Lecture 23:
Before Fall Break Loose Ends

November 21, 2019

Five Topics Today

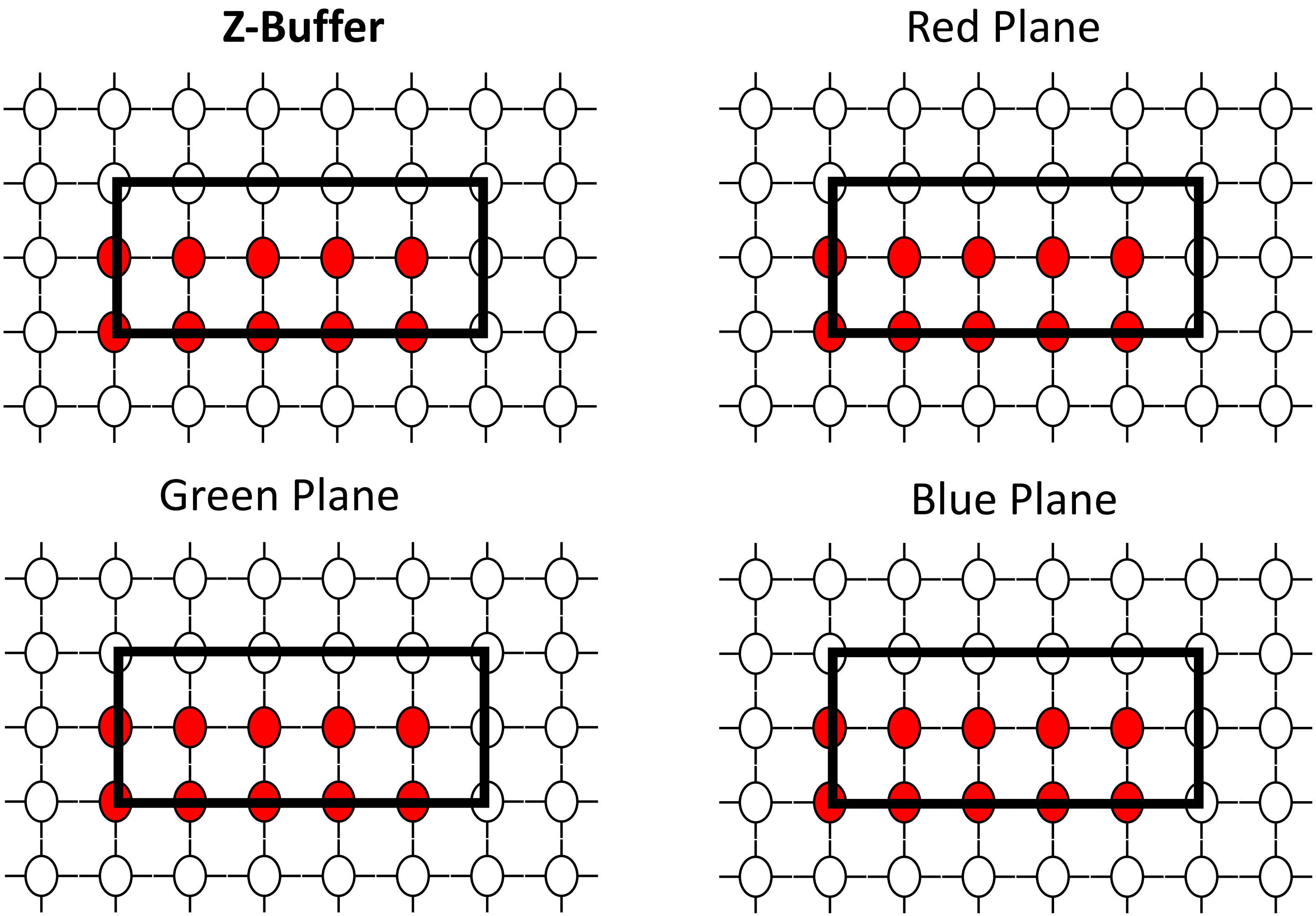
- Reflections on Debugging in CS 410
- Z-buffers and psydo-depth
- Thin Lens Modeling Kept Simple
- Six Degree of Freedom Mapping
- Overview Programming Assignment 5

CS 410 and Debugging

- About Debuggers
  - Need them for segmentation fault line numbers
  - Otherwise, a mixed blessing (too much info.)
- Small scale testing
  - Render something simple!
  - Instrument your code (means print statements)
  - Compute it two ways
- Bigger more complex issues
  - Spreadsheets can be very useful!

Projection Pipeline and Depth

Render this Rectangle



What colors and when.

Using a Z-Buffer

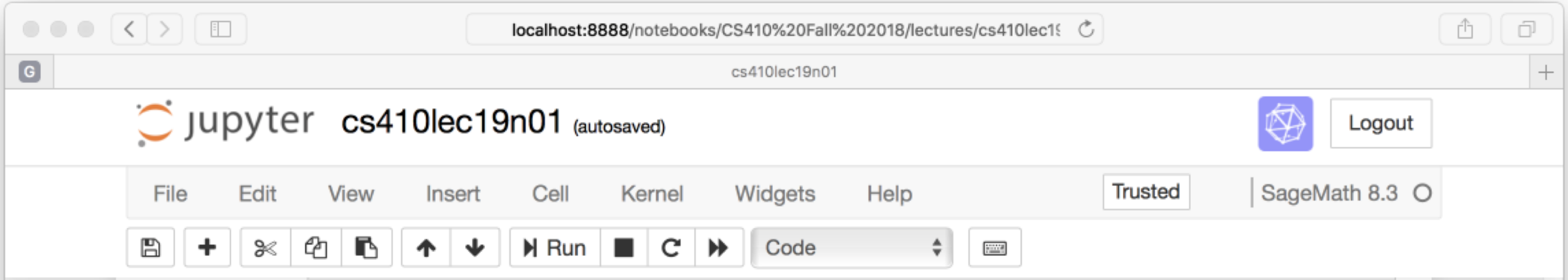
- Record depth at every vertex
- For every pixel in polygon (previous lecture)
  - – Interpolate to get depth at specific pixel.
  - – Is depth less then currently recorded?
    - Yes: Record in Z-Buffer and paint pixel
    - No: Move along, nothing to do here
- “Paint” is shorthand for compute the surface illumination for that position on the polygon.

About depth: the z-value

- Z-buffering based upon pseudo-depth is key to modern polygon rendering.
- Depth already revealed in SageMath notebook on the Canonical View Volume.
- Here let us briefly dive into the calculation of pseudo-depth using essentially that example.

SageMath Notebook

- Emphasize the z coordinate of transform



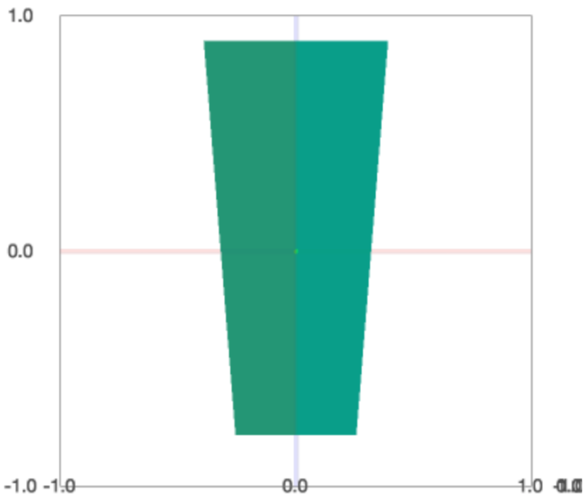
$$\left(\begin{array}{c} -\frac{umax+umin}{umax-umin} \\ -\frac{vmax+vmin}{vmax-vmin} \\ \frac{\frac{2\,farnear}{far-near}-\frac{(far+near)z}{far-near}}{z} \\ 1 \end{array}\right) = \left(\begin{array}{cccc} \frac{2\,near}{umax-umin} & 0 & -\frac{umax+umin}{umax-umin} & 0 \\ 0 & \frac{2\,near}{vmax-vmin} & -\frac{vmax+vmin}{vmax-vmin} & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & \frac{2\,farnear}{far-near} \\ 0 & 0 & 1 & 0 \end{array}\right) \left(\begin{array}{c} 0 \\ 0 \\ z \\ 1 \end{array}\right)$$

$$P_{cc} = \left(\begin{array}{c} -\frac{umax+umin}{umax-umin} \\ -\frac{vmax+vmin}{vmax-vmin} \\ \frac{\frac{2\,farnear}{far-near}-\frac{(far+near)z}{far-near}}{z} \\ 1 \end{array}\right) \text{ and the z term only } pz = \frac{\frac{2\,farnear}{far-near}-\frac{(far+near)z}{far-near}}{z}$$

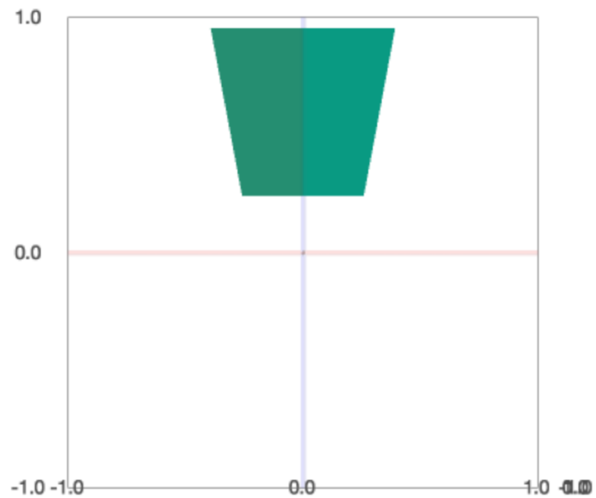
```
In [68]: if (case != 'sym') :
```

First the Symptom

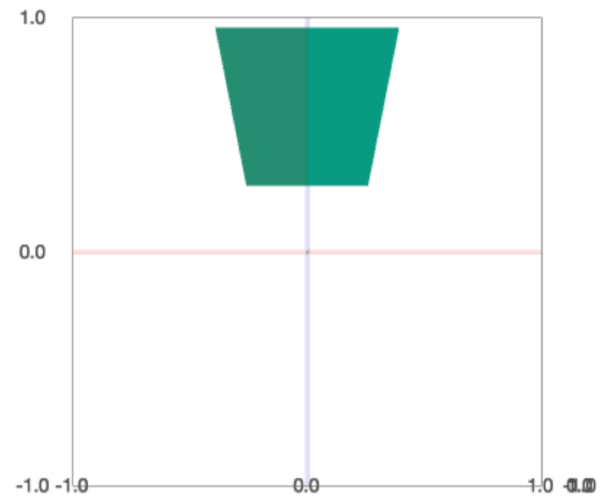
Near = -25
Far = -75



Near = -25
Far = -750



Near = -25
Far = -7500



Remember, the house lies between z of 30 and 54 in world coordinates.

Even pushing the far clipping plane 2 orders of magnitude further back from -75 still results in the house occupying most of the pseudo-depth range between 0 and 1.

Back to the Math

- Camera at origin no world cam. rotation

$$\begin{vmatrix} -\dfrac{(umax+umin)z}{umax-umin} \\ -\dfrac{(vmax+vmin)z}{vmax-vmin} \\ \dfrac{2\,farnear}{far-near} - \dfrac{(far+near)z}{far-near} \\ z \end{vmatrix} = \begin{vmatrix} \dfrac{2\,near}{umax-umin} & 0 & -\dfrac{umax+umin}{umax-umin} & 0 \\ 0 & \dfrac{2\,near}{vmax-vmin} & -\dfrac{vmax+vmin}{vmax-vmin} & 0 \\ 0 & 0 & -\dfrac{far+near}{far-near} & \dfrac{2\,farnear}{far-near} \\ 0 & 0 & 1 & 0 \end{vmatrix} \begin{vmatrix} 0 \\ 0 \\ z \\ 1 \end{vmatrix}$$

$$P_{cc} = \begin{vmatrix} -\dfrac{umax+umin}{umax-umin} \\ -\dfrac{vmax+vmin}{vmax-vmin} \\ \dfrac{\dfrac{2\,farnear}{far-near} - \dfrac{(far+near)z}{far-near}}{z} \\ 1 \end{vmatrix}$$

and the z term only $pz = \dfrac{\dfrac{2\,farnear}{far-near} - \dfrac{(far+near)z}{far-near}}{z}$

Pseudo-depth

pz At near and far

- Equation: $$pz = \frac{2 * far * near}{(far\ - near) * z} - \frac{(far + near)}{(far - near)}$$

Let z equal $near$

$$pz = \frac{2 * far * near}{(far\ - near) * near} - \frac{(far + near)}{(far - near)}$$

$$pz = \frac{2 * far - far - near}{(far\ - near)}$$

$$pz = \frac{far - near}{(far\ - near)}$$

$$pz = 1$$

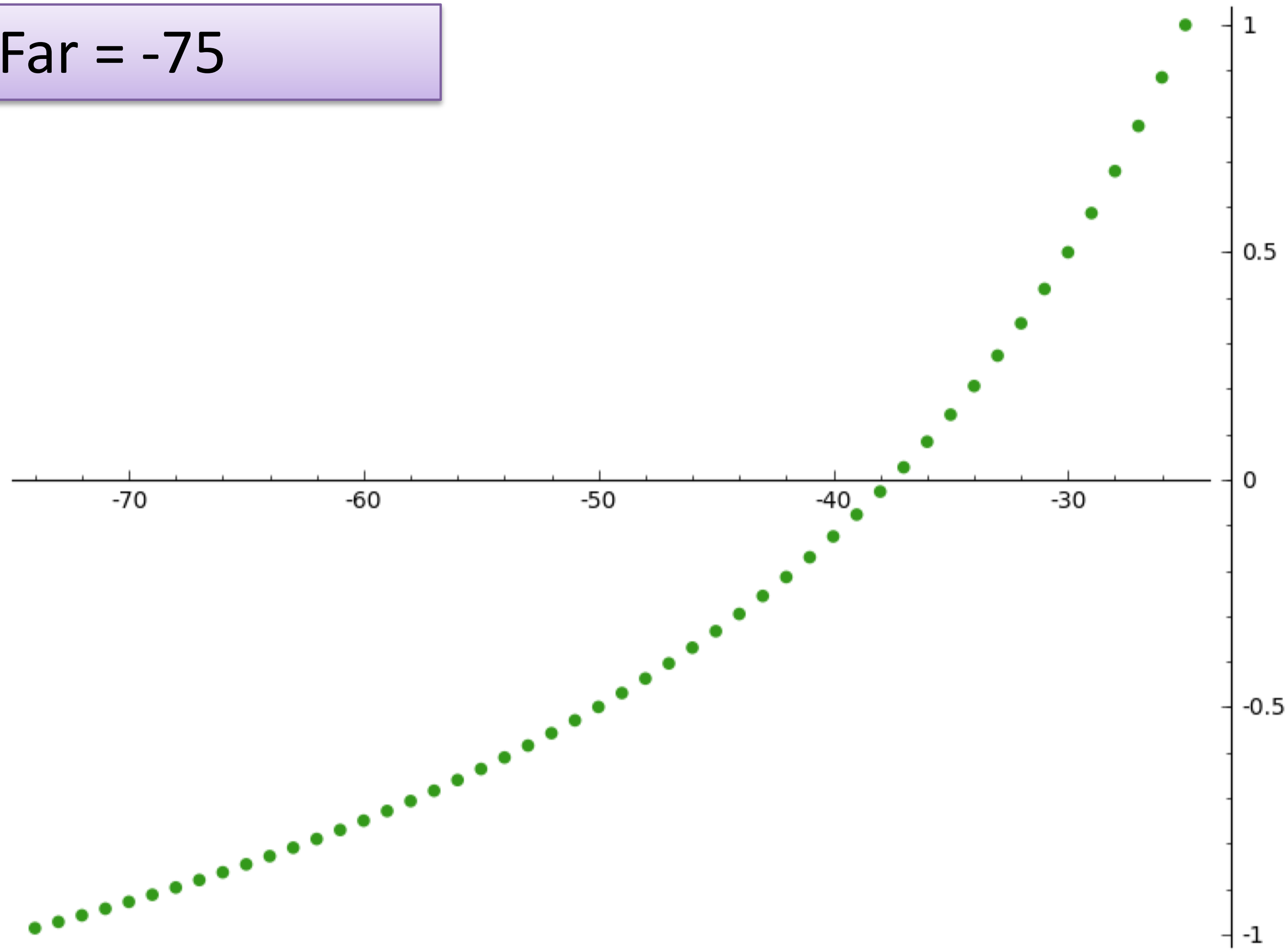
Similarly …

Let z equal far

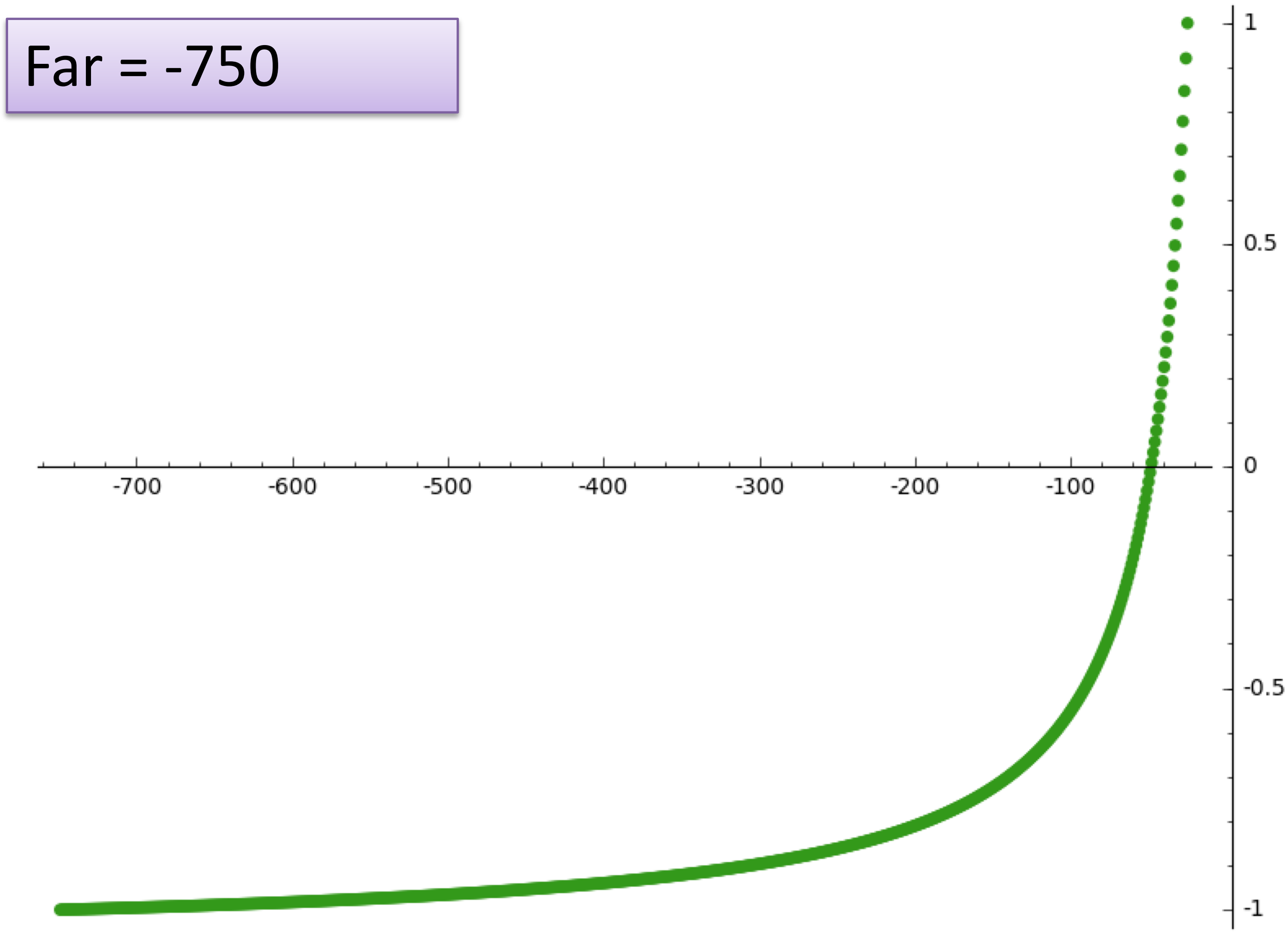
$$pz = -1$$

Plot actual Depth to Pseudo-depth

Far = -75

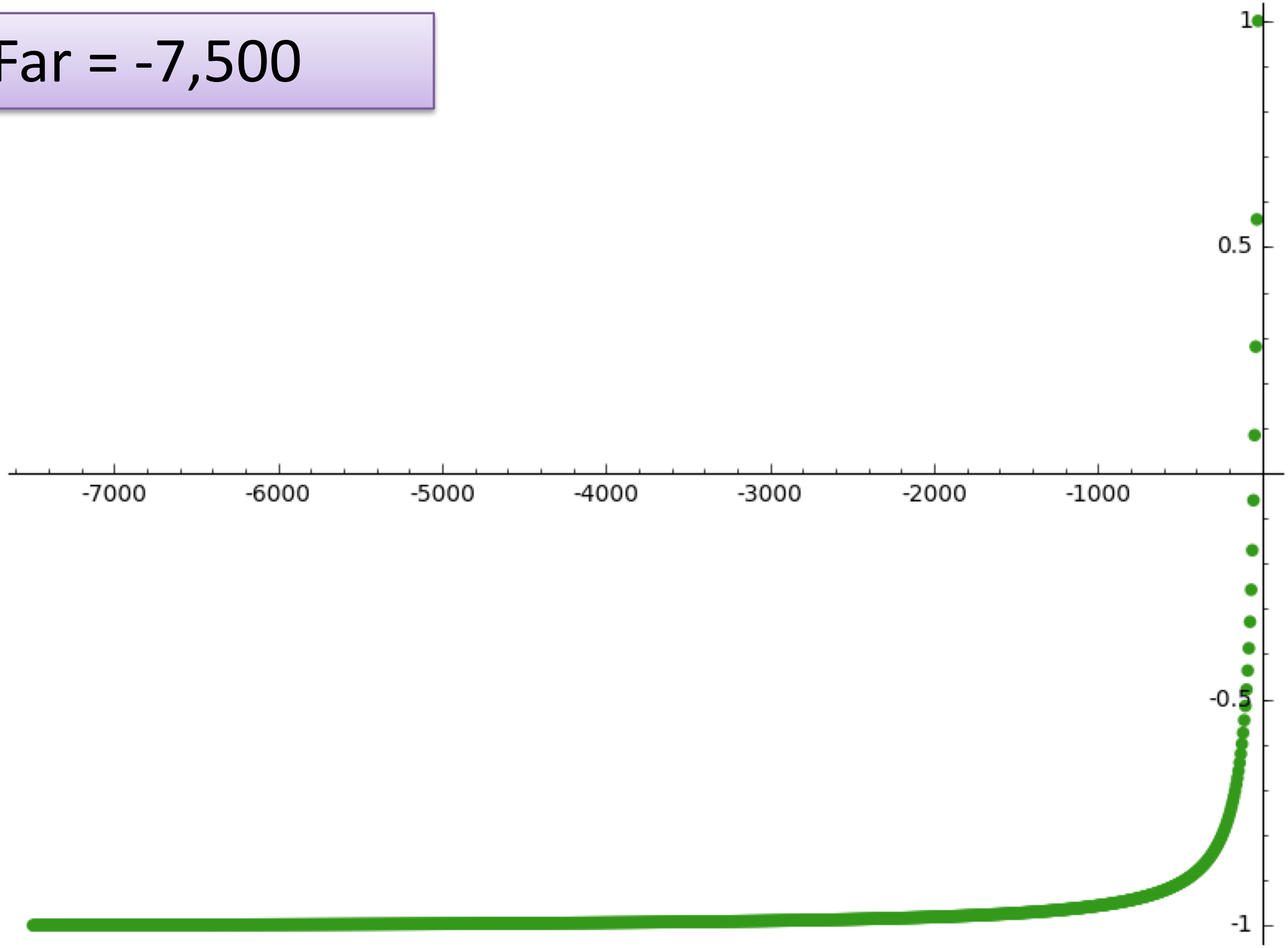


Plot actual Depth to Pseudo-depth

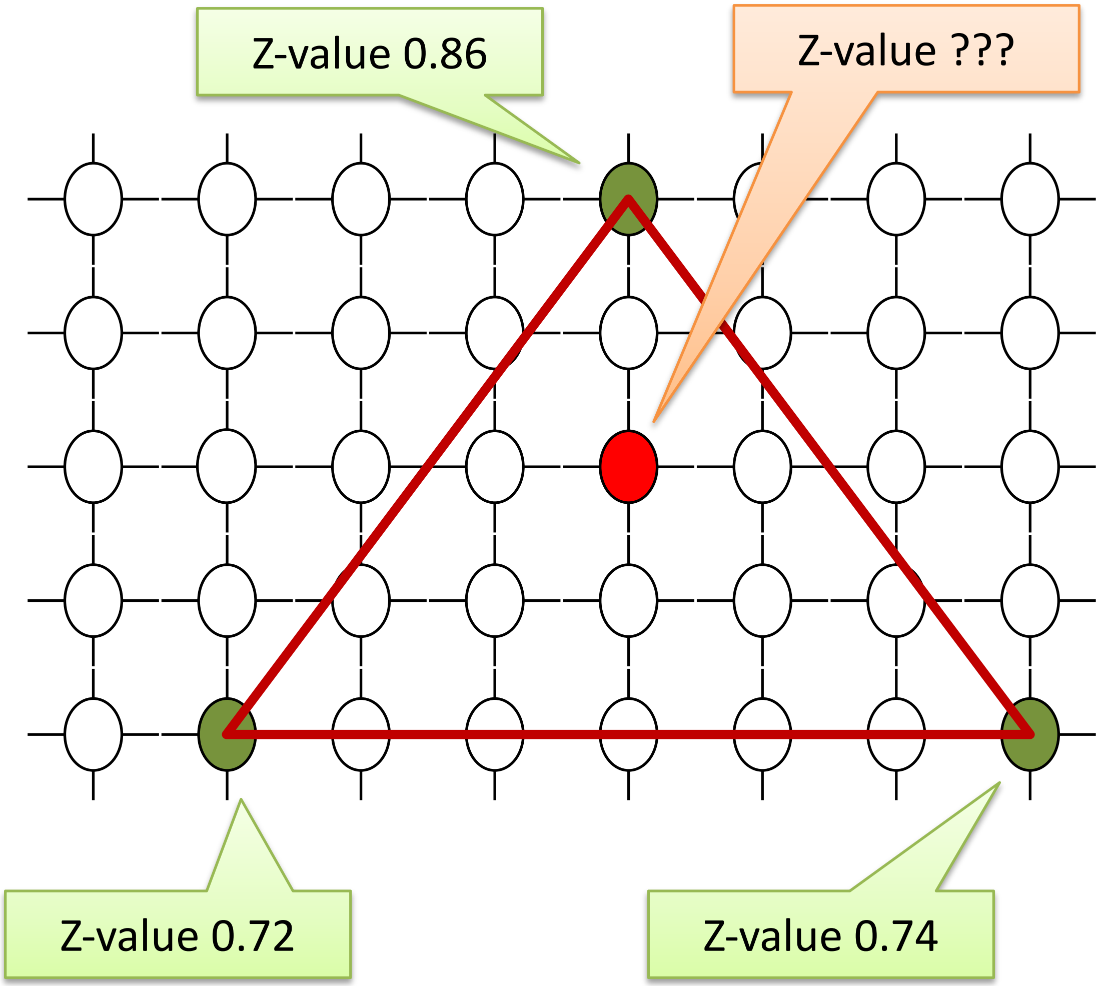


Plot actual Depth to Pseudo-depth

Far = -7,500



Interpolate Z-value



There are various ways to interpolate in order to arrive at an estimated z-value for a interior point on any given triangle.

Common is to first interpolate up the sides and then to interpolate across.

Z-Buffer Summary

- A Z-buffer is an array of doubles
- Size of the frame buffer / image
- Initialized to -1.0, i.e. far clipping plane
- Now consider a specific triangle
- For each pixel to be filled
  - Interpolate pixels z-value
  - Test if larger then what is in the Z-buffer
  - If yes then “paint” that pixel for that triangle

What if you Want Depth?

- Mapping may be inverted.

$$pz = \frac{2 * far * near}{(far\ - near) * z} - \frac{(far + near)}{(far - near)}$$

$$z = \frac{2 * far * near}{(far\ - near) * pz + far + near}$$

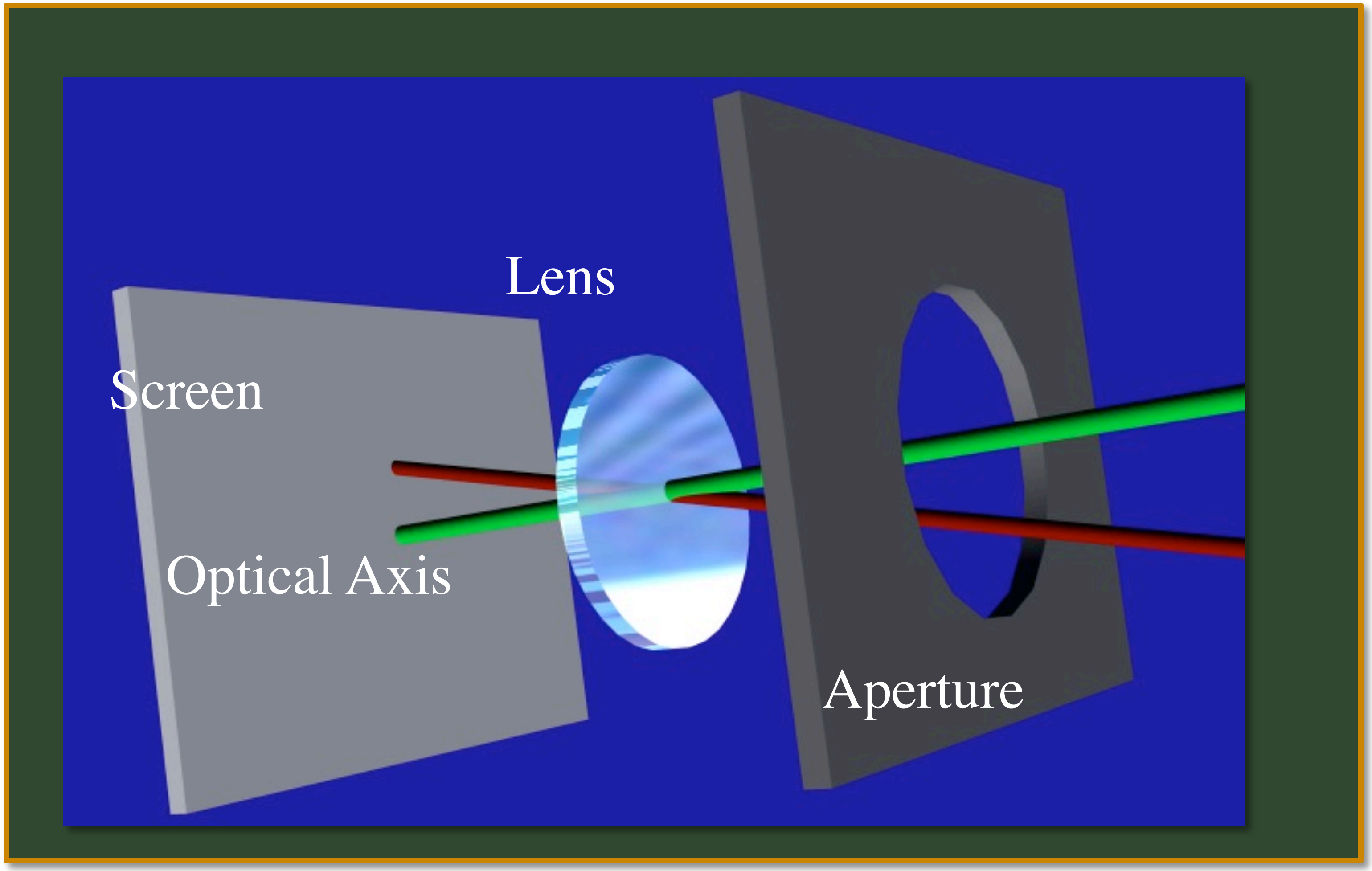
```
In [1]:  var('y','near','far','z')
         eq = y == (2*far*near)/((far-near)*z) - (far + near)/(far - near)

In [2]:  eq
Out[2]:  y == -(far + near)/(far - near) + 2*far*near/((far - near)*z)

In [3]:  solve(eq,z)
Out[3]:  [z == 2*far*near/((far - near)*y + far + near)]
```

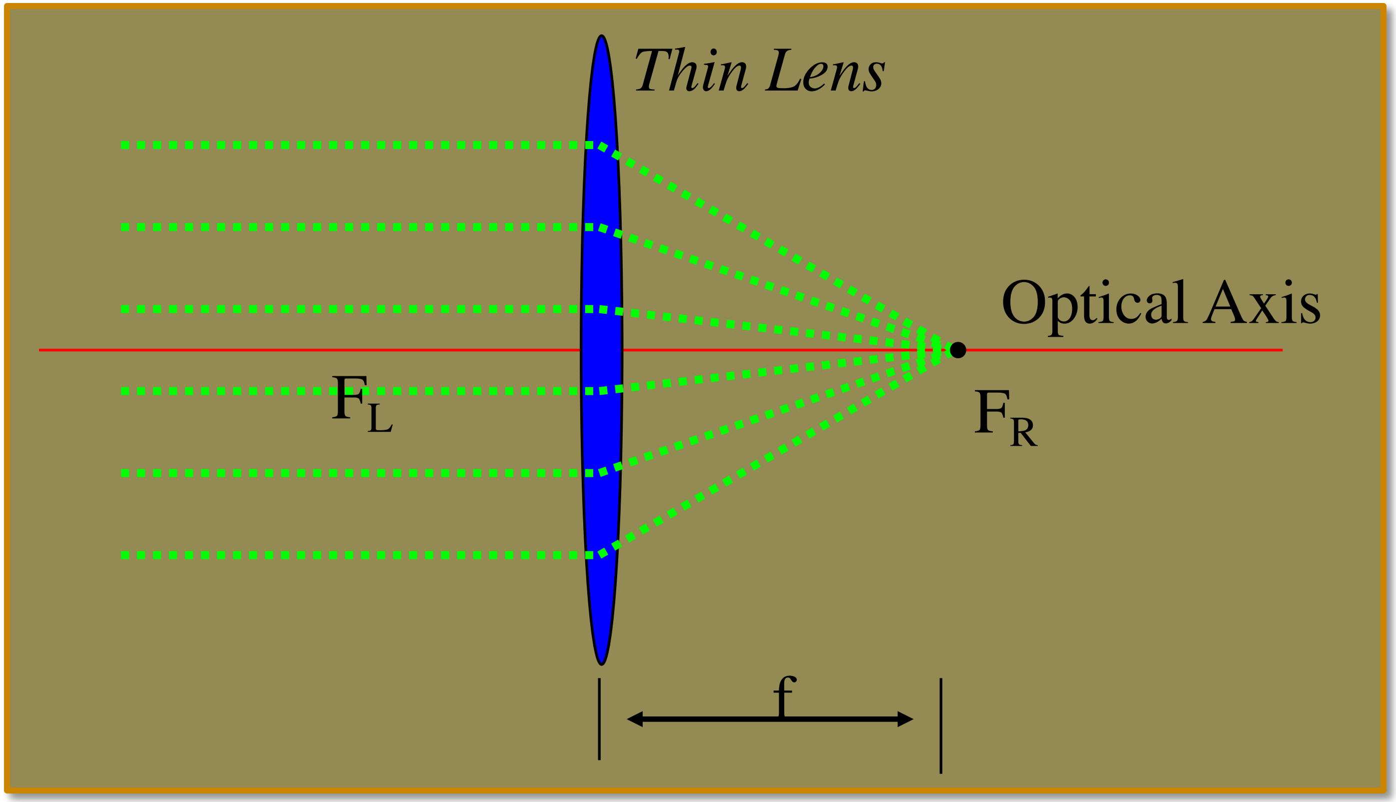
There are worse things then checking your work in a symbolic math package.

Thin Lens Modeling



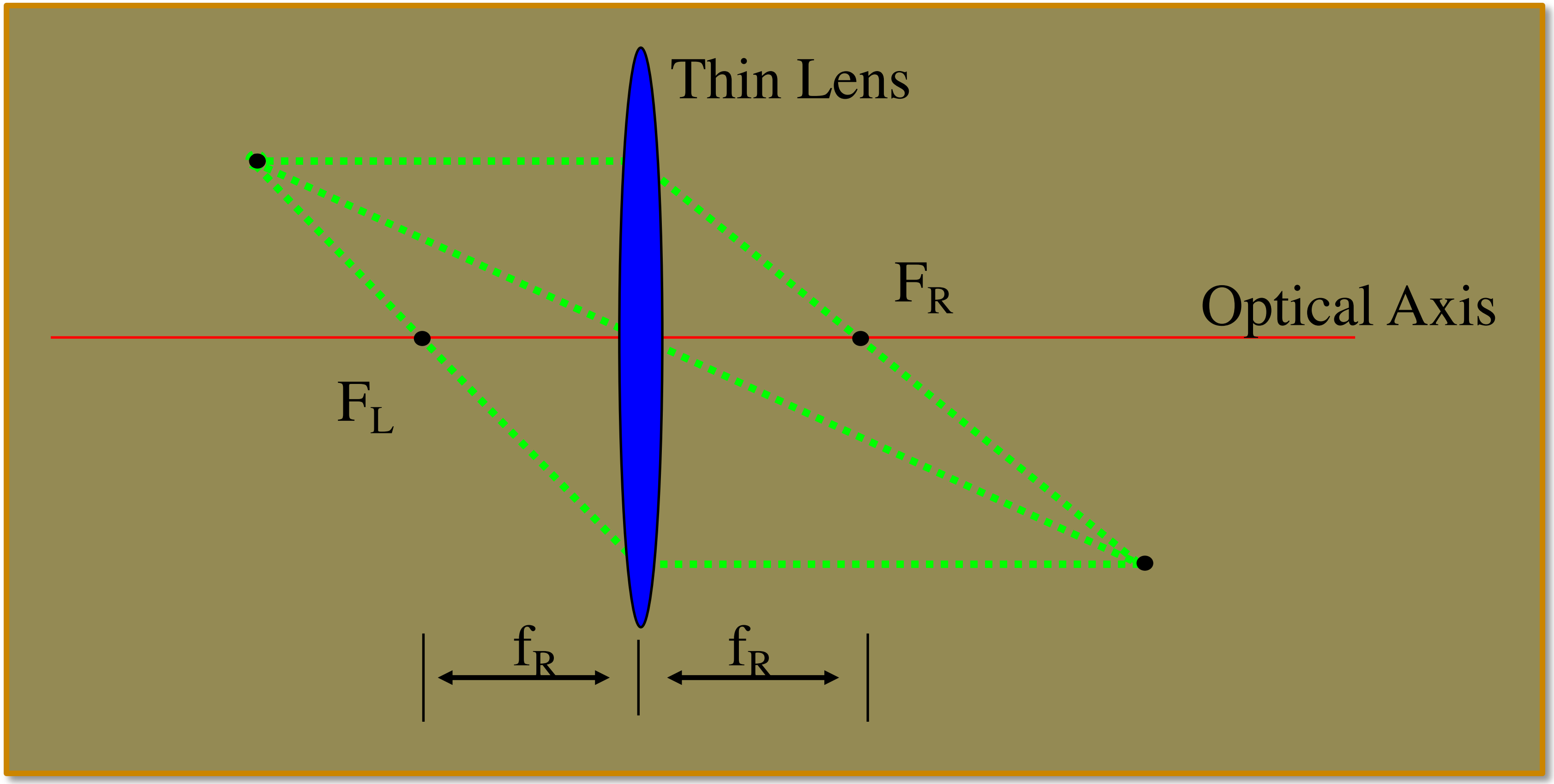
Thin Lens Model

- Parallel rays on one side converge at focal point on the other side.
- Rays diverging from the focal point become parallel.



Thin Lens Model

- Thus many paths join together.



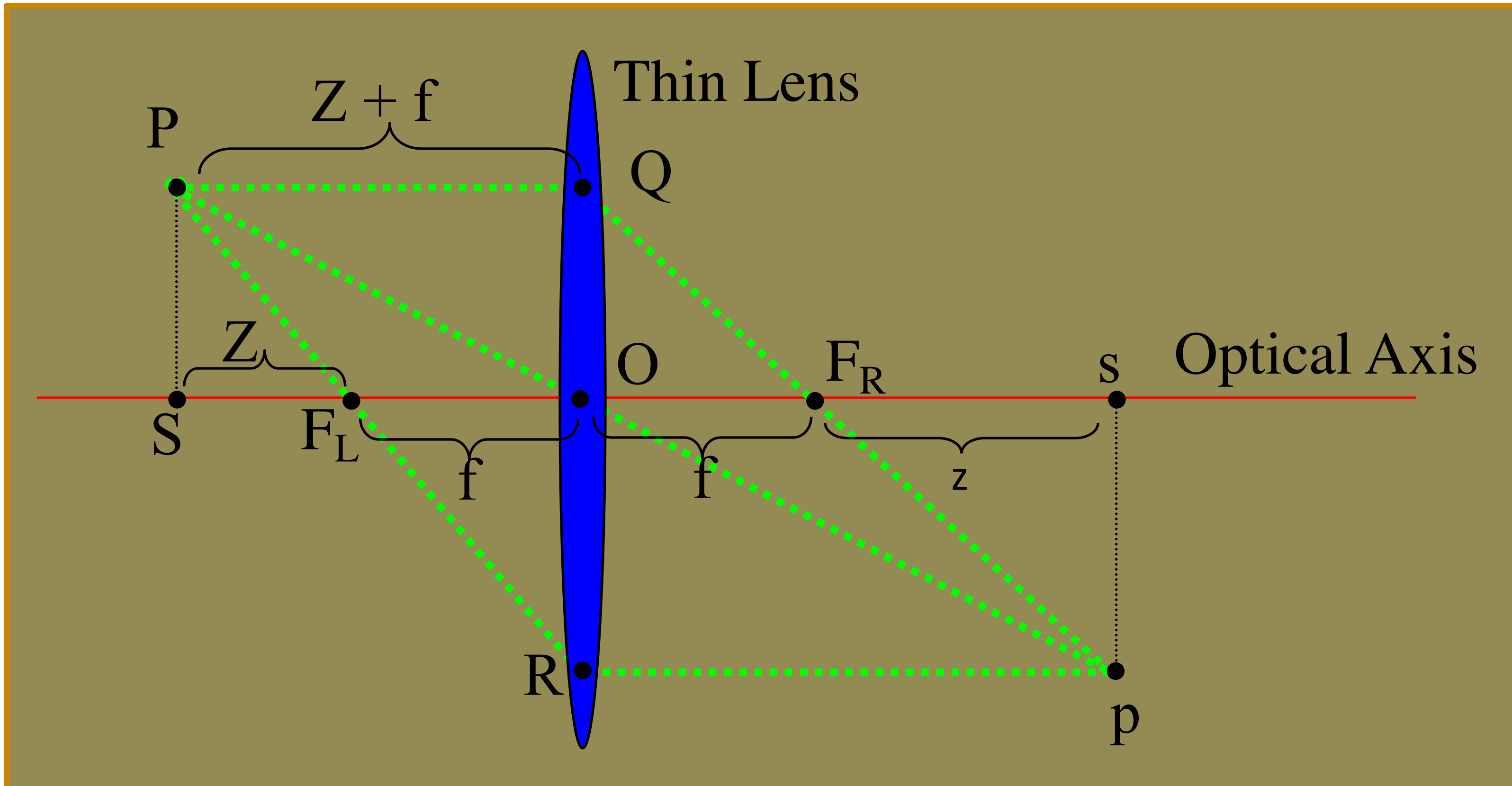
Thin Lens Constraints

#1 All rays emanating from a single point in space must converge on a single point in the image plane (definition of focus)

#2 Any ray entering the lens parallel to the axis on one side goes through the focus point on the other side

#3 Any ray entering the lens from the focus point on one side emerges parallel to the axis on the other side

Fundamental Equation of Thin Lenses

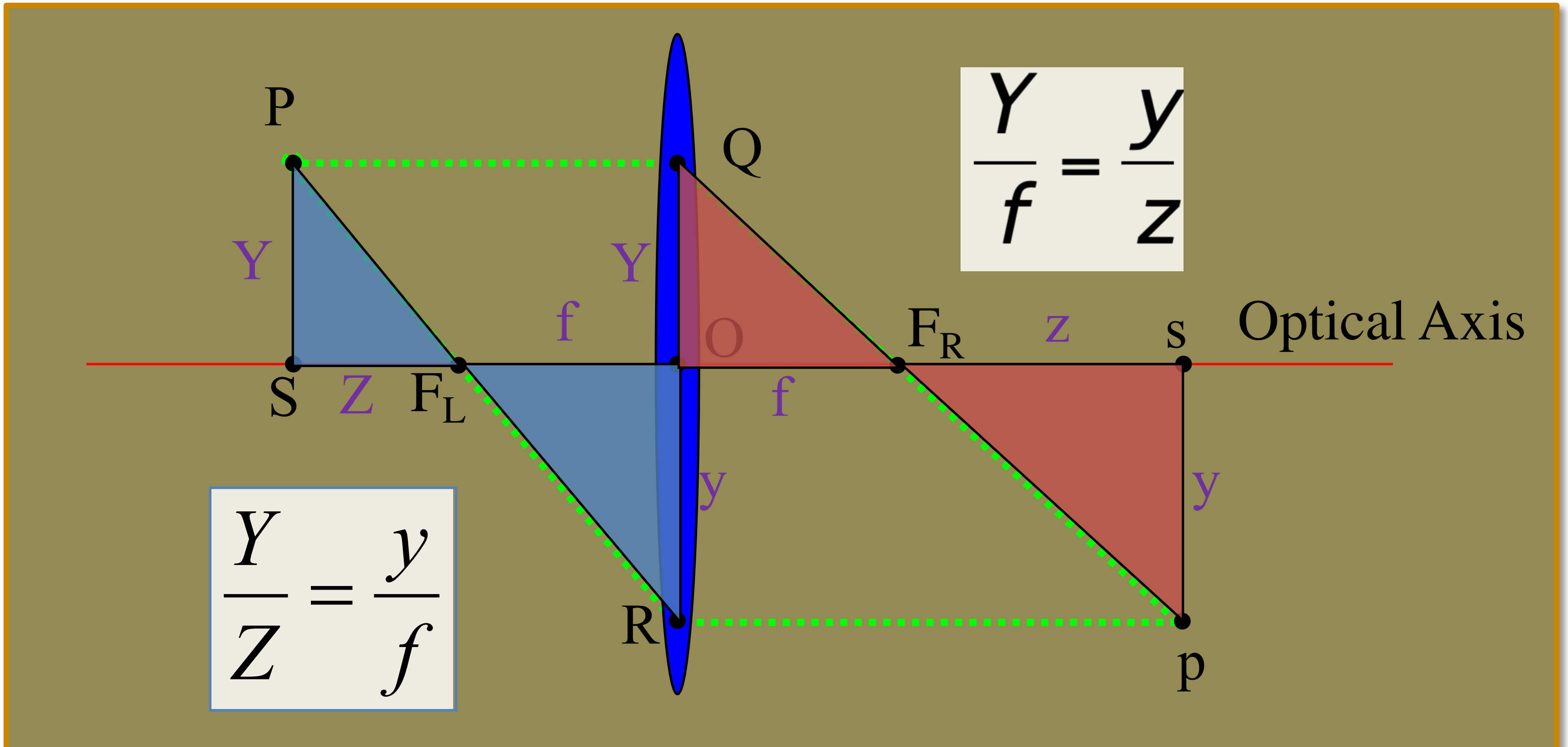


Note: P is "not too far" from optical axis

Fundamental Equation (II)

- The ray PQ (parallel to the optical axis) must be deflected to pass through FR by property #2
- The ray PR must be deflected so that it becomes parallel to the optical axis by property #3
- After deflection, PQ & PR must intersect at p, by property #1.
- Now, use similar triangles….

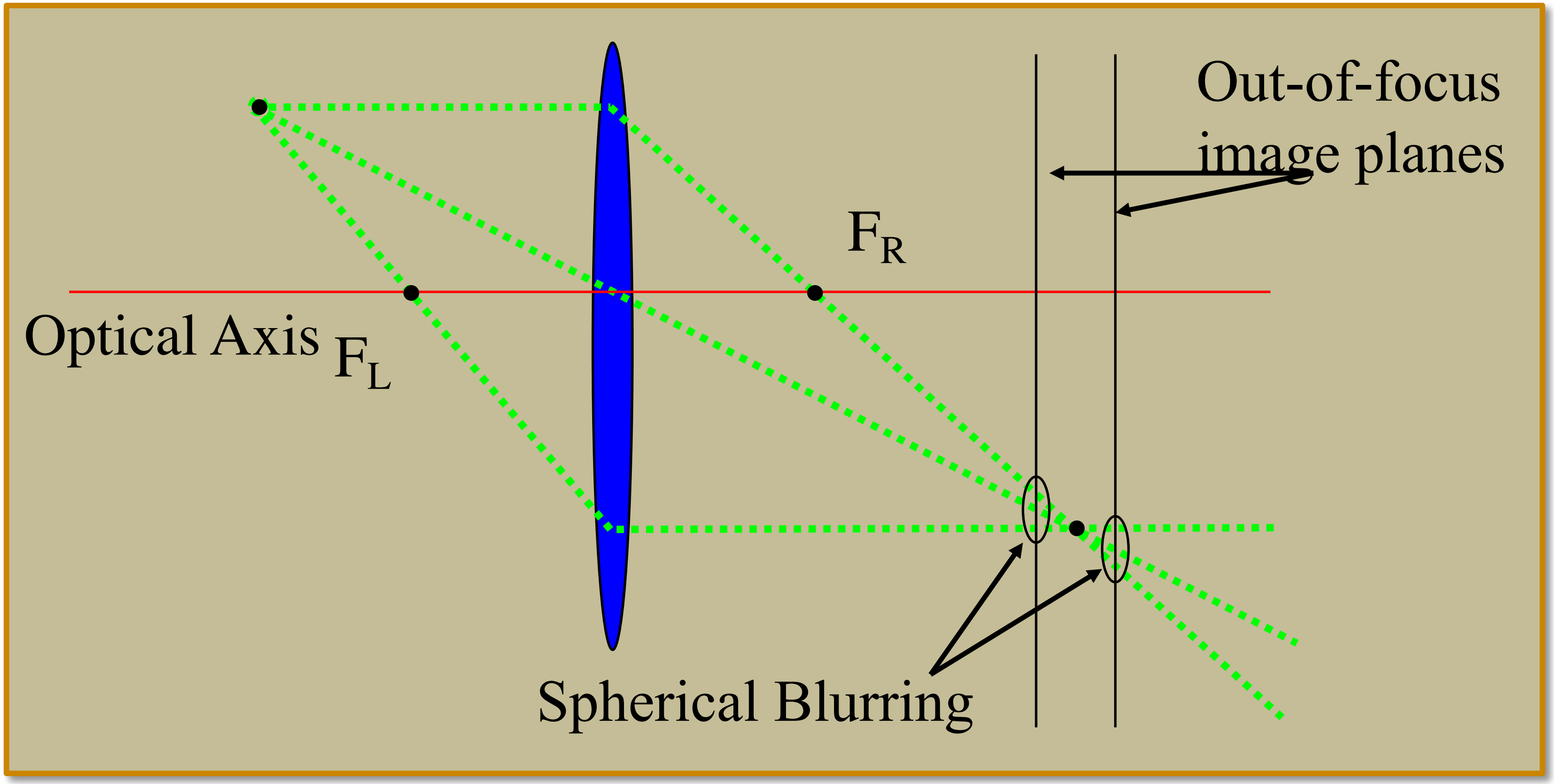
Fundamental Eq. (III)



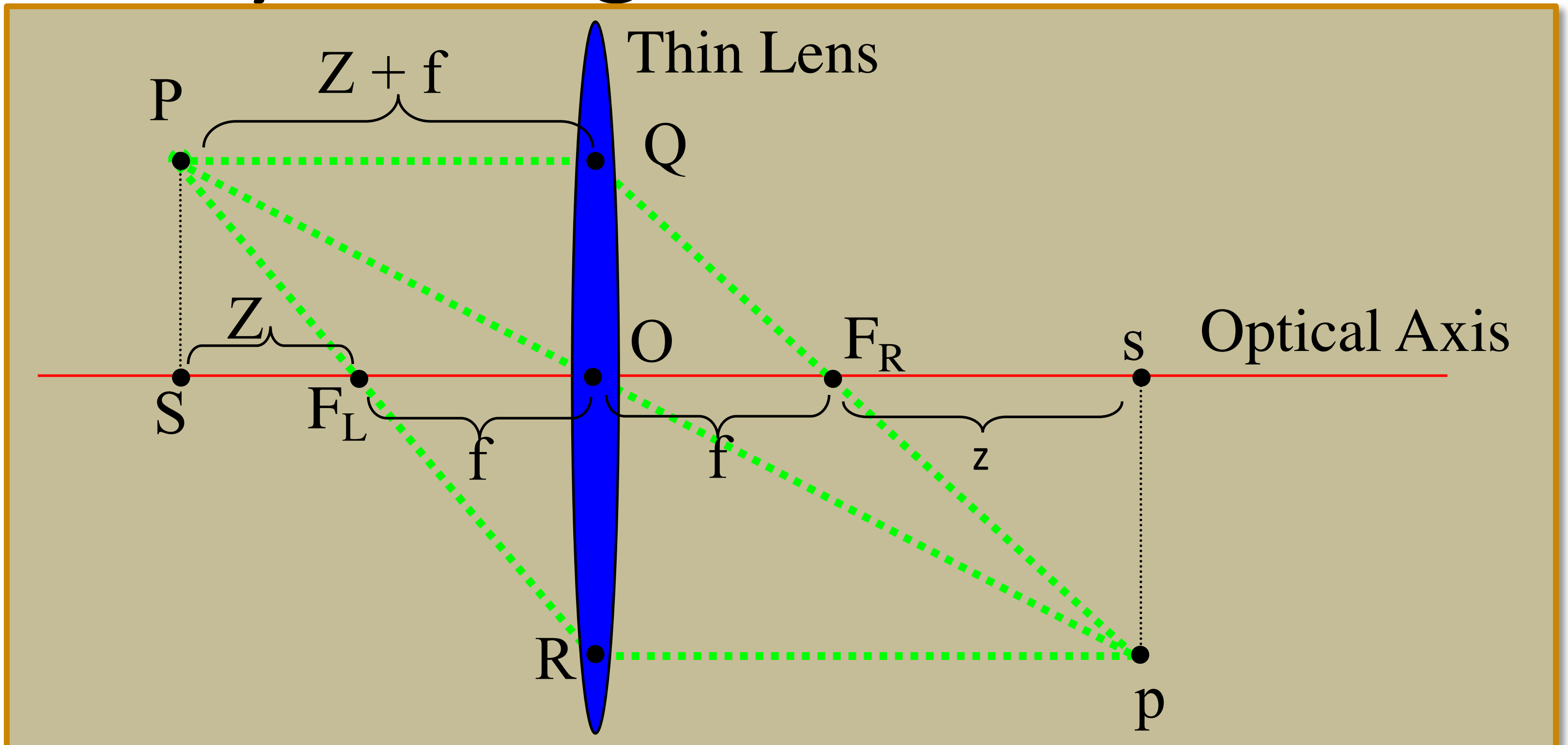
Substitute for y and solve: $f^2 = zZ$, *or* $\frac{1}{Z+f} + \frac{1}{z+f} = \frac{1}{f}$

Out of Focus Images

What happens when $\frac{1}{Z+f} + \frac{1}{z+f} \neq \frac{1}{f}$



Ray Tracing with a Thin Lens



Same picture as before – new question:

Given a point p in the camera (a pixel), where in the world is P such that all rays through the lens from P to p focus perfectly?

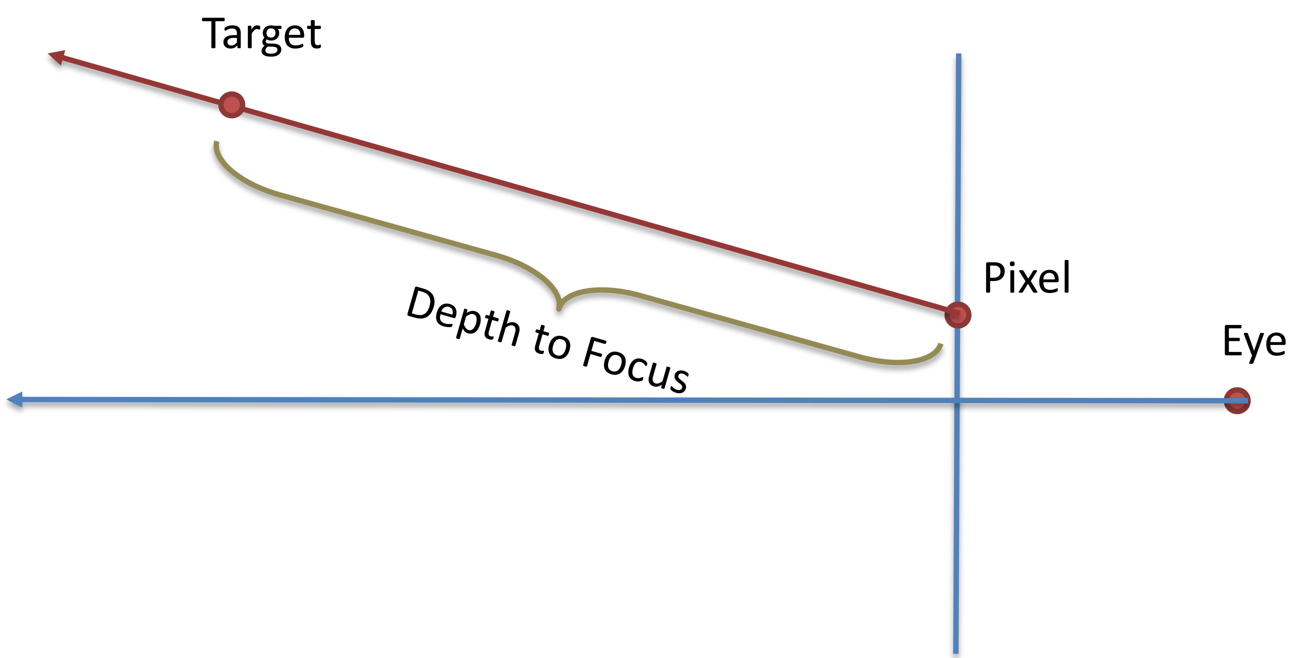
Point of Focus

- The position of P = (X, Y, Z) can [illegible] calculated by finding the intersection of [illegible]ays that converge at (a, b, z)
- One ray goes throu[illegible] left focal point, strikes the lens [illegible]b, L), and proceeds parallel to th[illegible]ic axis to (a, b, z)
- One ra[illegible]s from (X, Y, Z) parallel to the opti[illegible]s until is strikes the lens, and is then [illegible]cted through the right focal point to (a, b, [illegible])

No, we will simplify

Keep It Simple

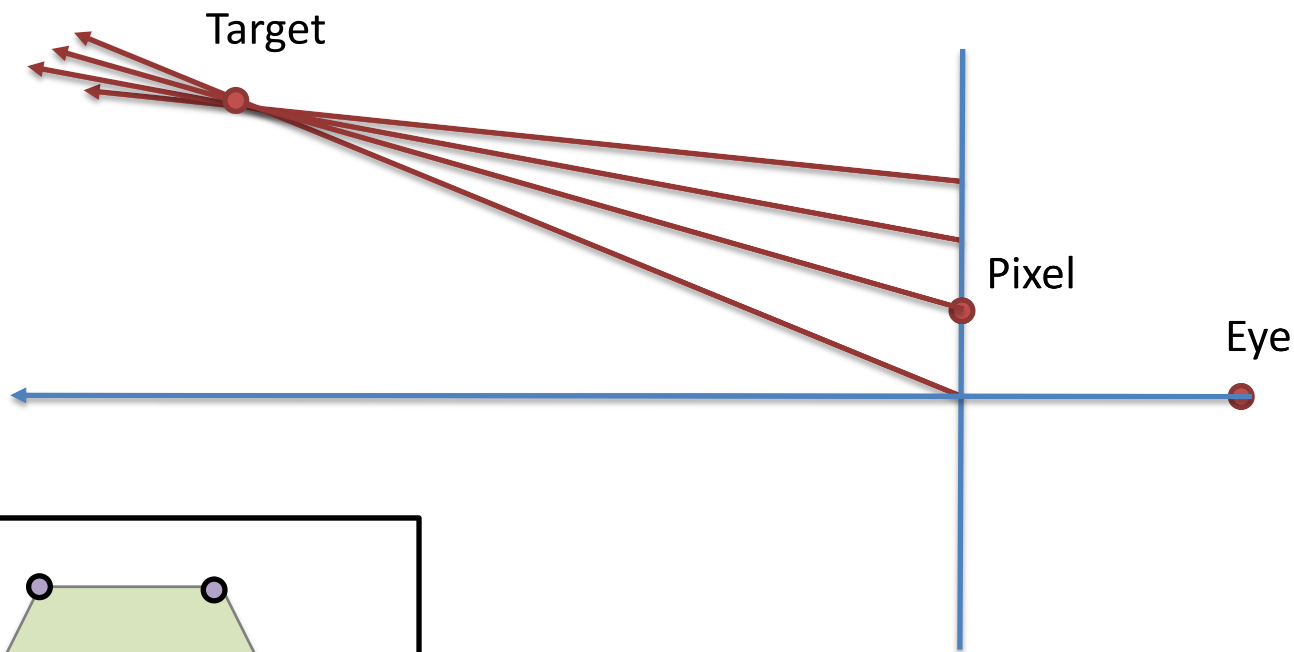
- Arbitrarily set a depth for perfect focus.

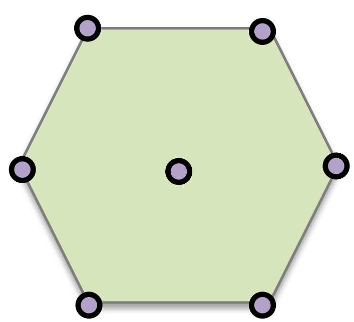


Compute the 3D target point by going an amount tau along the ray from the pixel. Instead of firing one ray from the pixel 3D position to the target, fire k from slightly different pixel coordinates sampled around the true position.

Keep It Simple

- Showing a couple of sample rays.





Example Sample Pattern

The size of the sample pattern is exaggerated above.

Review 6 DOF from Lec 22

localhost:8888/notebooks/CS410%20Fall2019/lectures/cs

jupyter cs410lec22n01 Last Checkpoint: 5 hours ago (unsaved changes)

Texture Coordinates Solve Transform

This is a quick workup of solving for texture coordinates given three pairings. In the next equation the six degree of freedom transformation mapping from a 3D coordinate on a survace and a 2D coordinate in a texture map is expressed using six free variables indicated 'a' through 'f'.

Ross Beveridge, November 21, 2019

In [1]:

```
var('a', 'b', 'c', 'd', 'e', 'f')
var('x', 'y', 'z')
var('u','v')
TM = Matrix(SR, 2,3, ((a,b,c),(d,e,f)))
UV = Matrix(SR, 2,1, ((u),(v)))
PT = Matrix(SR, 3,1, ((x),(y),(z)))
pretty_print(UV, LatexExpr(" = "), TM, PT)
```

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$