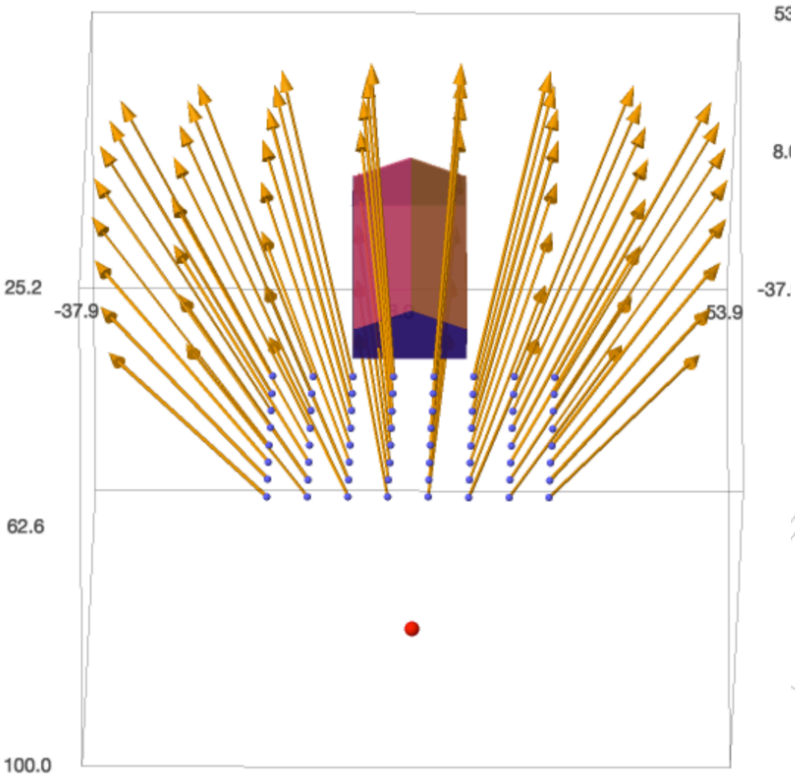
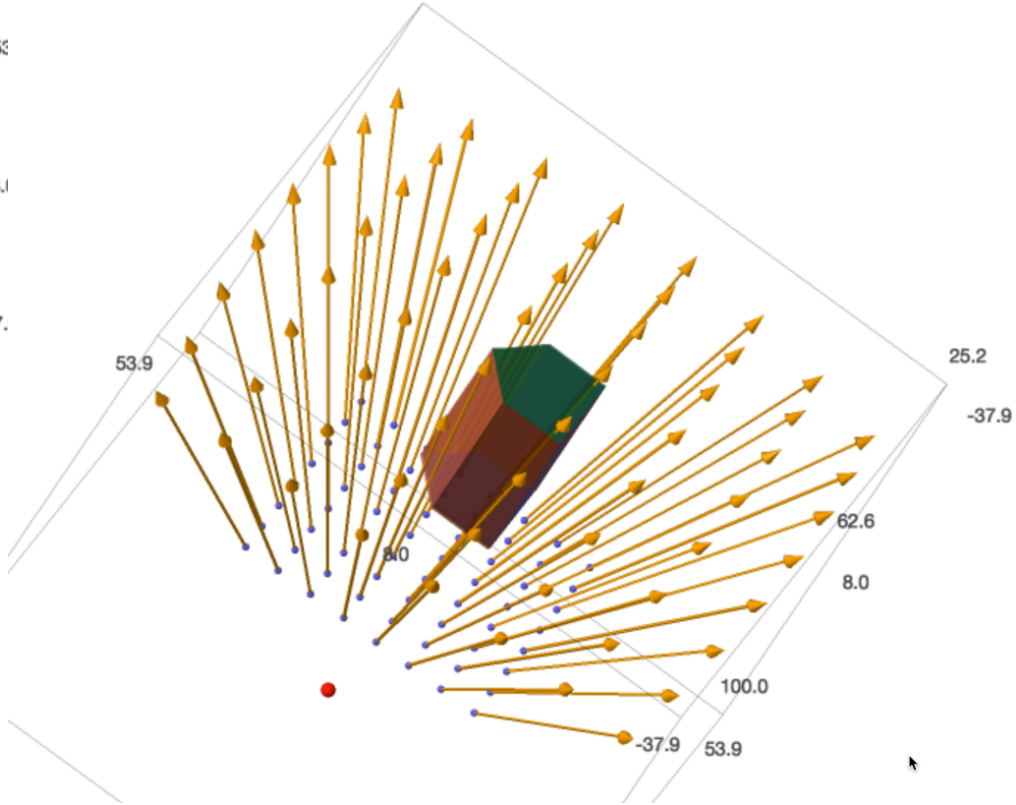
Lecture 07: Shooting Rays

September 12, 2017

Goal: Understand …





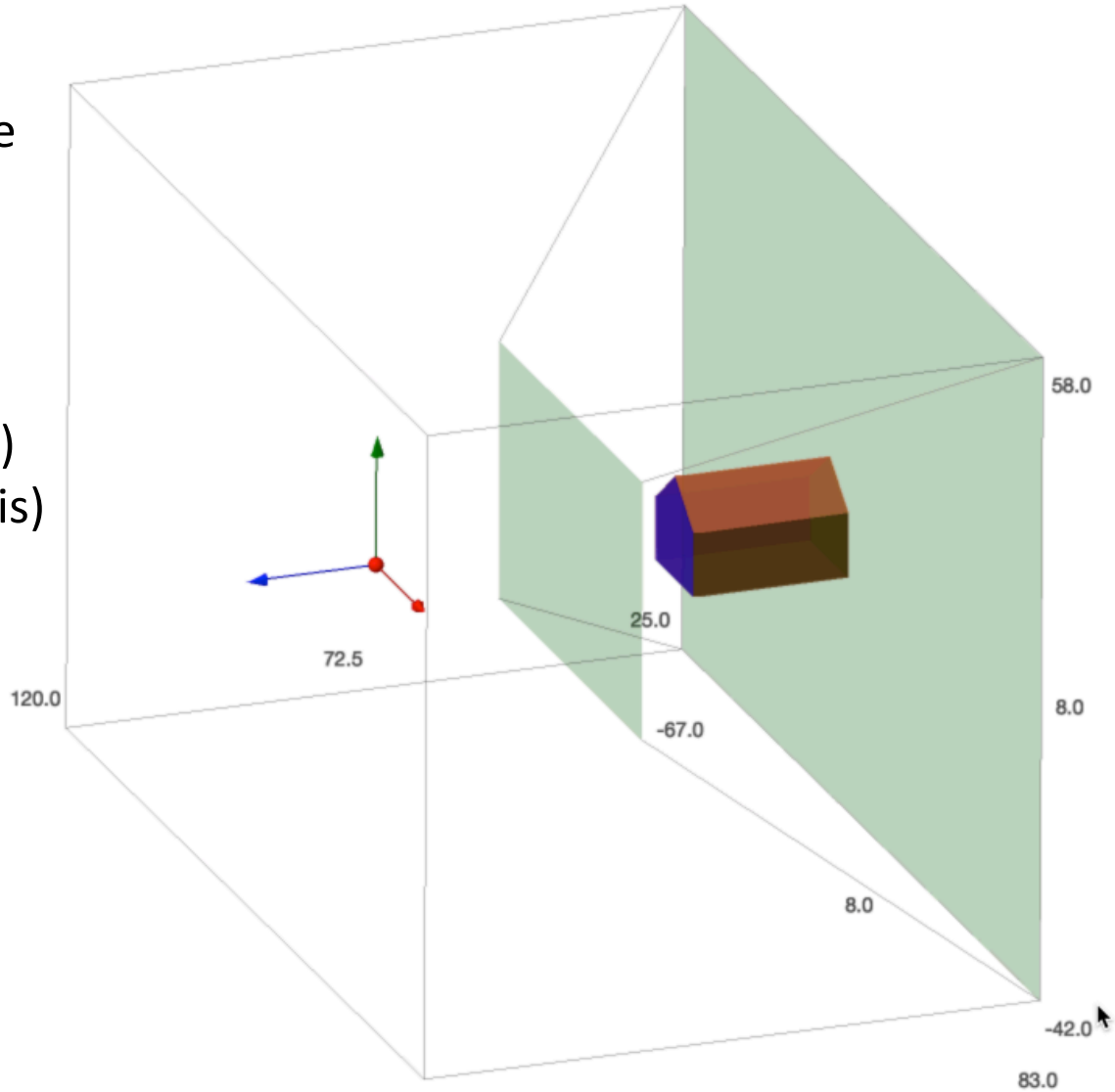
Understand means:

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

Review the Frustum

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

- Eye (Camera Position)
- Focal Point
- VPN (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

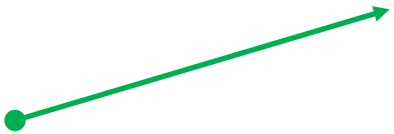
- The following is complete – a bit hard to read.

```
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.
- But, 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.

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



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) and the pixel position.
  - Both measured in world coordinates.

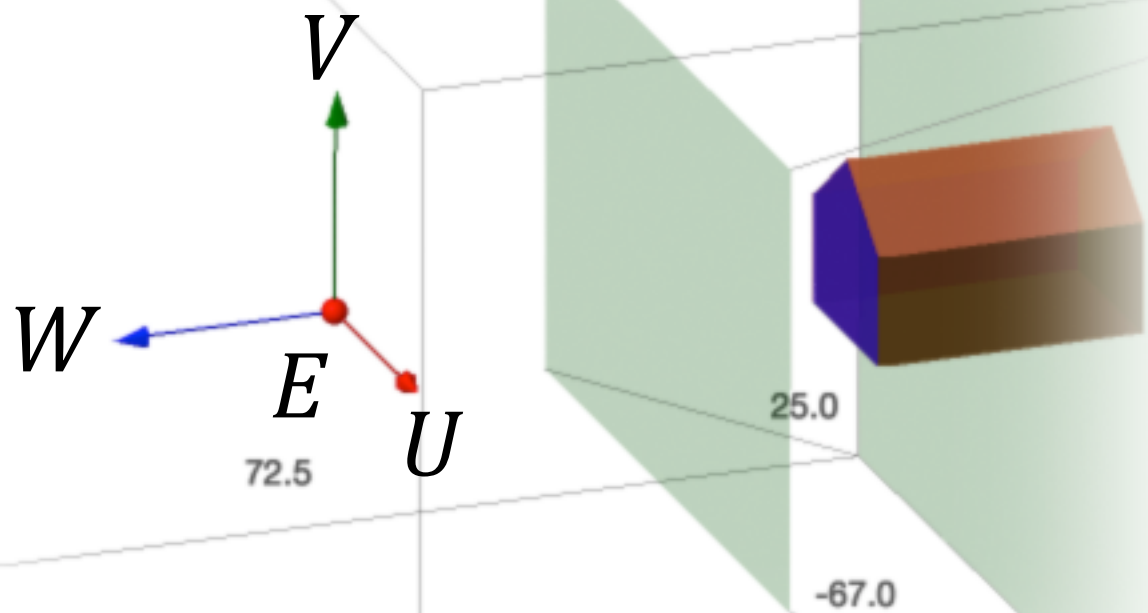
Pixel 3D Coordinates

- 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:

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

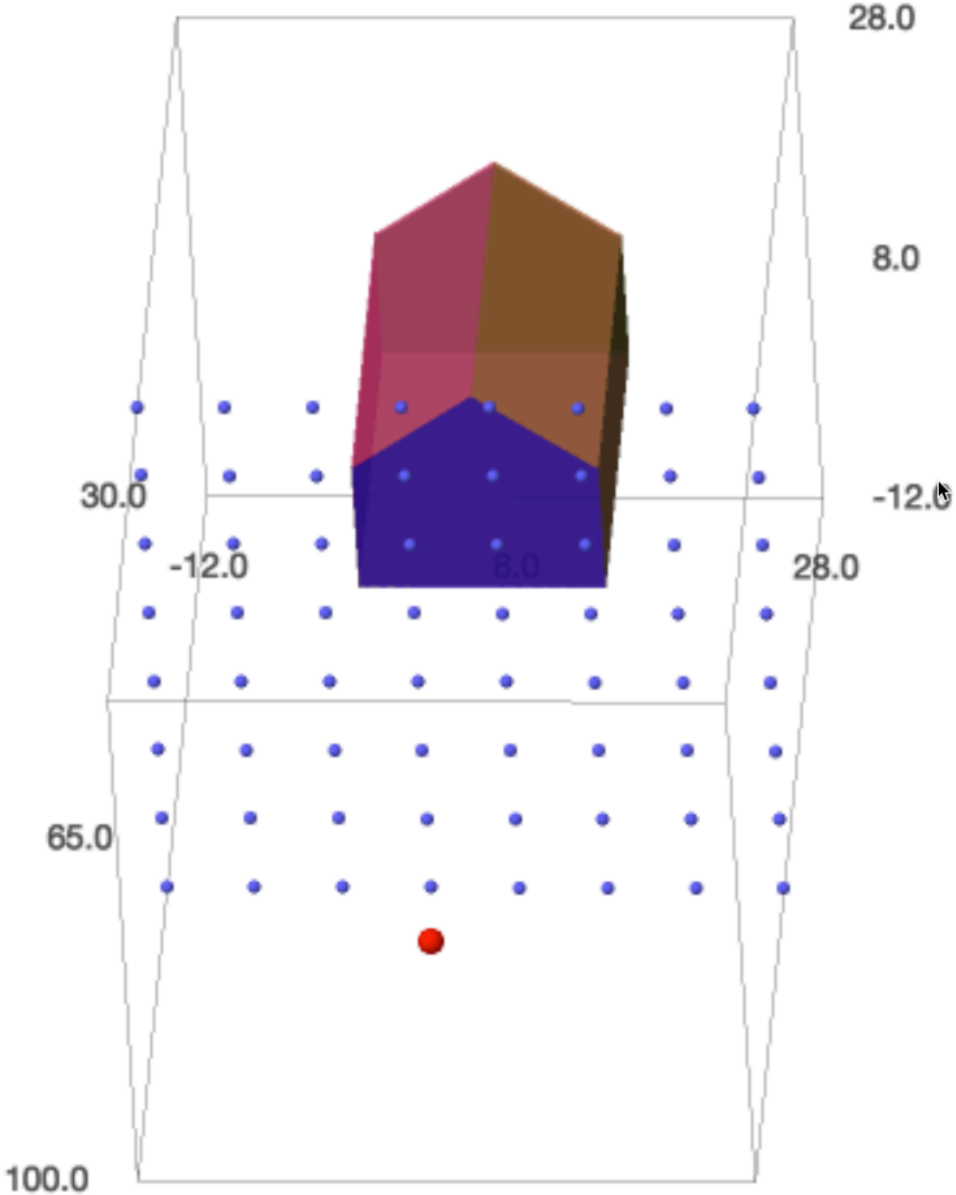
$$px(width - 1) = \frac{width - 1}{width - 1} * (right - left) + left = right$$

64 3D Pixels

At this stage be aware that it is not common to force you 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: Datastructure

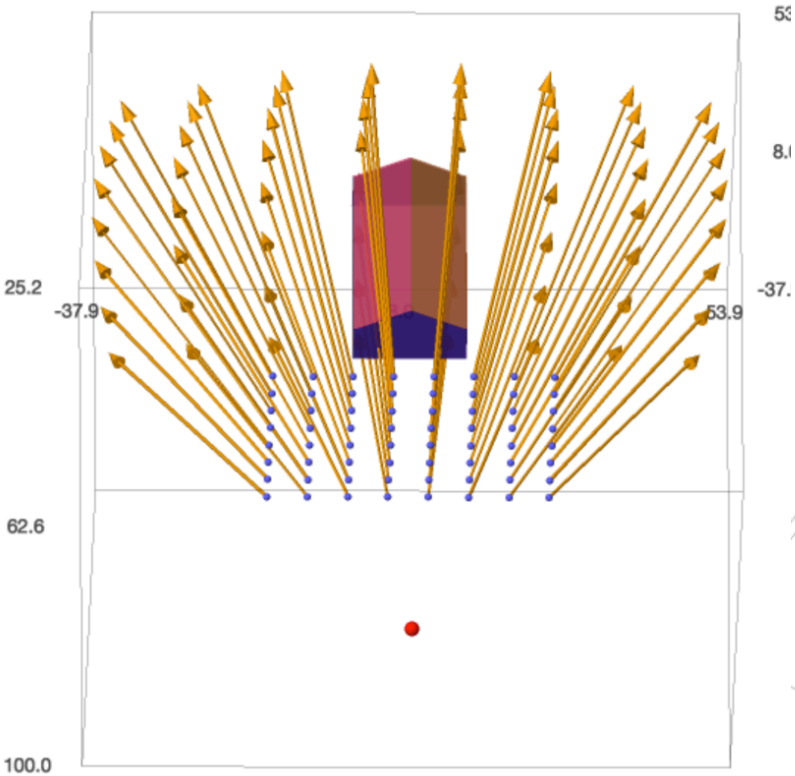
- Code view is somewat different.
- You have many options …
- Here is one approach
  - Object class for **`Point`**
  - Object class for **`Vector`**
  - Object class for **`Ray`**
    - A Ray includes a **`Point`**
    - A Ray includes a **`Vector`**

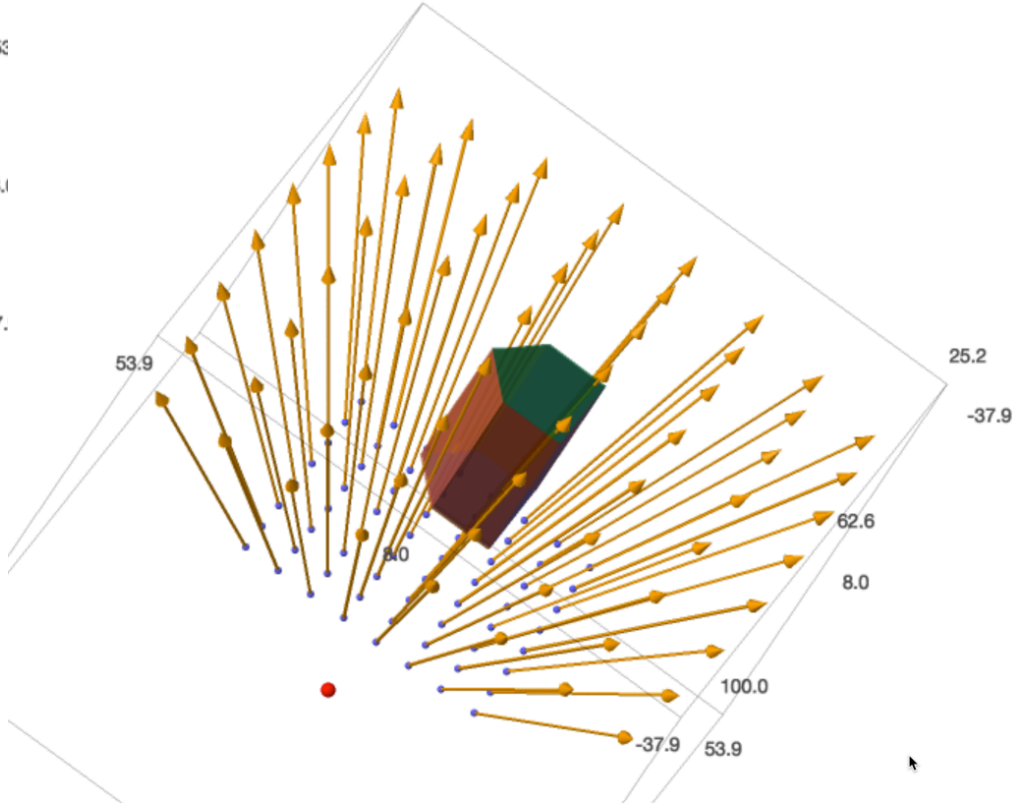
Ray in SageMath

```
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);
    ray   = pixpt - EV; ray = ray / ray.norm();
    raypt = pixpt + ray * 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.
- Beware, ‘ray’ is used locally in a different manner in this SageMath code.

Déjà vu

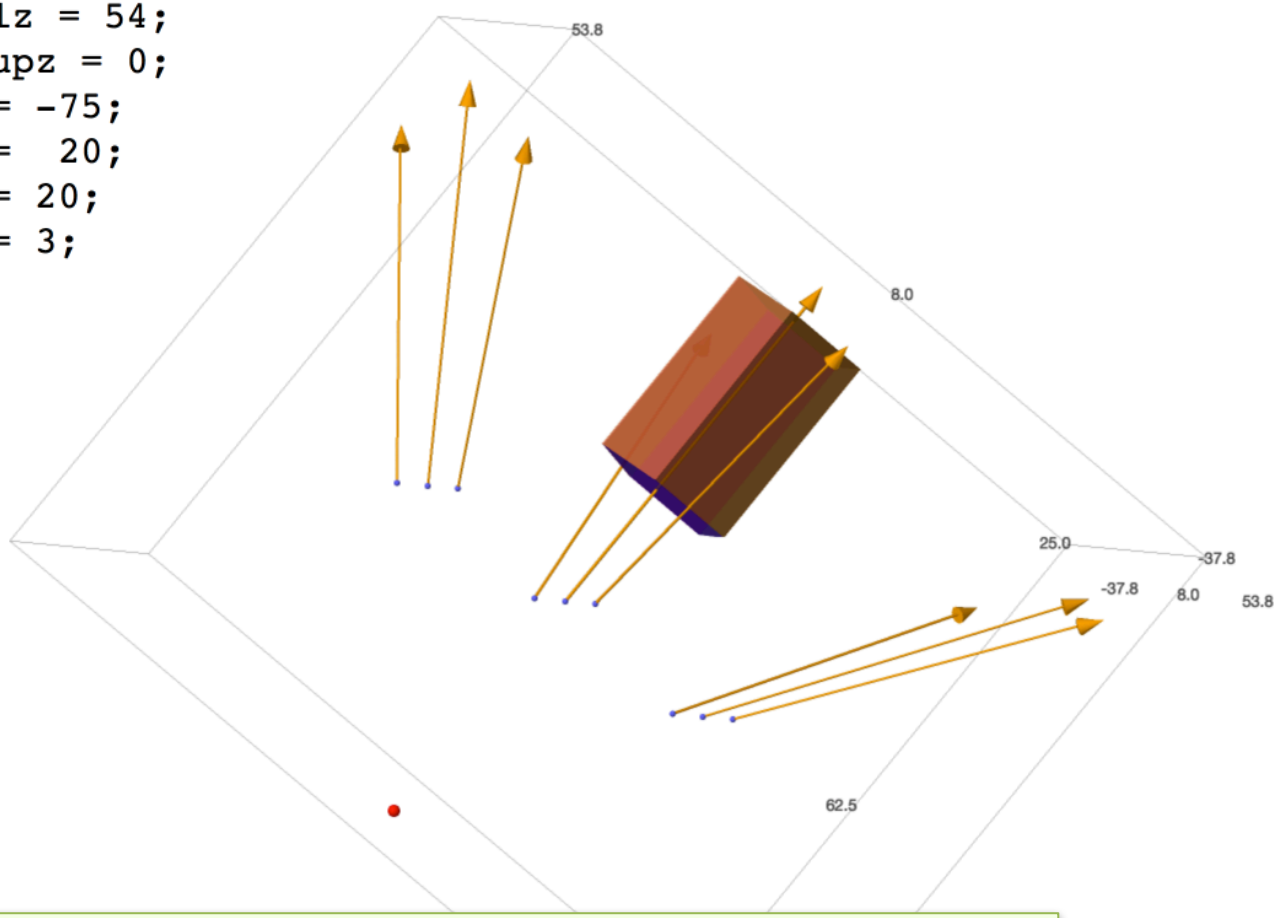




Rays are enumerated, one ray per pixel, using the camera specification. Now let us explore some alternative configurations.

Example 1: Optical Axis

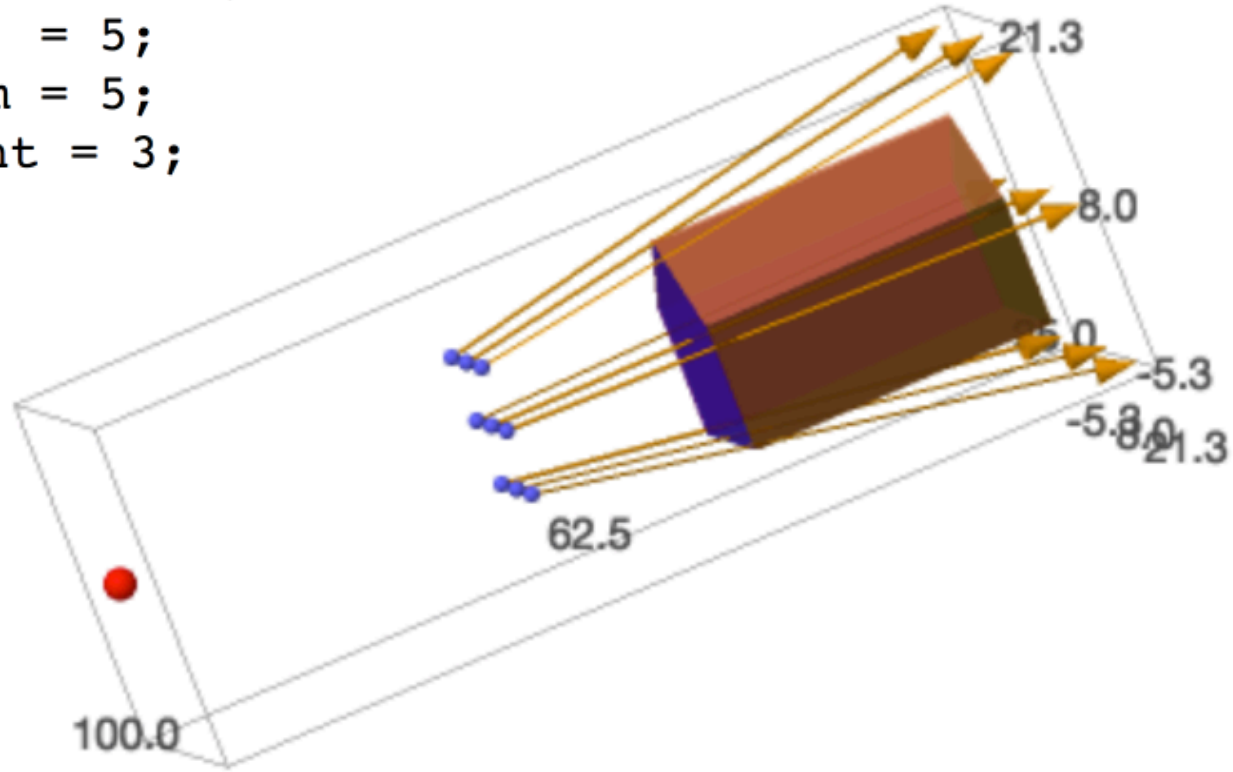
```
# Setup specific Camera
ex = 8;    ey = 8;    ez = 100;
lx = 8;    ly = 8;    lz = 54;
upx =0;   upy = 1;    upz = 0;
near   = -30; far     = -75;
left   = -20; right   =  20;
top    = -20; bottom = 20;
width =    3; height = 3;
```



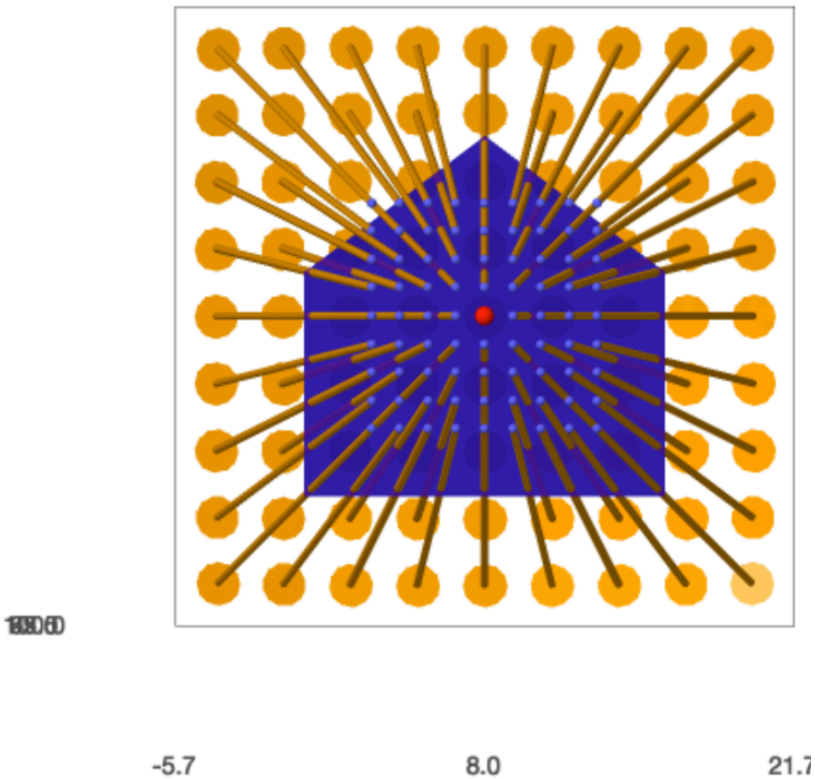
Is there a pixel centered on the optical axis?

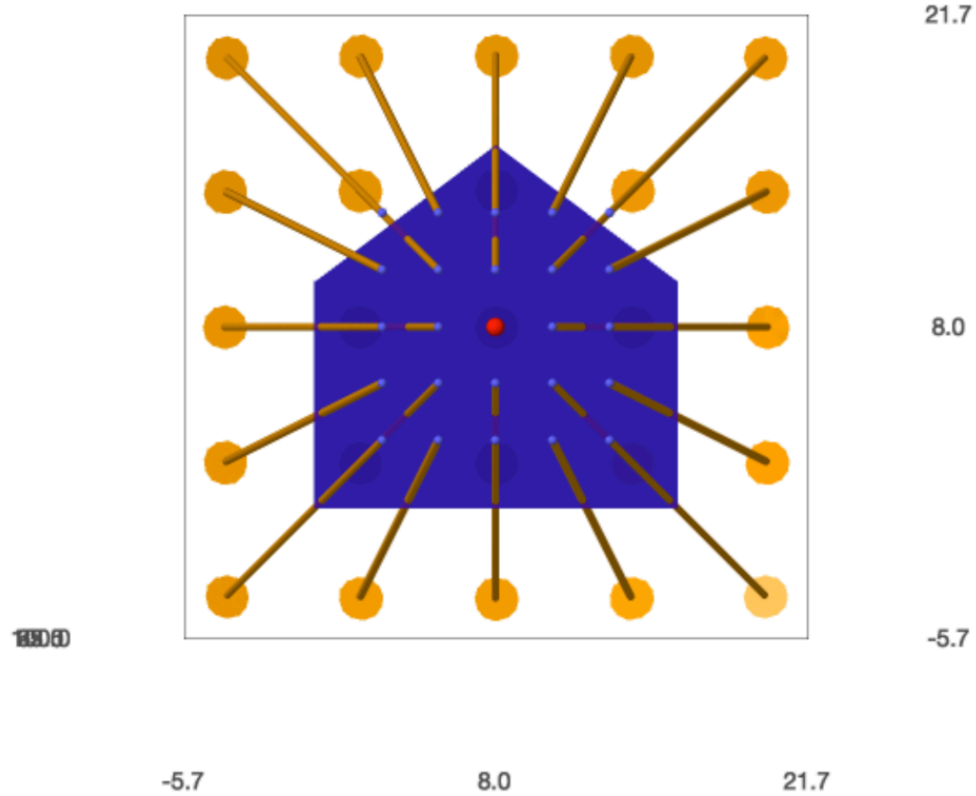
Example 2: Zoom

```
# 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





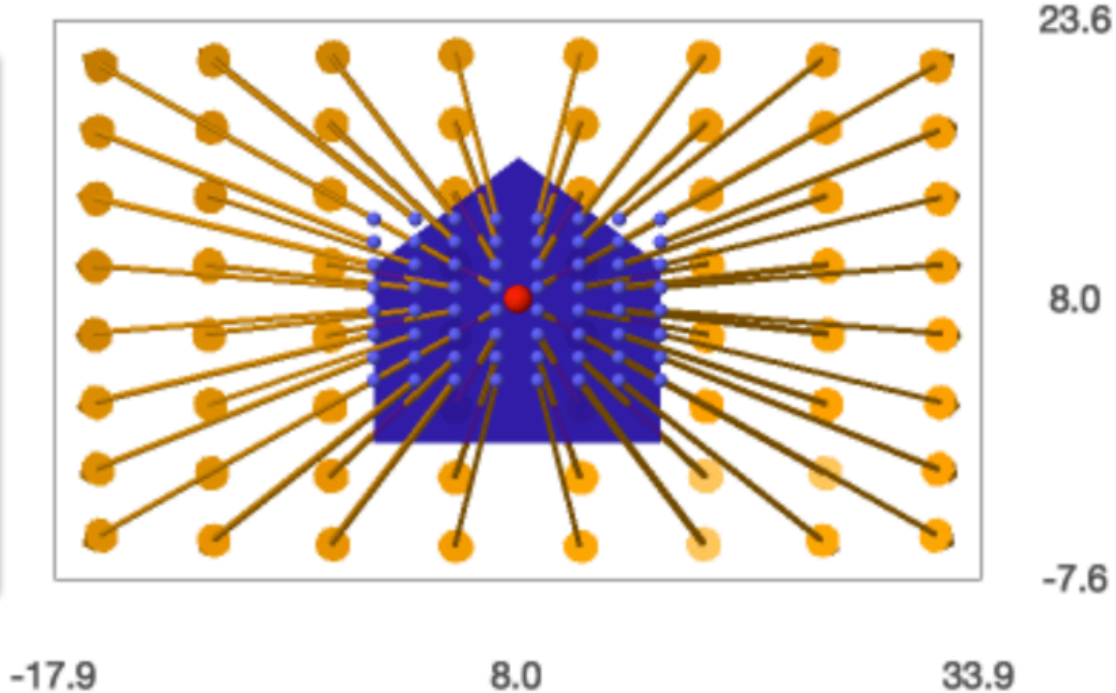
Do NOT confuse camera field of view with pixel density, i.e. the number of pixels in an image.

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;
```

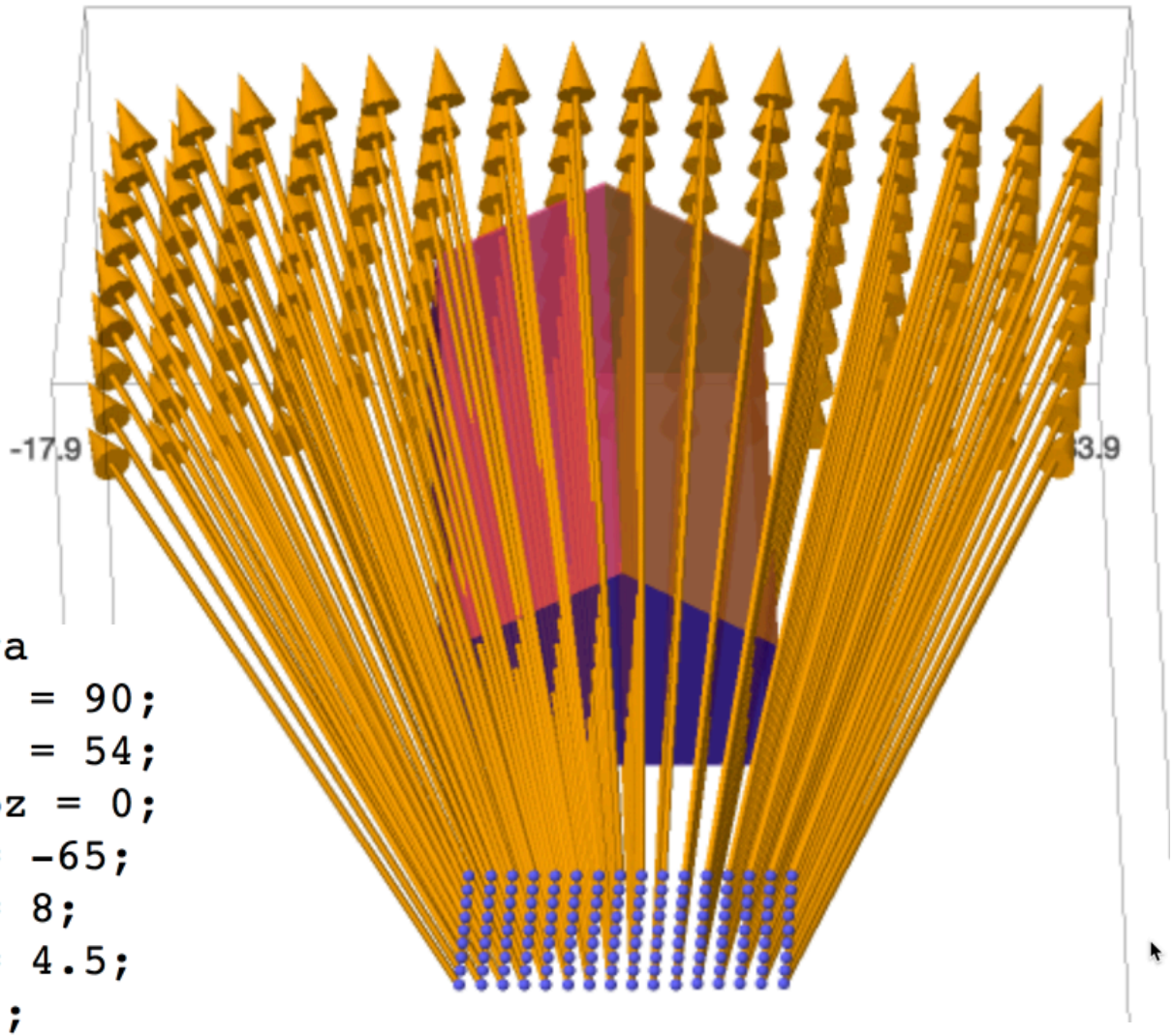
This is an 8x8 pixel image.

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



Example 5: Square is Good

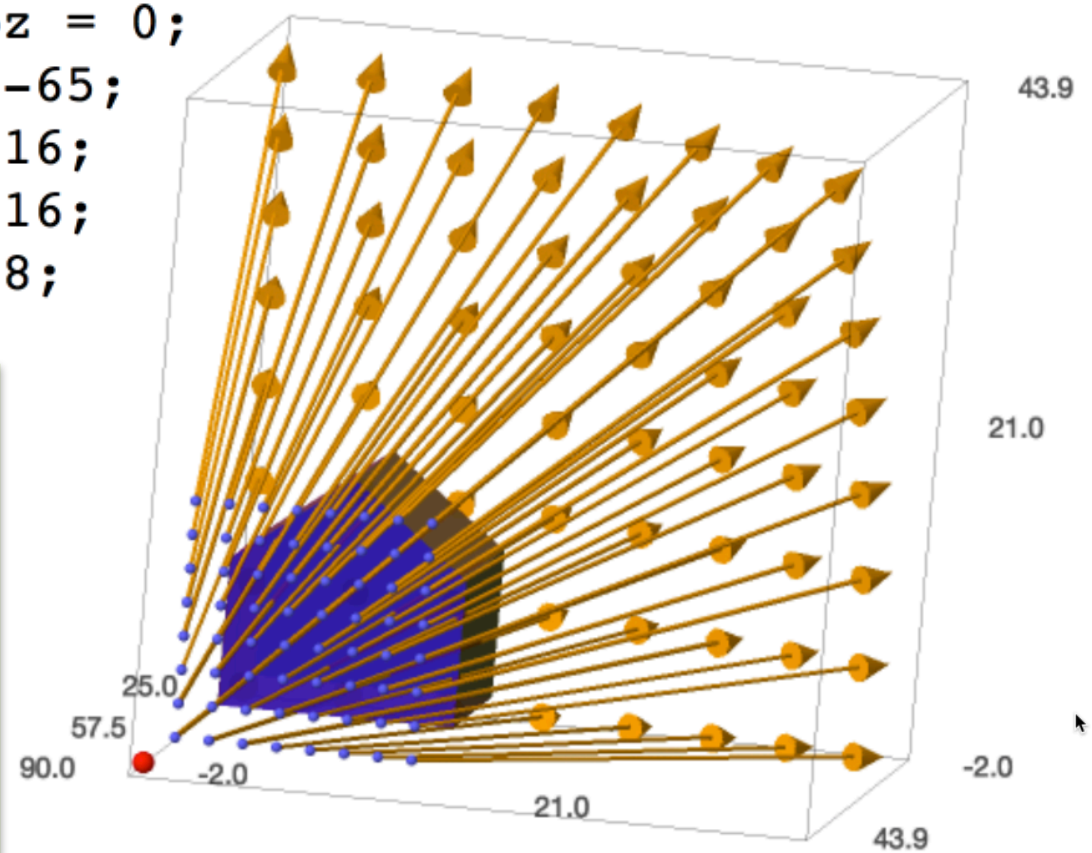
Square Pixels mean equally spaced samples horizontally and vertically.




```
# 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: Skewed Perspective

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



This is at best an uncommon case of camera geometry.

Where is the optical axis?

How are objects distorted?