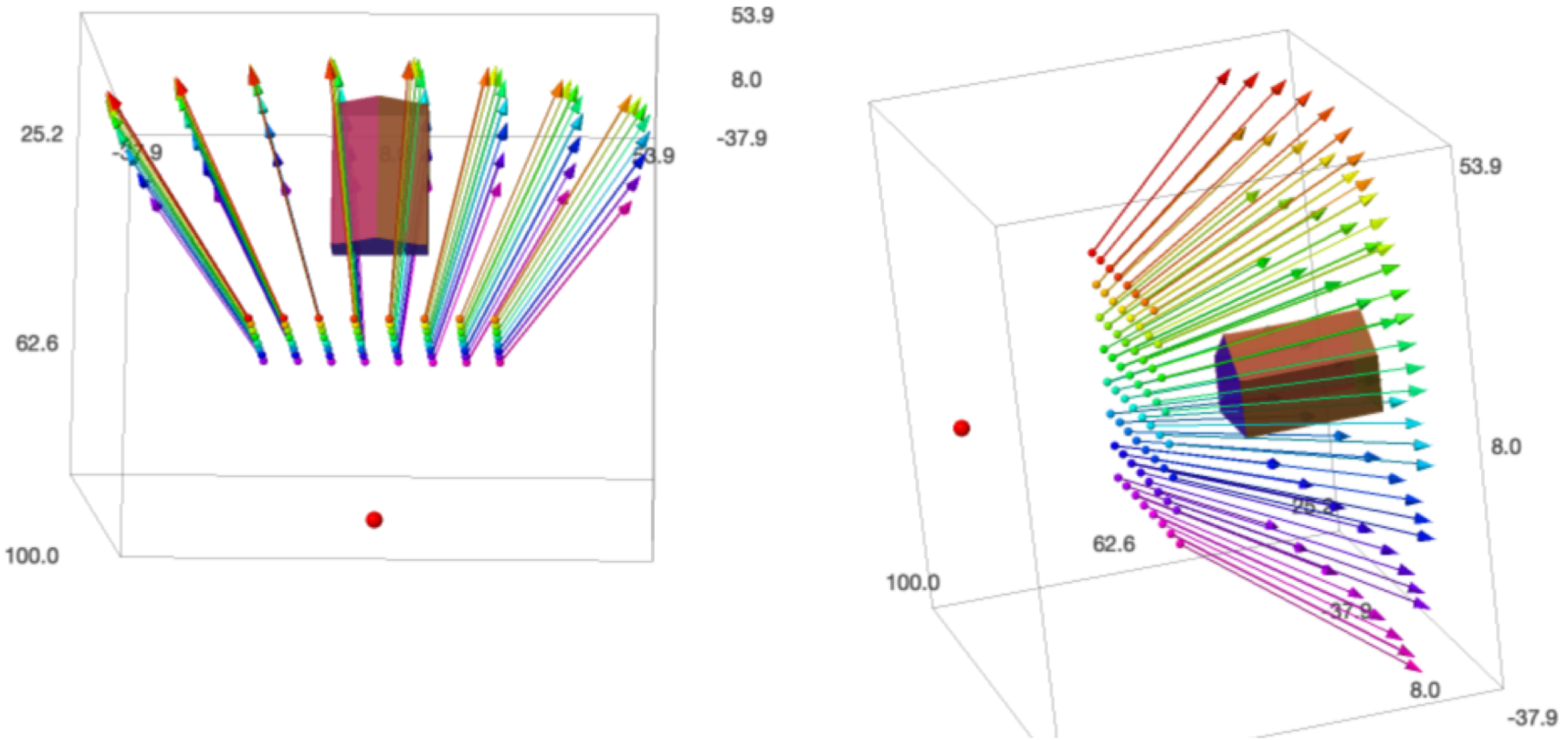
Lecture 7:
Rays and Spheres

September 17, 2019

To Start – Shooting Rays



Goal:

Be able to code ray generation from scratch using only a camera specification and explain that code in linear algebraic terms!

This Material is Covered In

jupyter cs410lec07n01 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted SageMath 8.8

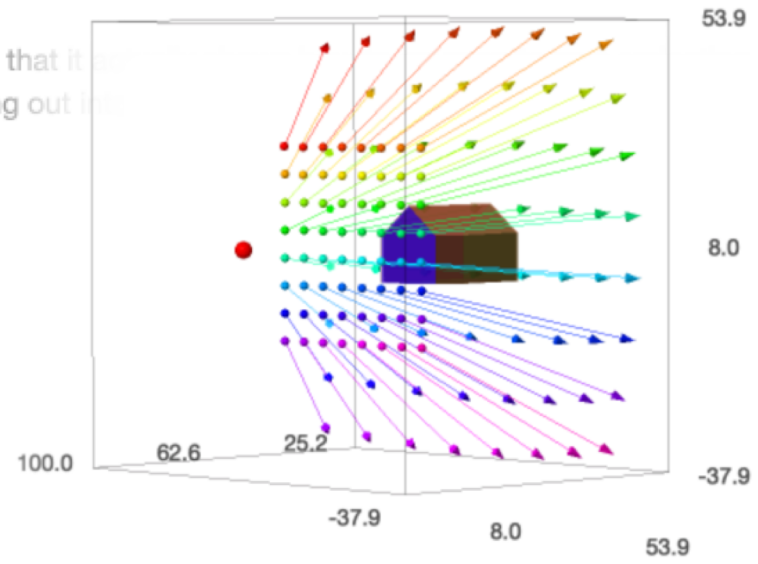
Run Code

Shooting Rays into a Scene

This notebook shares much in common with the Part 1 notebook that shows the 3D viewing volume. Also known as the frustum.

This notebook differs from the previous in that it
from pixels on the image plane and moving out

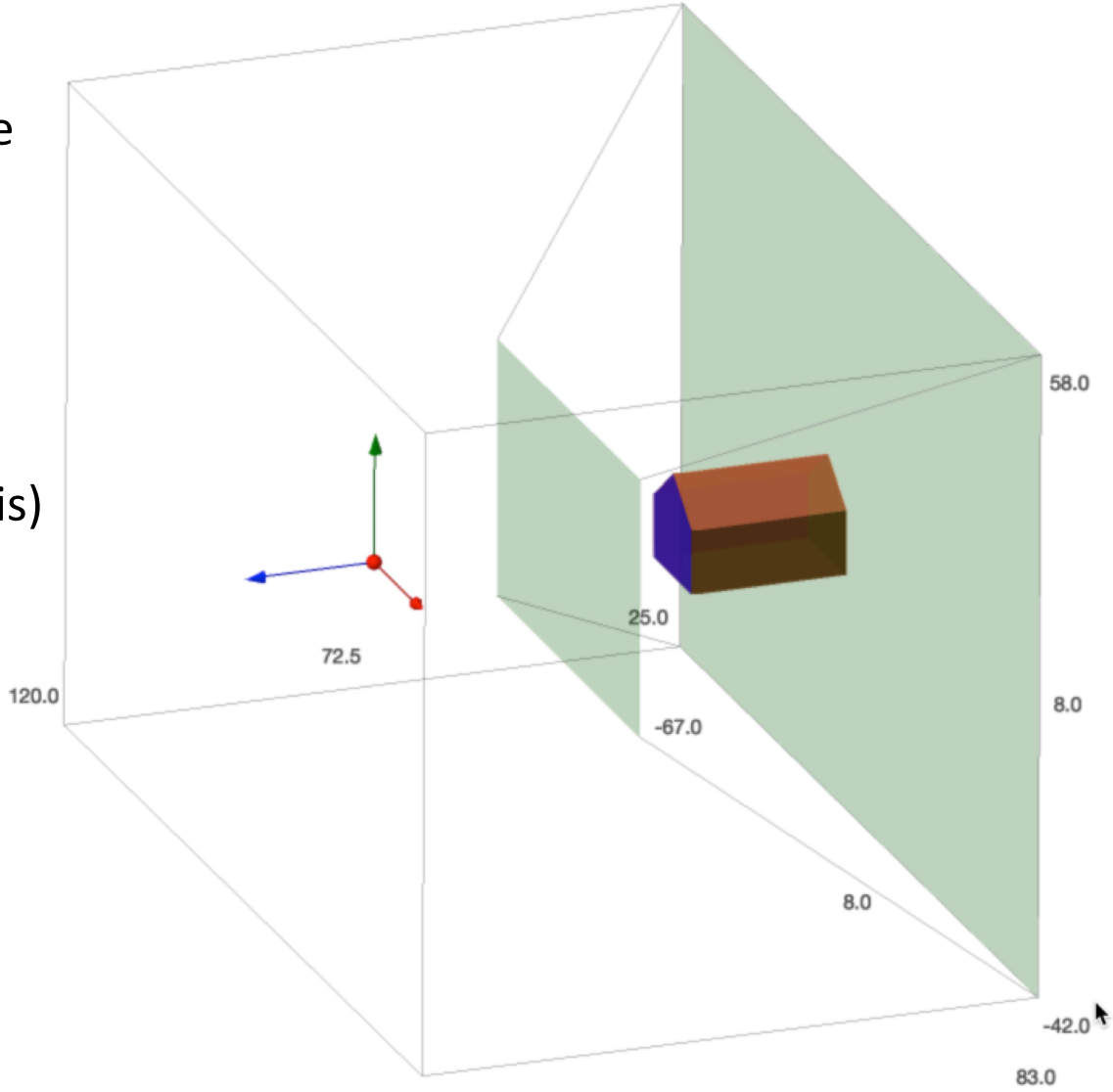
Ross Beveridge September 17, 2019



Review the Frustum

In this figure you need to quickly be able to recognize the following:

- Eye (Camera Position)
- Focal Point
- W (Viewplane Normal)
- U (Camera horizontal axis)
- V (Camera vertical axis)
- Near clipping plane
- Far clipping plane
- View plane
- Bounded image plane
- Lookat point
- Up vector



Camera Specification

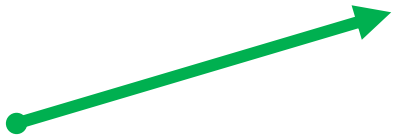
- The following is complete – a bit hard to read.

```
var('ex', 'ey', 'ez');                  # Eye position in the world, also focal point position.
var('lx', 'ly', 'lz');                  # Lookat position in the world.
var('upx', 'upy', 'upz');               # The up vector in the world coordinates.
var('right','left','top','bottom'); # View Volume Sides
var('near', 'far');                     # Distance to the near and far clipping planes.
var('width', 'height');                 # Number of pixels horizontal and vertical
# Setup specific Camera
ex  = 8;   ey = 8;   ez  = 100;      # World origin same as camera
lx  = 8;   ly = 8;   lz  =  54;      # Point toward the positive Z axis
upx = 0;  upy = 1;   upz =   0;      # Let the world y axis represent UP
near  = -30; far    = -75;          # The near and far clipping planes
left  = -30; right  =  30;
top   =  20; bottom = -20;
width =   2; height = 2;
# Build camera system origin and axes in world coordinates
EV = vector(SR, 3); EV[0] = ex;  EV[1] = ey;  EV[2] = ez;
LV = vector(SR, 3); LV[0] = lx;  LV[1] = ly;  LV[2] = lz;
UP = vector(SR, 3); UP[0] = upx; UP[1] = upy; UP[2] = upz;
WV = EV - LV; WV = WV / WV.norm();
UV = UP.cross_product(WV); UV = UV / UV.norm();
VV = WV.cross_product(UV);
```

What is a Ray?

- Beware this simple question
  - The answer may vary by context.
- For our purposes here, a ray is ..
  - A pair consisting of a Point and a Vector
  - The ray 'originates' at the Point P.
  - Moves in the direction indicated by vector D.

$$\mathrm{R}(t) = P + tD \quad t \geq 0$$



This is our first of a parametric object.

A Pixel's Ray

- We want to 'fire' a ray from each pixel in an image which we are constructing.
- What is the point where the ray starts?
  - The 3D world position of the pixel.
- What direction does it travel?
  - The direction defined by
    - the focal point (Eye)
    - the pixel position.
  - Both are measured in world coordinates.

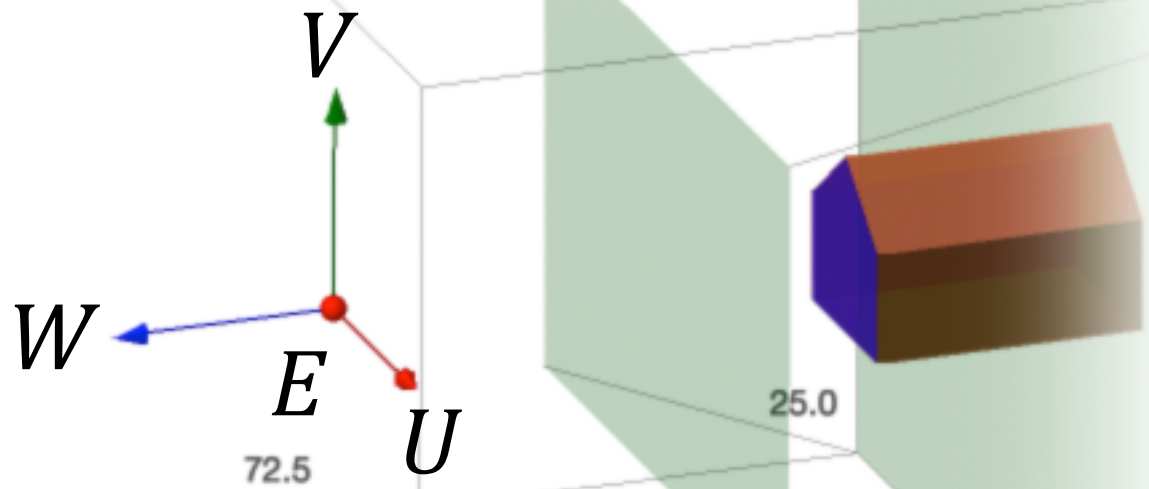
Pixel 3D Coordinates

- Directions needed to arrive at a pixel.
  - Begin a the camera focal point (Eye)
  - Move on the z-axis to the image plane
  - Move on the x-axis to the ‘proper position’
  - Move on the y-axis to the ‘proper position’
  - Proper position means converting pixel (i, j) to distances of travel in the world.

$$P = E + nW + \alpha U + \beta V$$

Equation Illustrated

$$P = E + nW + \alpha U + \beta V$$



Now consider:

Is n a positive or negative scalar?

What do we need to compete α and β?

SageMath Code

```
def pixelPt(i,j):
    px = i/(width-1)*(right-left)+left;
    py = j/(height-1)*(top-bottom)+bottom;
    pixpt = EV + (near * WV) + (px * UV) + (py * VV);
    return point(pixpt,size=10);
```

- near value is negative, e.g. -30 in example
- px is position along U of pixel at index i
  - i from 0 to (width – 1), e.g. 0 to 7 in example
  - Test boundary cases:

$$px(0) = \frac{0}{width - 1} * (right - left) + left = left$$

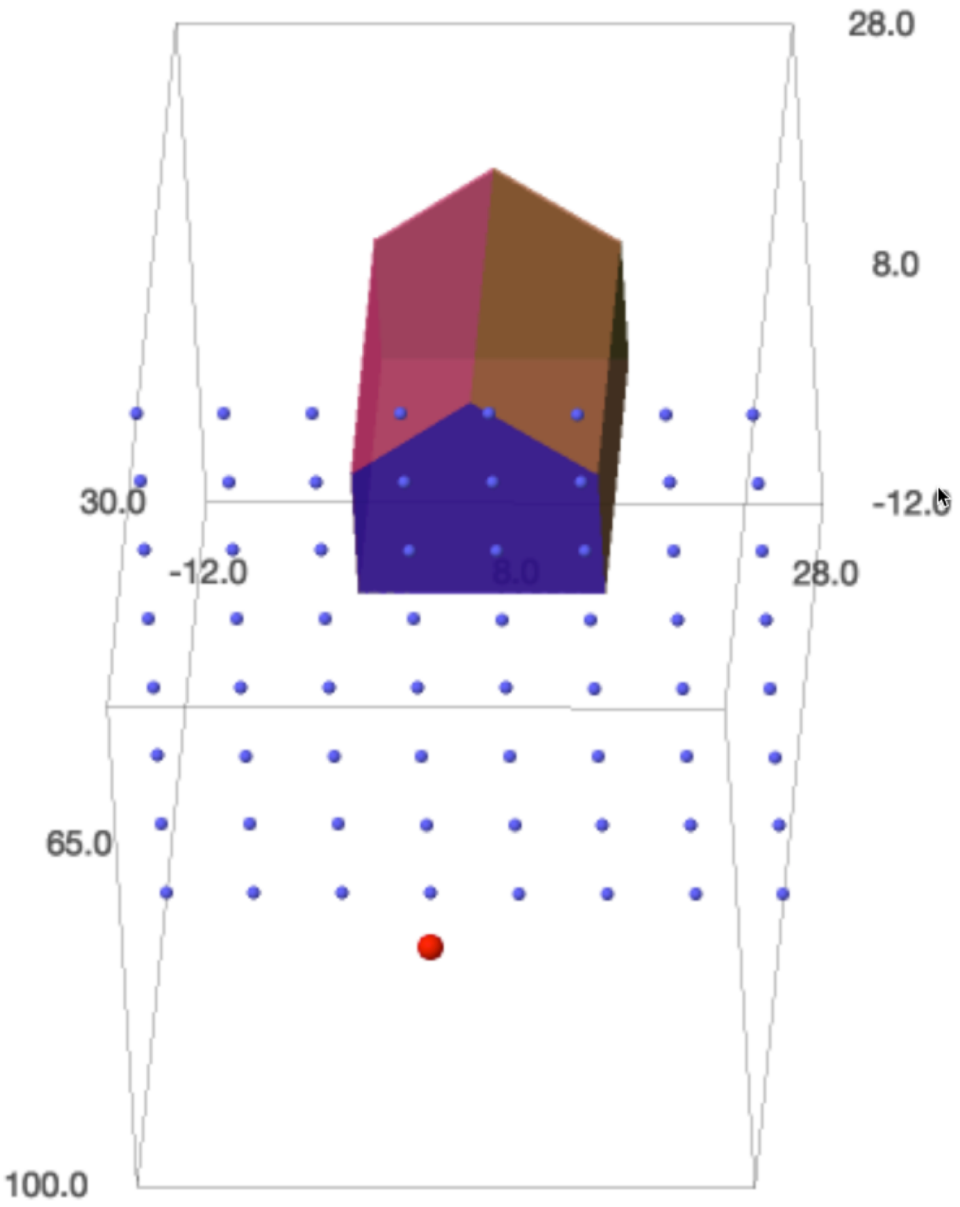
$$px(width - 1) = \frac{width - 1}{width - 1} * (right - left) + left = right$$

64 3D Pixels

It is NOT common to force students to think of pixels in terms of 3D world/scene points so early in learning graphics.
However, once you get used to the idea, then camera placement for ray-tracing will become ‘simple’.

Pay attention to …

- Bounds on horizontal axes
- Bounds on vertical axes
- Is there a center pixel?



Ray From a Pixel: Math

- Let P be the pixel point.
- Let E be the focal point (Eye).
- The ray is:

$$R(t) = P + tD \qquad D = \frac{P - E}{\|P - E\|}$$

- Expressing direction as a unit length vector is generally a very good idea.

Ray From a Pixel: Data Structure

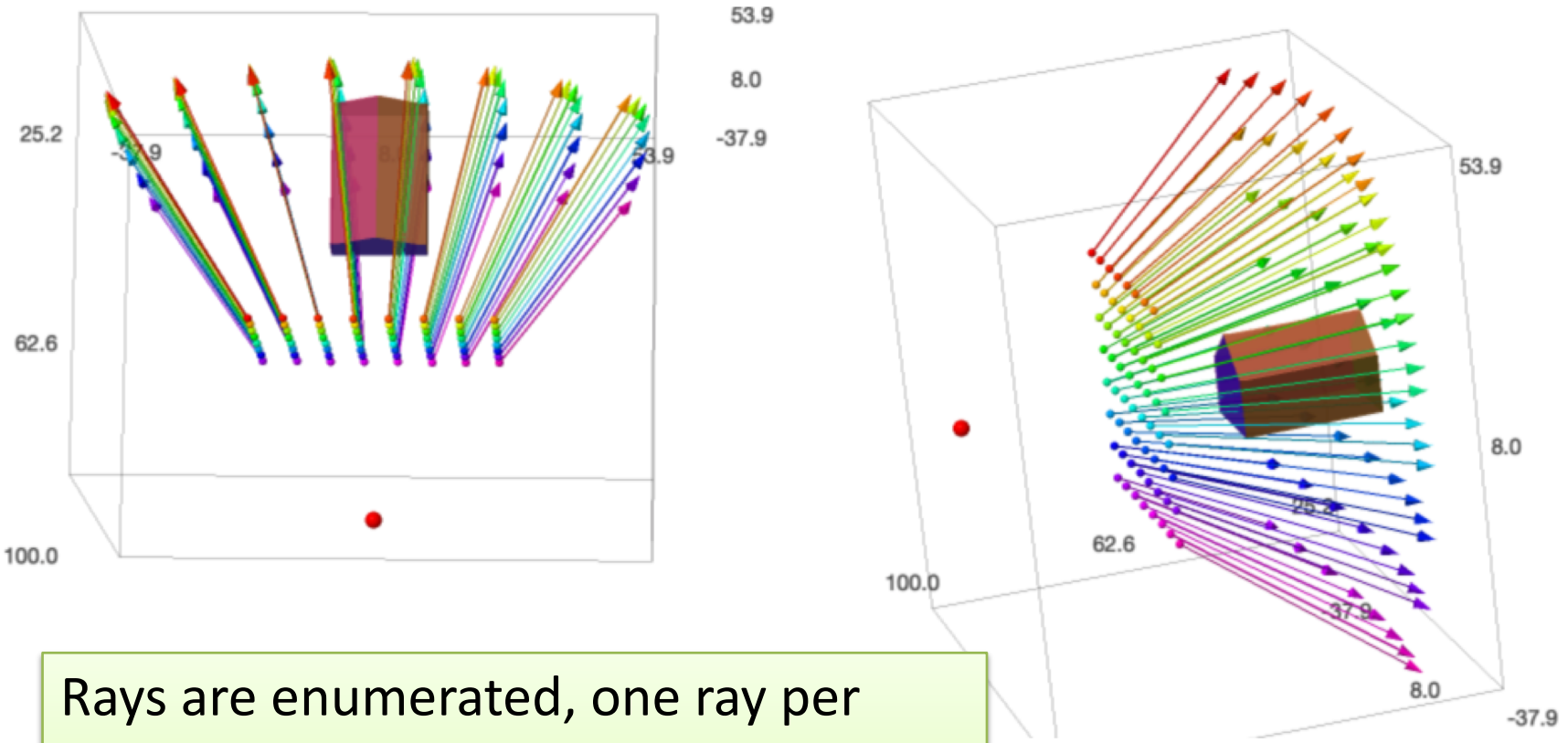
- Code view is somewhat different.
- You have many options …
- Here is one approach
  - Object class for **`Point`**
  - Object class for **`Vector`**
  - Object class for **`Ray`**
    - A Ray includes a **`Point`**
    - A Ray includes a **`Vector`**

Ray in SageMath

```
def pixelRay(i,j):
    px = i/(width-1)*(right-left)+left;
    py = j/(height-1)*(top-bottom)+bottom;
    pixpt = EV + (near * WV) + (px * UV) + (py * VV);
    shoot = pixpt - EV; shoot = shoot / shoot.norm();
    raypt = pixpt + shoot * abs(far-near);
    return arrow3d(pixpt, raypt, width=16,color='orange');
```

- Code creates an orange arrow from pixel point to a point in the direction of the ray and a distance (far-near) away from the pixel.
- Note the variable ‘shoot’ plays the role of ‘D’ in the ray equations above.

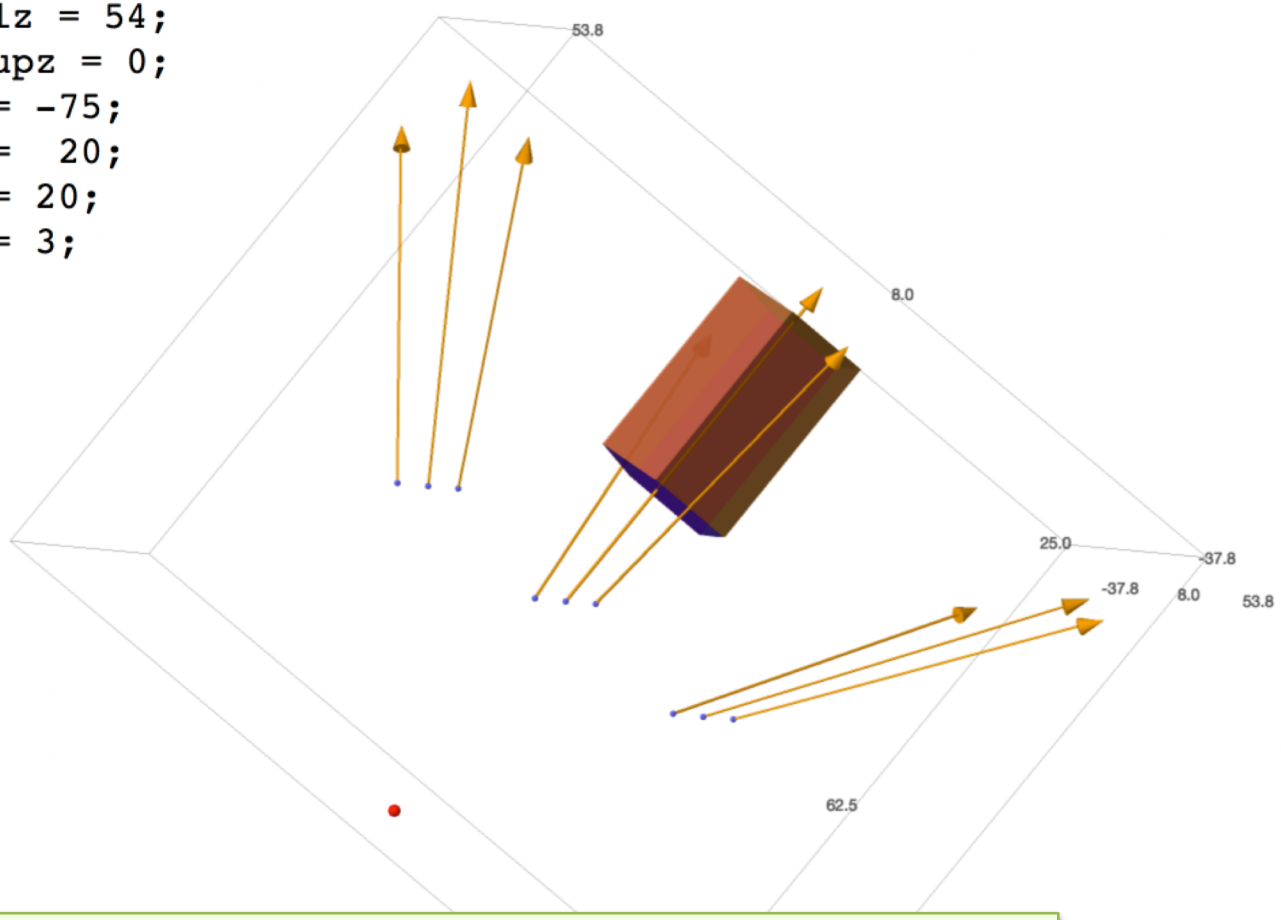
Déjà vu



Rays are enumerated, one ray per pixel, using the camera specification. Now let us explore some alternative configurations.

Example 1: Optical Axis

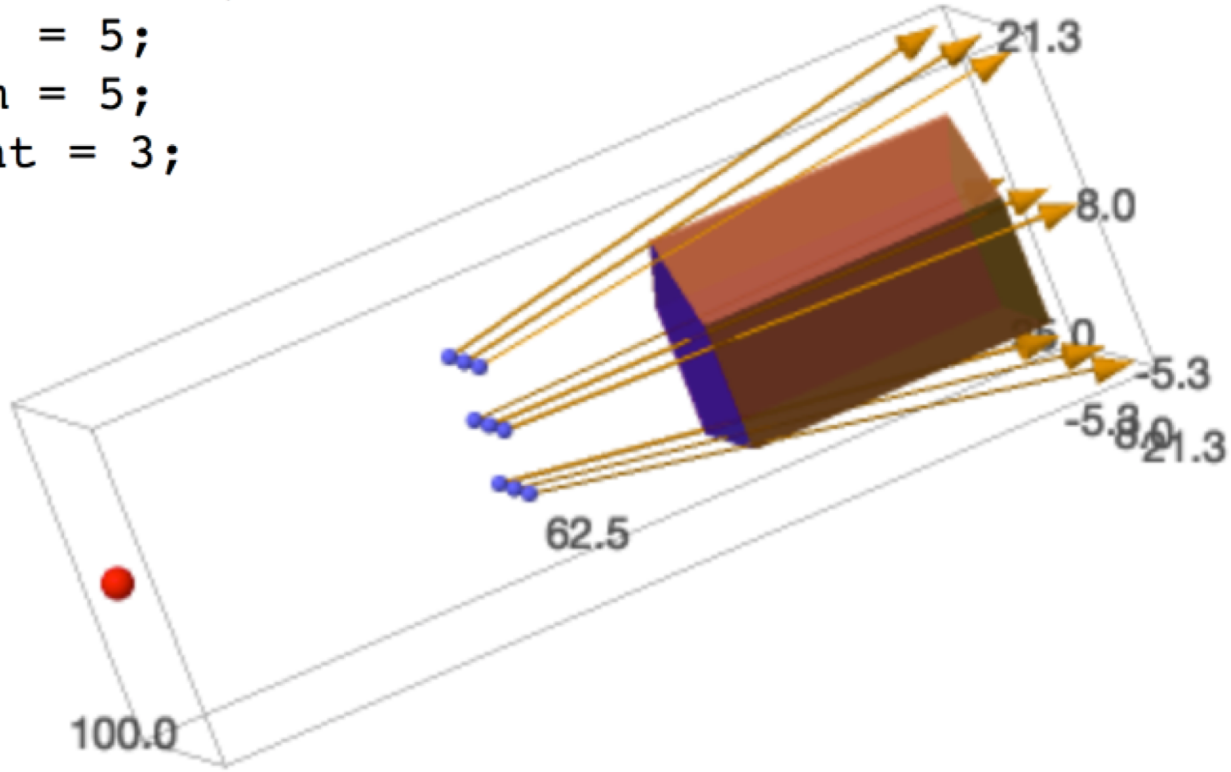
```
# Setup specific Camera
ex = 8;    ey = 8;    ez = 100;
lx = 8;    ly = 8;    lz = 54;
upx =0;   upy = 1;    upz = 0;
near  = -30; far    = -75;
left  = -20; right  =  20;
top   = -20; bottom = 20;
width =   3; height = 3;
```



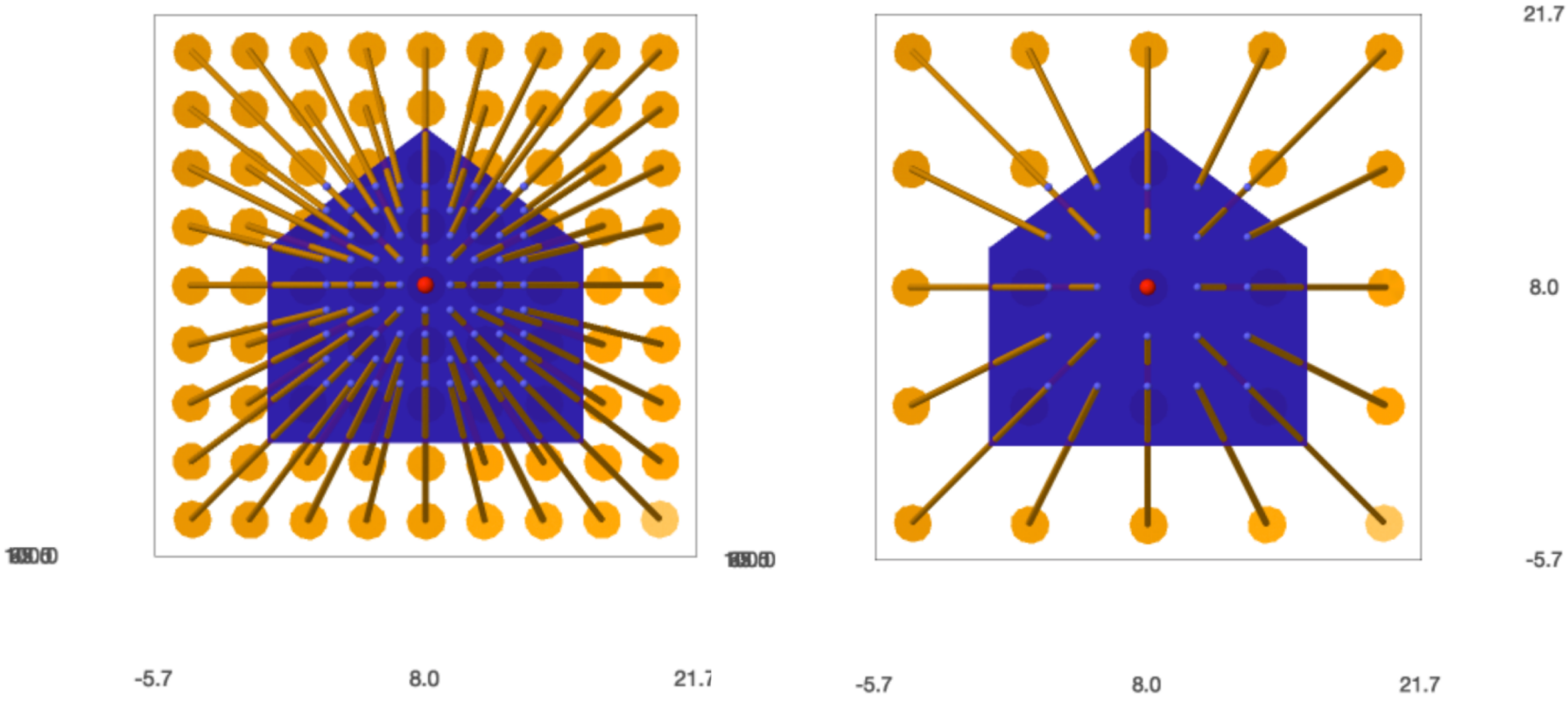
Is there a pixel centered on the optical axis?

Example 2: Zoom

```
# Setup specific Camera
ex = 8;    ey = 8;    ez = 100;
lx = 8;    ly = 8;    lz = 54;
upx =0;   upy = 1;    upz = 0;
near  = -30; far     = -75;
left  = -5; right   = 5;
top   = -5; bottom = 5;
width =   3; height = 3;
```



Examples: Pixel Density



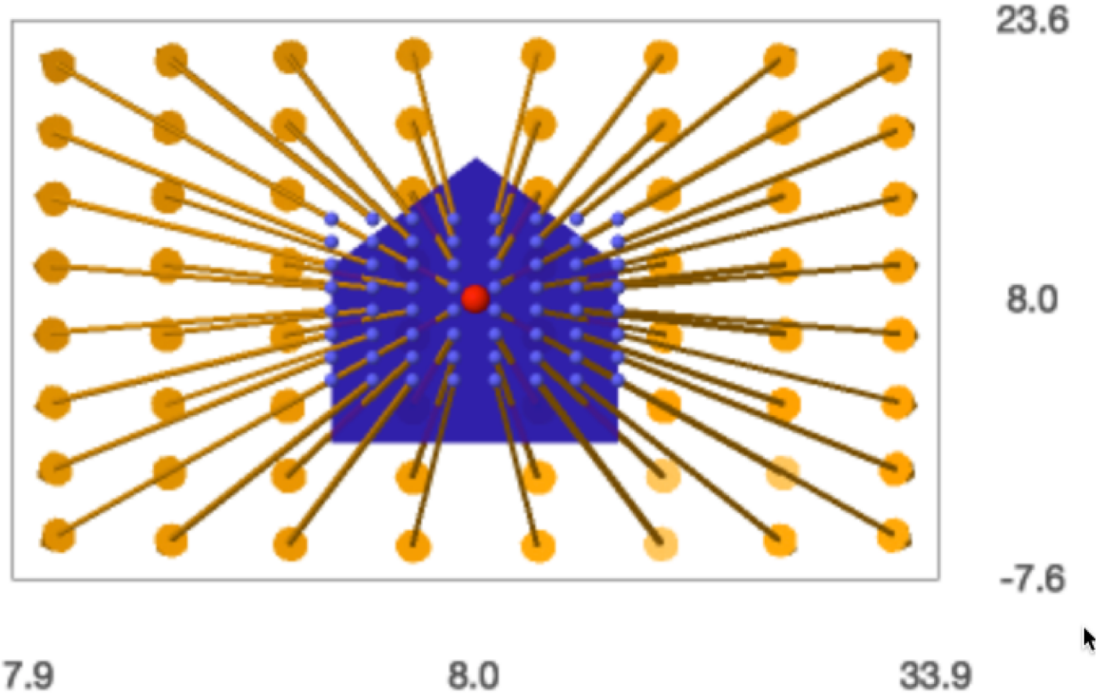
Do NOT confuse camera field of view with pixel density, i.e. the number of pixels in an image.

Example 04: Square Pixels?

```
# Setup specific Camera
ex = 8;   ey = 8;   ez = 90;
lx = 8;   ly = 8;   lz = 54;
upx =0;  upy = 1;   upz = 0;
near  = -20;  far    = -65;
left  = -8;   right  = 8;
top   = -4.5; bottom = 4.5;
width = 8; height = 8;
```

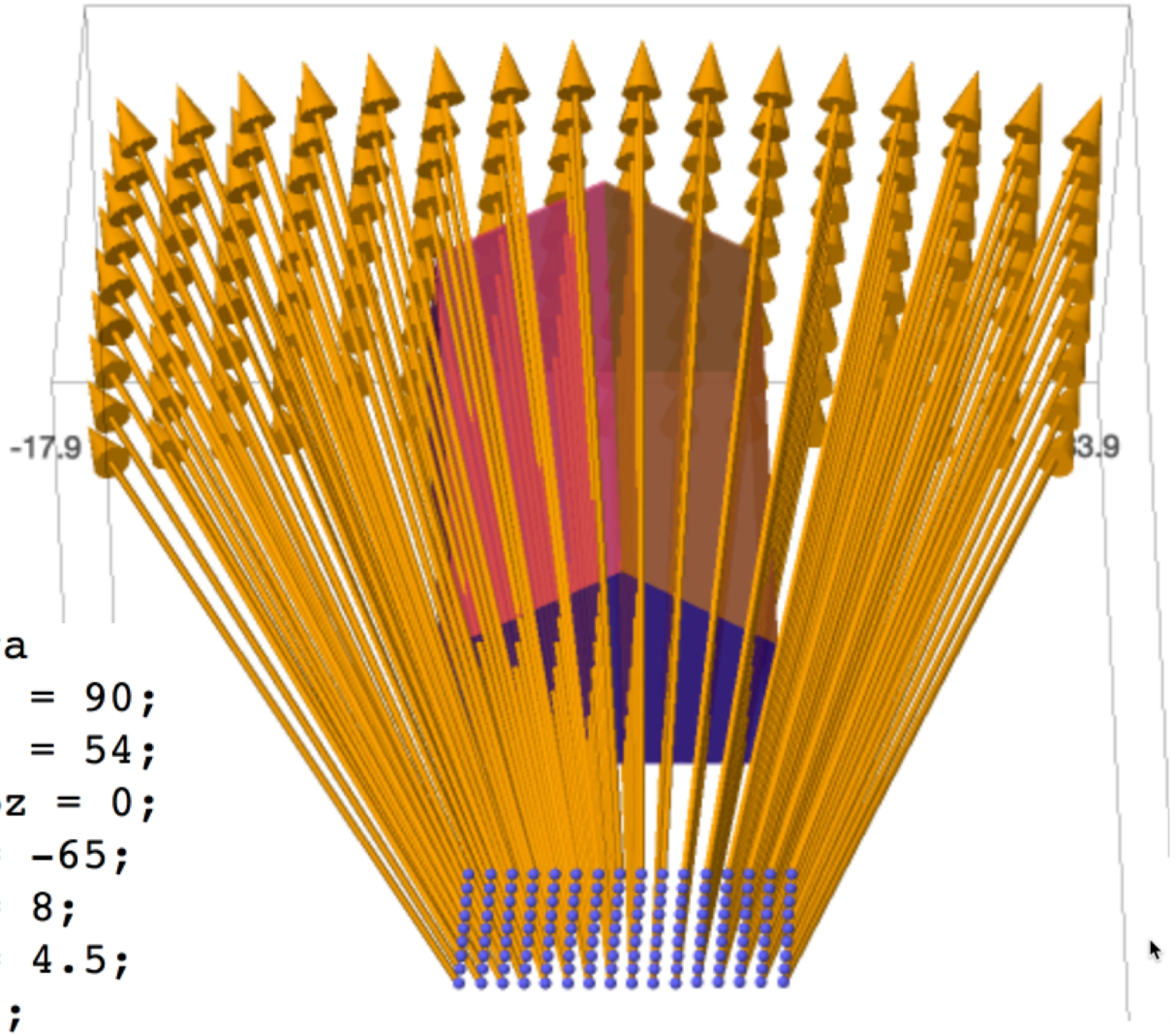
This is an 8x8 pixel image.

The aspect ratio of the frustum is defined by left, right, top and bottom.



Example 5: Square is Good

Square Pixels mean equally spaced samples horizontally and vertically.

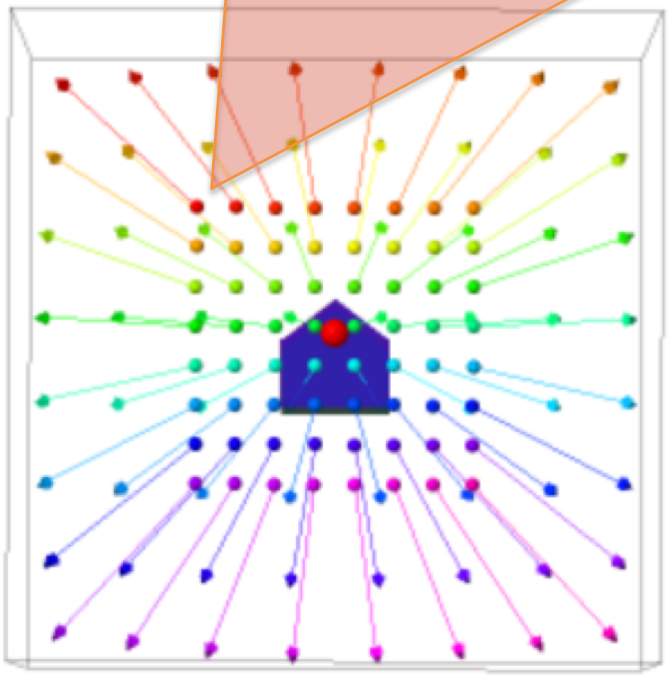


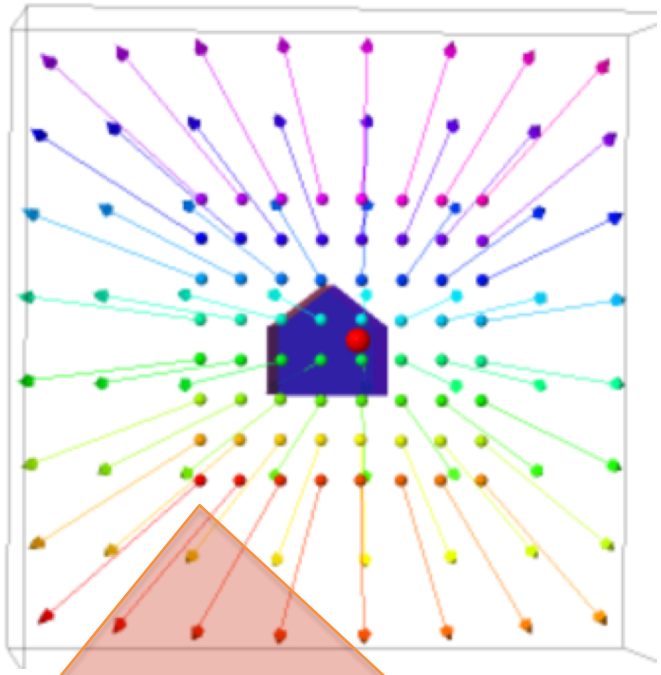

```
# Setup specific Camera
ex = 8;    ey = 8;    ez = 90;
lx = 8;    ly = 8;    lz = 54;
upx =0;   upy = 1;    upz = 0;
near   = -20;   far     = -65;
left   = -8;    right   = 8;
top    = -4.5; bottom = 4.5;
width = 16; height = 9;
```

Example 6: Pixel Zero Zero

On the rainbow map, pixel (0,0,) is red.

```
py = j/(height-1)*(bottom-top)+top;
```





```
py = j/(height-1)*(top-bottom)+bottom;
```

General Parametric Approach

- Sphere centered at Pc with radius r.

$$\left|P - P_c\right|^2 = r^2$$

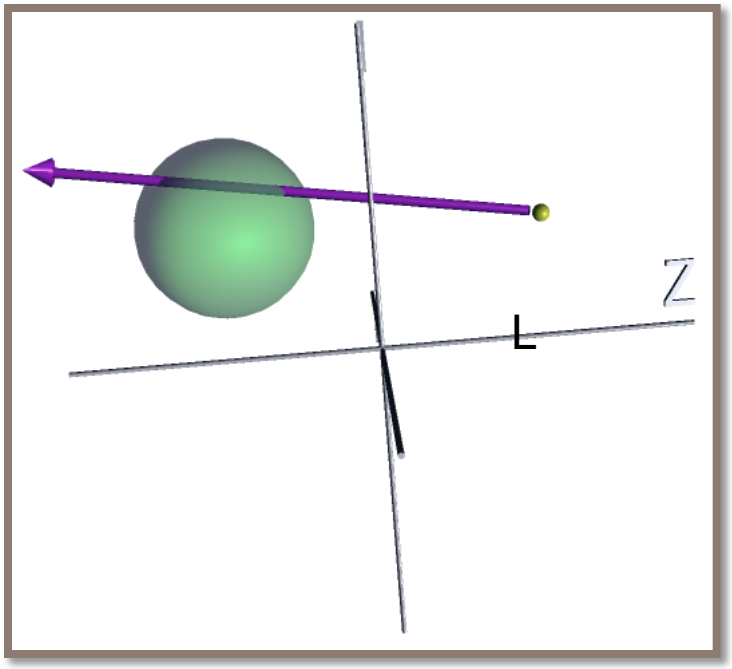
$$\left|L + sU - P_c\right|^2 - r^2 = 0$$

substitute

$$T = P_c - L$$

Yielding a quadratic equation

$$\left(sU - T\right)\cdot\left(sU - T\right) - r^2 = 0$$



Solve quadratic equation for *s* and take smaller of two real roots

But! There is a better way.

- Side-steps general parametric approach.
- Readily understood in terms of dot products.
- If you are looking additional information:
  - Glassner, A. (ed) An Introduction to Ray Tracing. Academic Press, 1989
  - http://www.cs.unc.edu/~rademach/xroads-RT/RTarticle.html
- Now, ray sphere intersection.

Heavy Reliance on SageMath

jupyter cs410lec07n02 Last Checkpoint: a few seconds ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted SageMath 8.8

Run Code

Clever Ray Sphere Intersection

This notebook presents a much faster and simpler means of computing the intersection between a sphere and a ray. There are several 'key insights' about the relative geometry of a ray and sphere that must be understood before the mathematics of this approach makes geometric sense. To try to help visualize these connections this notebook uses explicit values for an example.

Ross Beveridge, September 17, 2019

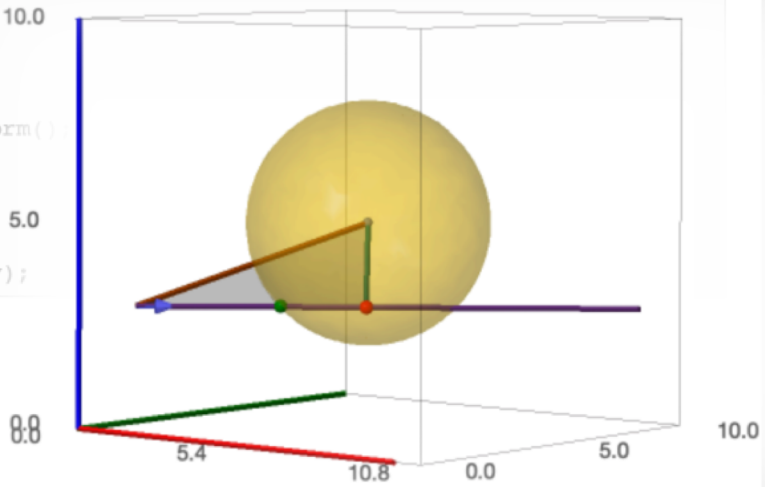
In [13]:
```
var('r');
r = 3;
Cv = vector(SR, 3, (5,5,5));
Lv = vector(SR, 3, (1,1,3));
Uv = vector(SR, 3, (4,3,0)); Uv = Uv / Uv.norm()
Tv = vector(SR, 3, (Cv - Lv));
pretty_print("Sphere center: C = ", Cv);
pretty_print("Ray start:     L = ", Lv);
pretty_print("Ray direction: U = ", Uv);
pretty_print("Base to Center: T = ", Cv - Lv);
```

Sphere center: $C = (5, 5, 5)$

Ray start: $L = (1, 1, 3)$

Ray direction: $U = \left(\frac{4}{5}, \frac{3}{5}, 0\right)$

Base to Center: $T = (4, 4, 2)$



Faster Method

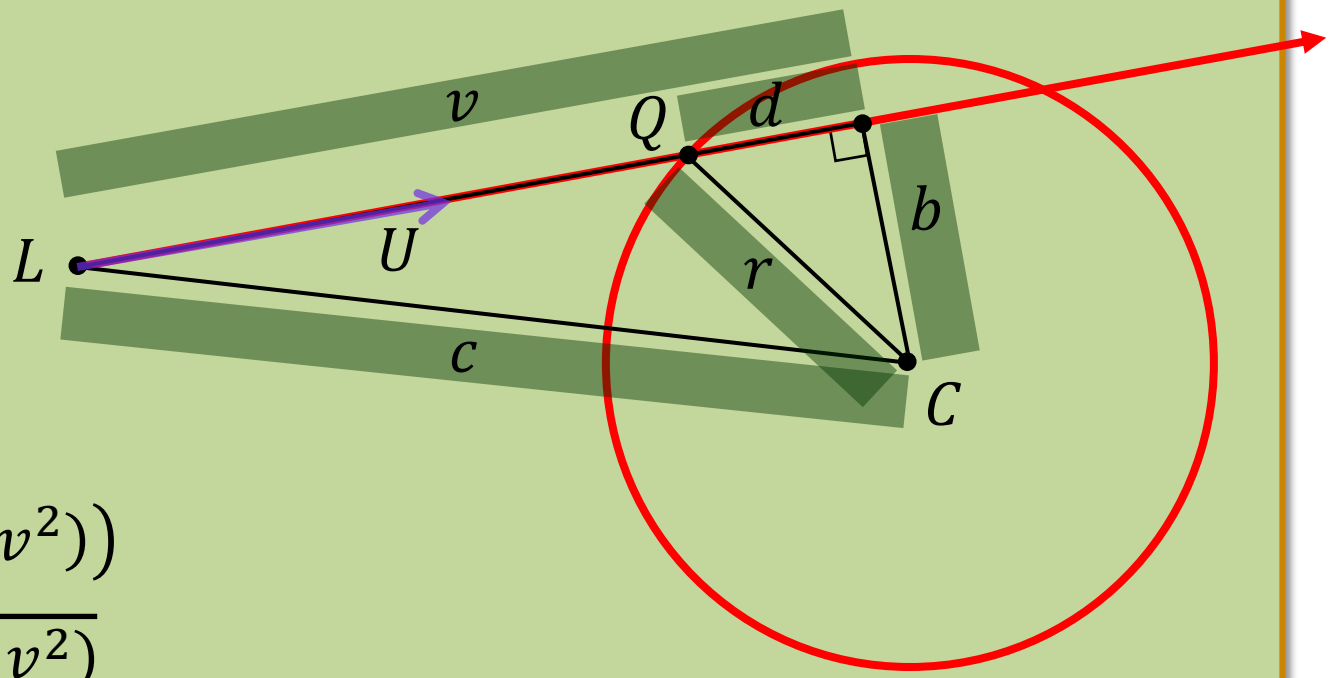
$$v = (C - L) \cdot U$$

$$v^2 + b^2 = c^2$$

$$d^2 + b^2 = r^2$$

$$d^2 = \left(r^2 - (c^2 - v^2)\right)$$

$$d = \sqrt{r^2 - (c^2 - v^2)}$$



If d^2 less than zero, no intersection.
Otherwise, $Q = L + (v - d)U$

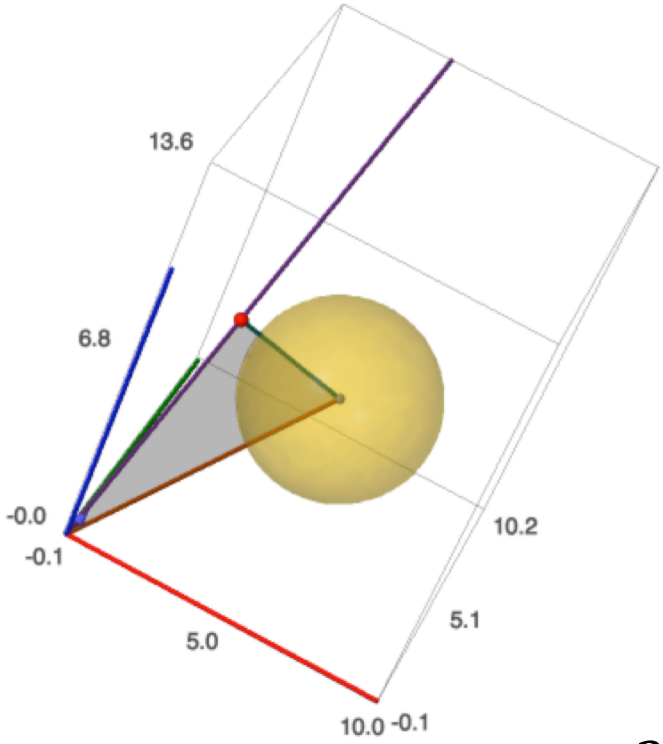
Example 1

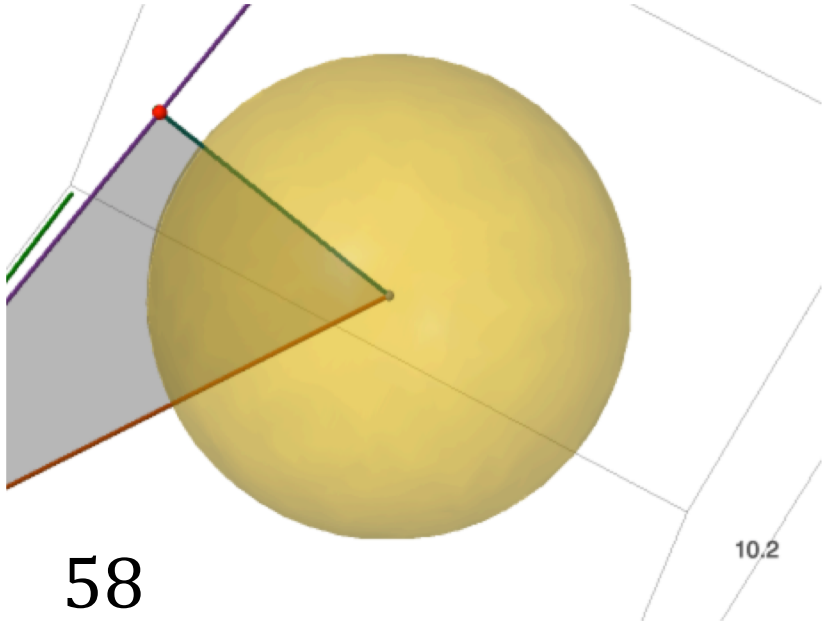
Sphere center: $C = (5, 5, 5)$

Ray start: $L = (0, 0, 0)$

Ray direction: $U = \left(\frac{1}{26}\sqrt{26}, \frac{3}{26}\sqrt{26}, \frac{2}{13}\sqrt{26} \right)$

Base to Center: $T = (5, 5, 5)$





$$r^2 - b^2 = -\frac{58}{13}$$

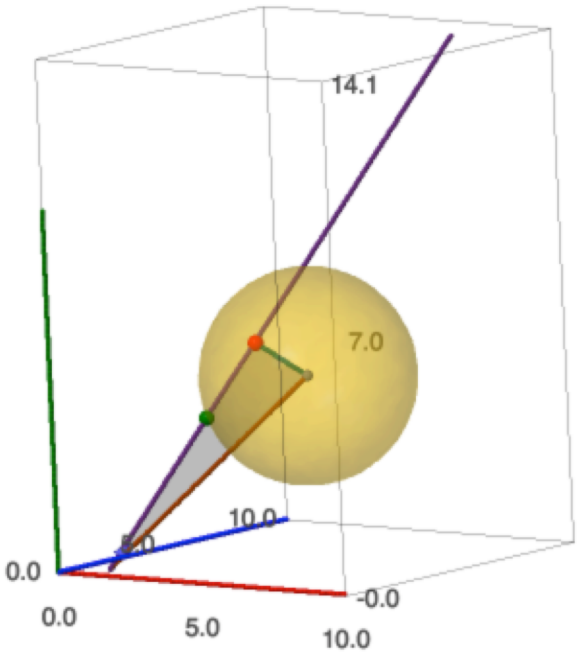
Example 2

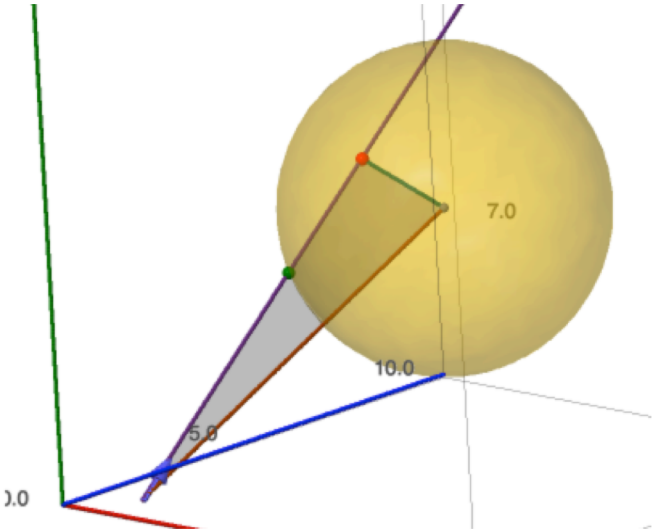
Sphere center: $C = (5, 5, 5)$

Ray start: $L = (1, 0, 1)$

Ray direction: $U = \left(\frac{1}{6}\sqrt{6}, \frac{1}{3}\sqrt{6}, \frac{1}{6}\sqrt{6}\right)$

Base to Center: $T = (4, 5, 4)$





Option: Rays from Focal Point

- Our rays originate from pixel in world cordinates L.

$$R(s) = L + sU$$

- With the unit vector pointing from focal point E to pixel L.

$$U = \frac{L - E}{\|L - E\|}$$

- Alternatively, let rays originate from focal point E.

$$R(s) = E + sU$$

How might this help?

Intermediate values remain constant across all pixels when always using the focal point E as the base of the ray.