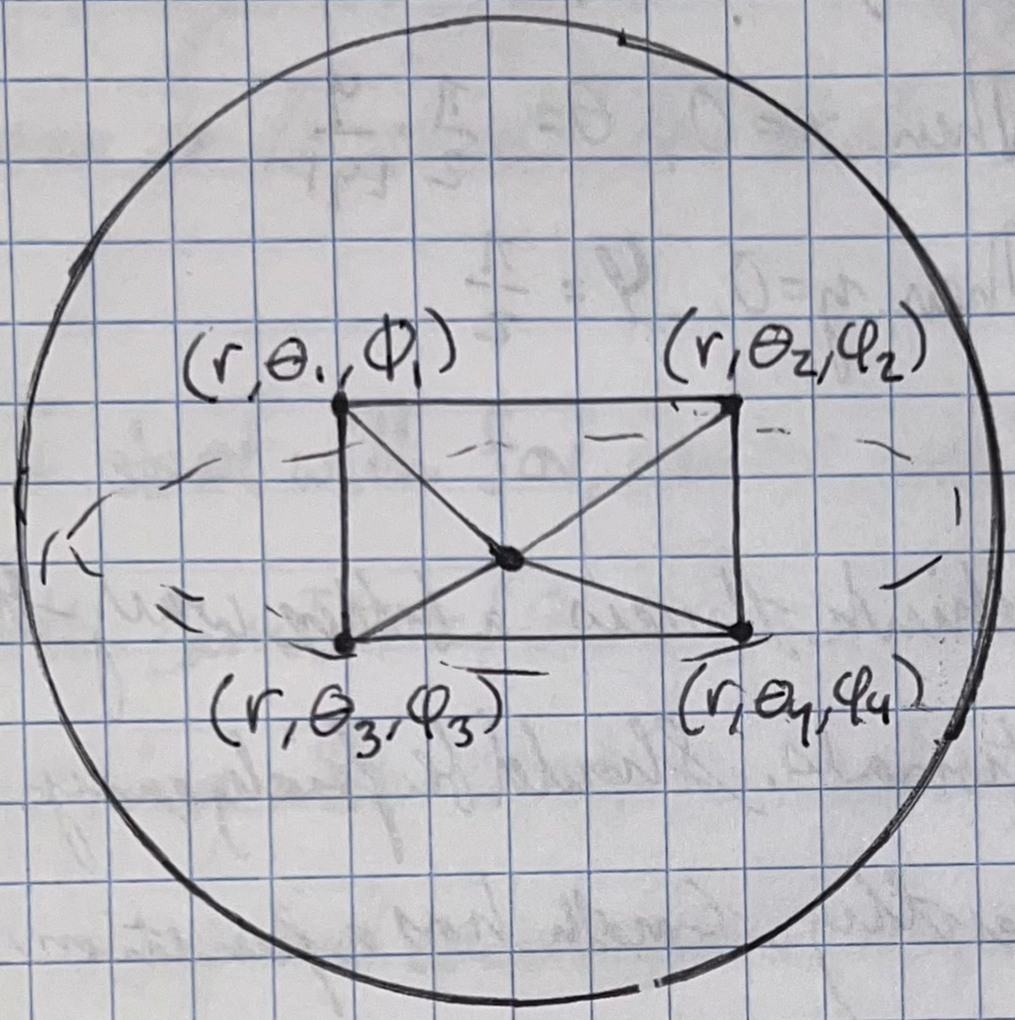
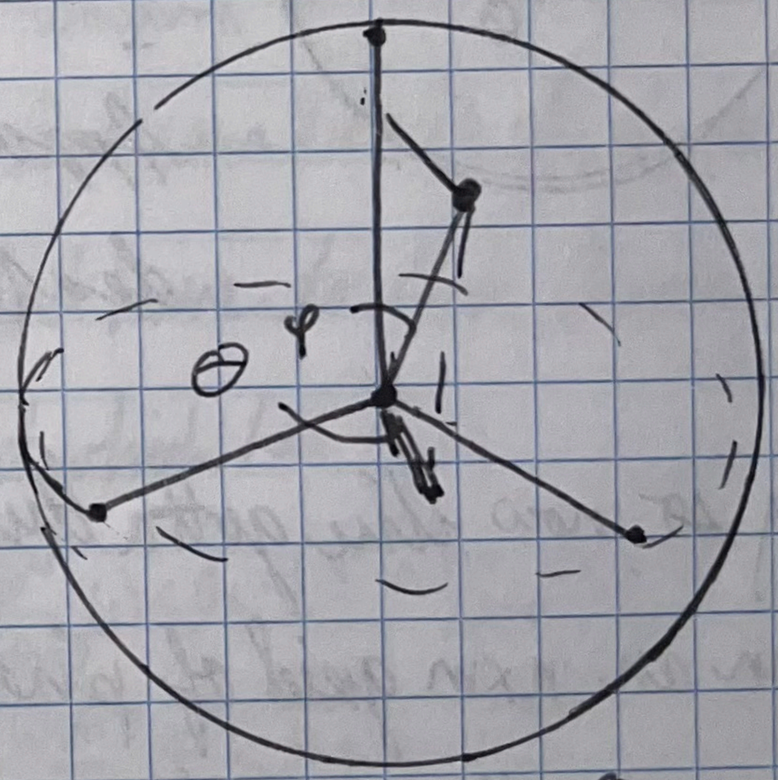




Given a vector, $\vec{v}$, I need to transform to polar. It's finally time to learn how to use the Jacobian!

Then I'm going to need to figure out how to describe the camera and projection onto the screen.



$$\begin{pmatrix} r \\ \theta \\ \varphi \end{pmatrix} = f\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \sqrt{x^2+y^2+z^2} \\ \arctan(y/x) \\ \arctan(\sqrt{x^2+y^2}/z) \end{pmatrix}.$$

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} \\ \vdots & & \\ \frac{\partial f_3}{\partial x} & \frac{\partial f_3}{\partial y} & \frac{\partial f_3}{\partial z} \end{pmatrix} = \begin{pmatrix} \frac{x}{\sqrt{x^2+y^2+z^2}} & \frac{y}{\sqrt{x^2+y^2+z^2}} & \frac{z}{\sqrt{x^2+y^2+z^2}} \\ -\frac{y}{x^2+y^2} & \frac{x}{x^2+y^2} & 0 \\ \frac{xz}{(x^2+y^2+z^2)\sqrt{x^2+y^2}} & \frac{yz}{(x^2+y^2+z^2)\sqrt{x^2+y^2}} & -\frac{\sqrt{x^2+y^2}}{x^2+y^2+z^2} \end{pmatrix}.$$

It occurs to me that the Jacobian might be a stupid-ass idea.
It might be easier to just apply the change of coords column-wise.

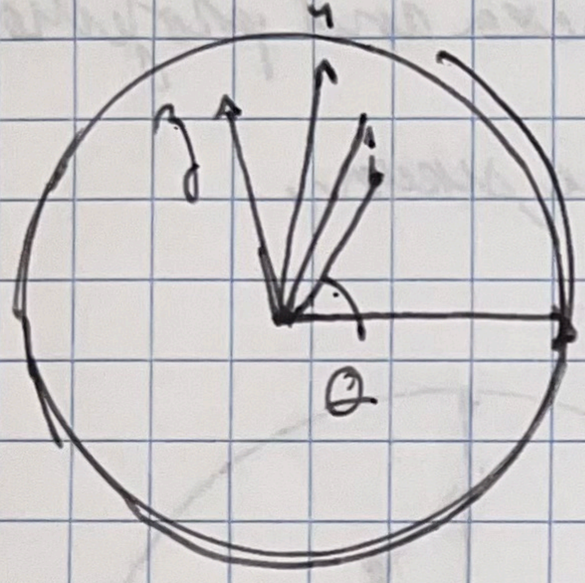
-

Okay so I'm a fucking idiot. $\arctan(\frac{x}{y})$ is undefined at $y=0$ which is like... bad. Let's think physically here...

$$\begin{pmatrix} r \\ \theta \\ \varphi \end{pmatrix} = \begin{pmatrix} \sqrt{x^2+y^2+z^2} \\ \arctan(y/x) \\ \arctan\left(\frac{\sqrt{x^2+y^2}}{z}\right) \end{pmatrix}$$

- This is chill over all space
- When $x=0$, $\theta = \frac{\pi}{2} \cdot \frac{y}{|y|}$
- When $y=0$, $\varphi = \frac{\pi}{2}$

-



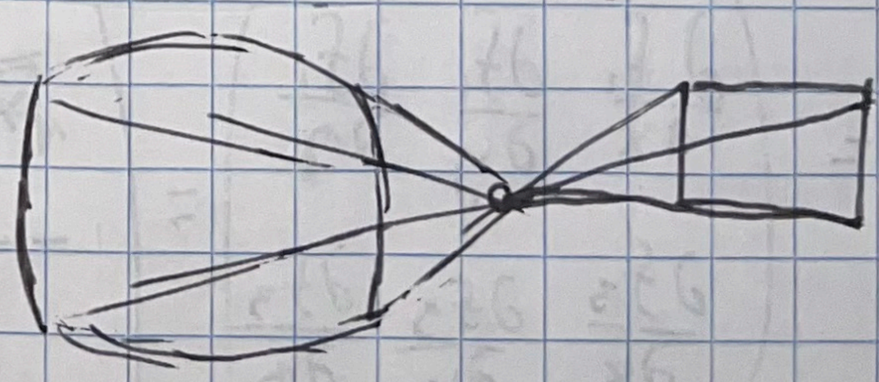
I don't think there's a better way than conditionals. Should be fairly easy.

-

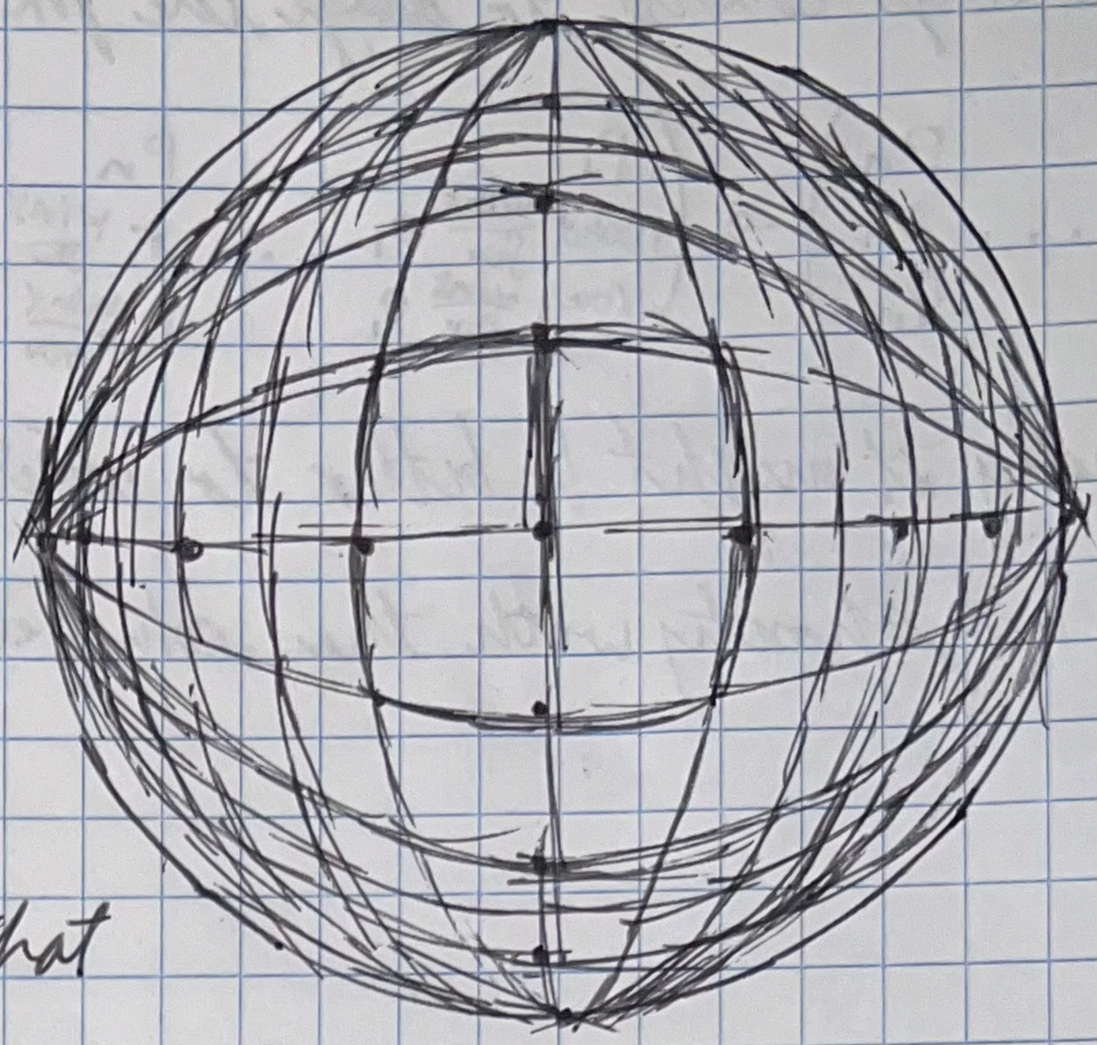
Apparently, Cmath has a function to deal with that automatically. std::atan2

-

Okay so now I've gotta think about how the projection will work. Given an $n \times m$ grid of pixels, how do I project the points? Each pixel will correspond to a certain range. The question is should each correspond to an angle Range or a more "cartesian" rectangular range?



I think either way will result in some warping. The spherical option, probably a fish-eye while the rectangular will stretch the edges. The spherical seems easier to do so-idk they that first.

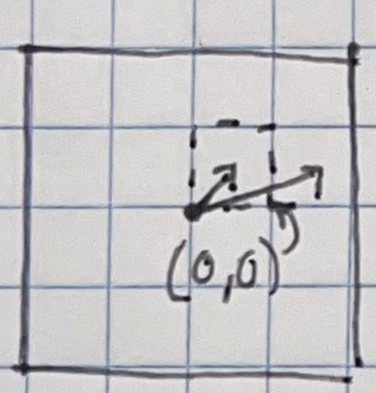


How? I start with fov, camDir, winDim, and a vector $\vec{v}=(r,\theta,\varphi)$

With $camDir=(\theta_1,\varphi_1)$ and $\vec{v}'=(\theta_2,\varphi_2)$, $s=(\theta_2-\theta_1,\varphi_2-\varphi_1)$

Anything with $s_x<fov/2$ and $s_y<\frac{fov}{winDimX}\cdot winDimY$ is rendered

Consider a special case as an example: Let $camDir=(0,0)$, $\vec{v}'=(0.5,0.5)=s$ $fov=4$ and $winDim=(4,4)$. I shouldn't have to look through every square to figure out which has (1,1).



$\frac{fov}{winX}(0)\le\theta_1=0.5\le\frac{fov}{winX}(1)$, $\frac{fov}{winX}(1)<\theta_2=1.5<\frac{fov}{winX}(2)$

$n\le\frac{winX}{fov}\theta\le n+1$ $\quad K\le\frac{winY}{fov}\varphi<K+1$

My function implementation is atrocious here. I need to apply basically $\text{Round}\left(\frac{\text{int}\to\text{doub winX}}{\text{doub fov}}\cdot\text{shape.polarVerts}()(i,j)\right)$ for int $i\in(1,2)$ and int $j\in(1,\text{shape.polarVerts}().\text{row}(1).\text{size}())$ but this is awful and I hate it.

Maybe I go colwise to dodge the for loop? Oh maybe write a component-wise function first and apply that?

$$\begin{pmatrix} \rho_1 & \cdots & \rho_n \\ \theta_1 & \cdots & \theta_n \\ \varphi_1 & & \varphi_n \end{pmatrix} \to \begin{pmatrix} \rho_1 & & \rho_n \\ \text{round}\,\frac{\text{winX}}{\text{fov}}\,\theta_1 & \cdots & r\,\frac{\text{winX}}{\text{fov}}\,\theta_n \\ \text{round}\,\frac{\text{winX}}{\text{fov}}\,\varphi_1 & & r\,\frac{\text{winX}}{\text{fov}}\,\varphi_n \end{pmatrix}$$

Actually it might be better to write it vector-wise so I can just hit shorty with the .colwise().