
The Ancient
Secrets
Computer Vision

Logistics:

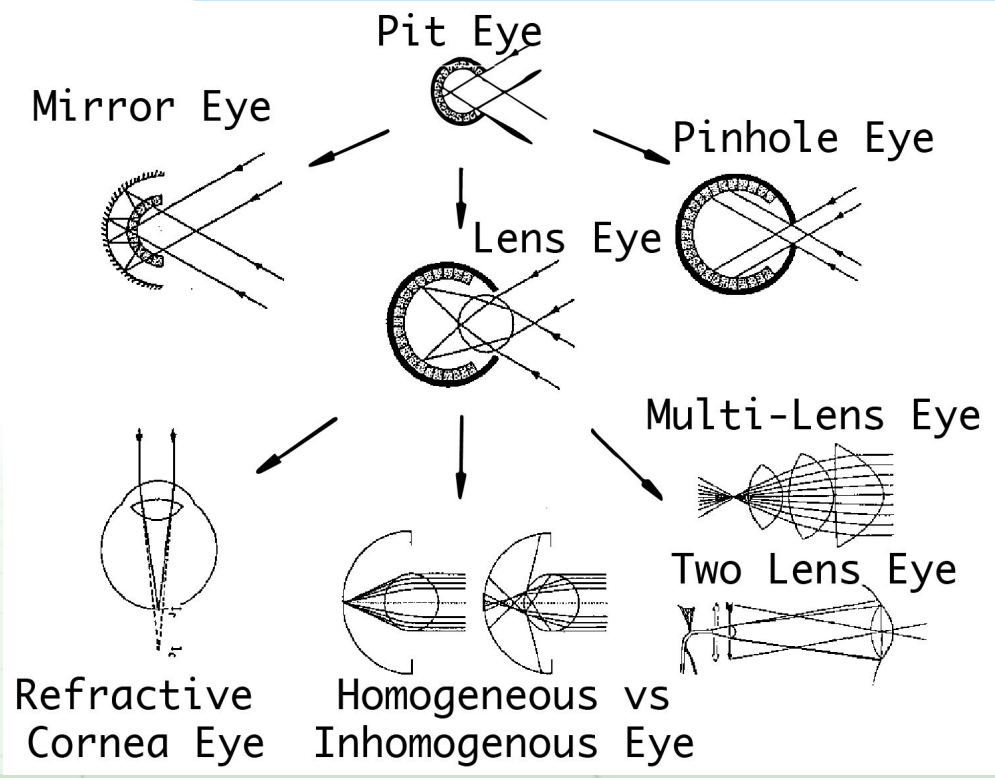
- Homework 1 is out!
  - Due on Thursday
  - Should be pretty straightforward, get you used to the image framework you'll be building
  - Start now so you can let us know if you have trouble
- Wednesday office hours with Nancy changed
  - 4-5pm, CSE 007
  - Just this week


Previously
On
the
Ancient Secrets
of Computer Vision

Vision is good!

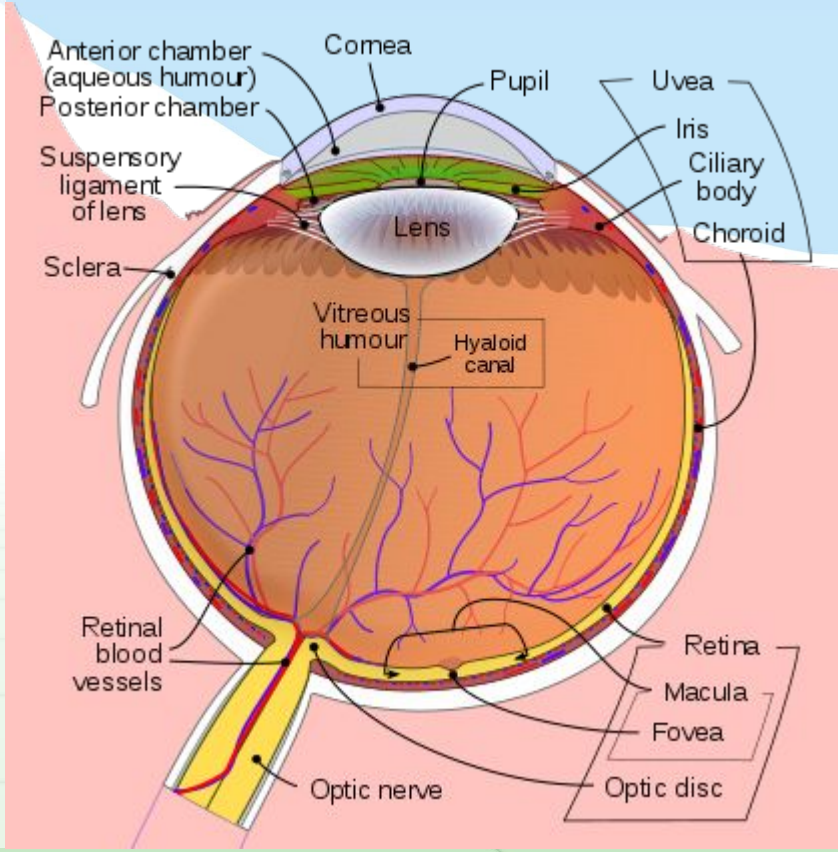
Simple eyes:28 of 33 animal phyla

Complex eyes: 6 of 33 phyla, 96% of species!



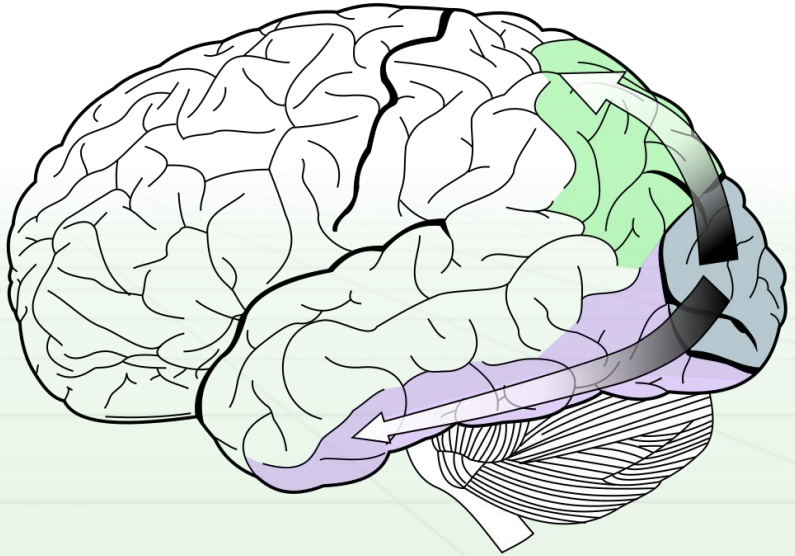
Human eyes

- Light passes through
  - Cornea, humours, lens refract light to focus
- Hit the retina
- Absorbed by rods, cones
- Info transmitted through optic nerve, processed by visual cortex

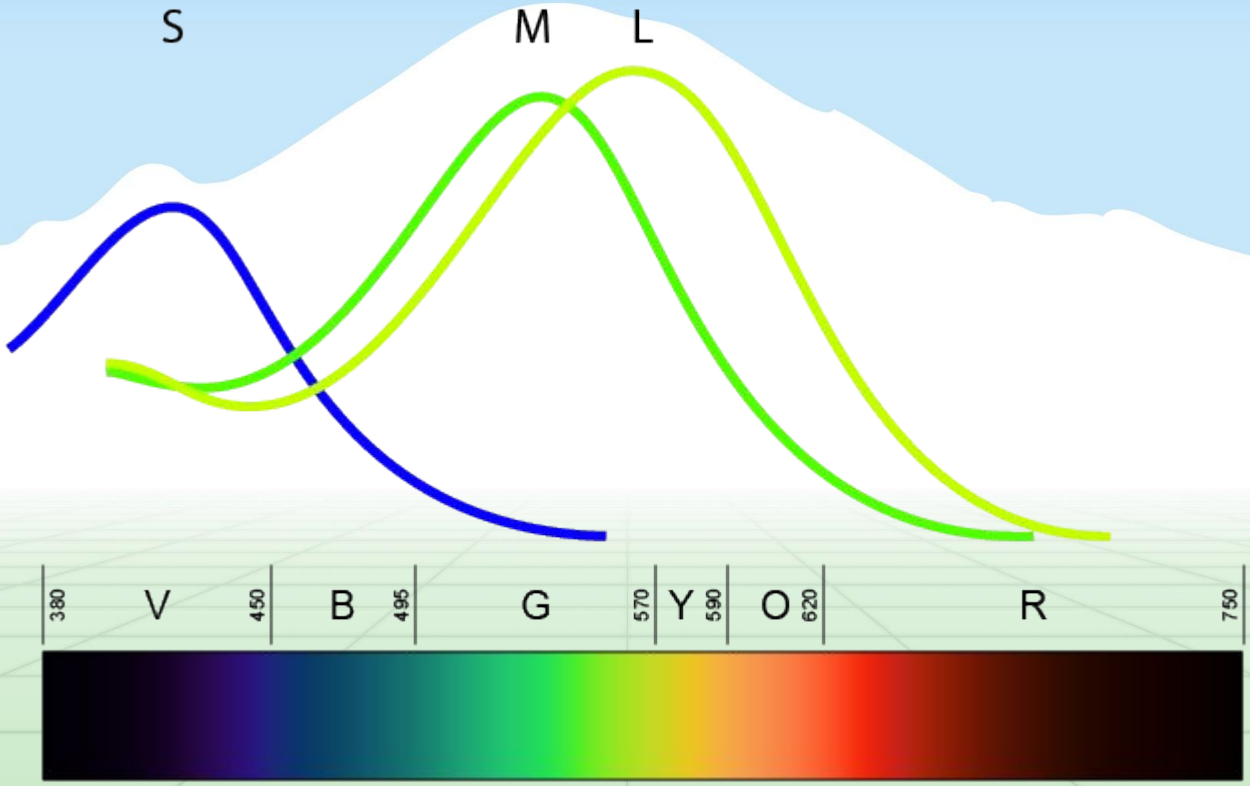


Ventral/dorsal split

- Ventral: fine grained visual recognition
- Dorsal: vision related to motion and planning
- The brain is integral to human vision, even when we are conscious of it!

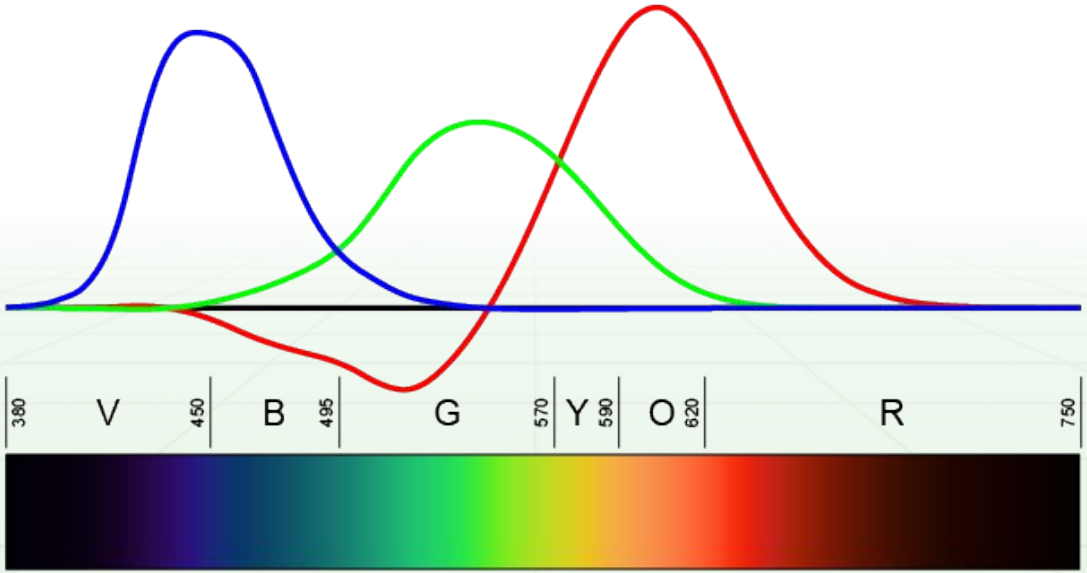


Rods and Cones: Color!



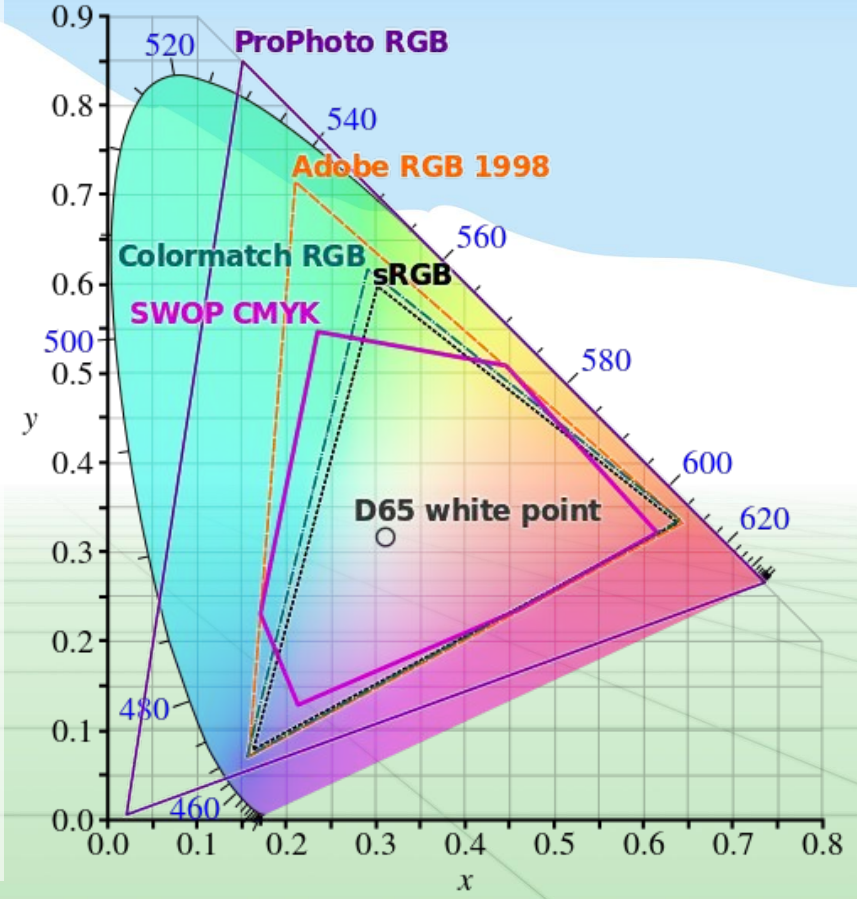
CIE 1931 color matching

- Match single wavelength with mix of primaries
- Most participants had similar mixes

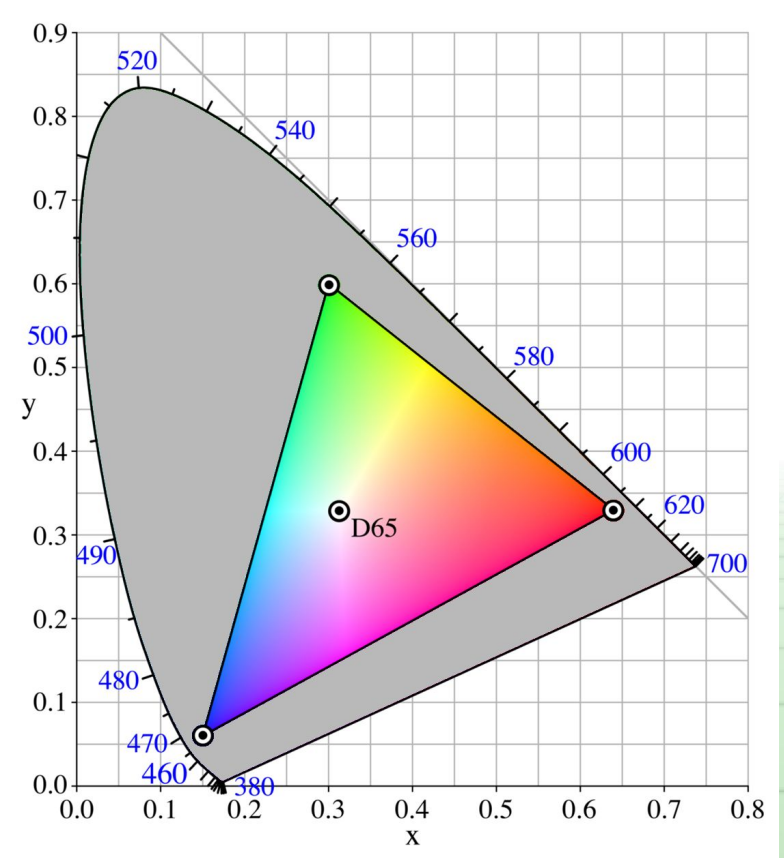


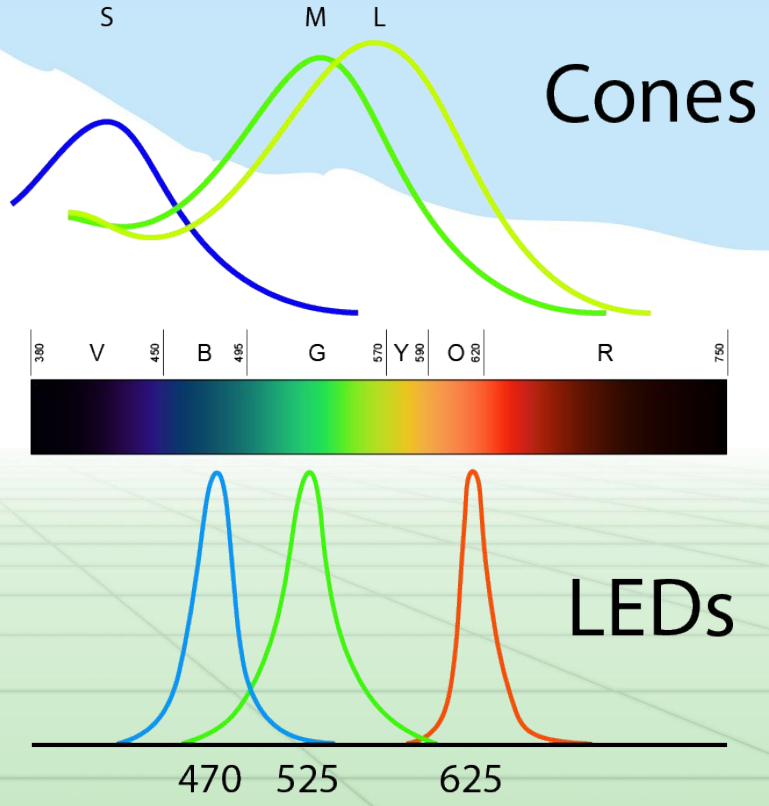
Can make a “map” of color

- Can make new colorspace by choosing primaries
- Range of representable colors: gamut
- Not all colors are representable! Have to go outside sometimes…



sRGB: most widely used colorspace

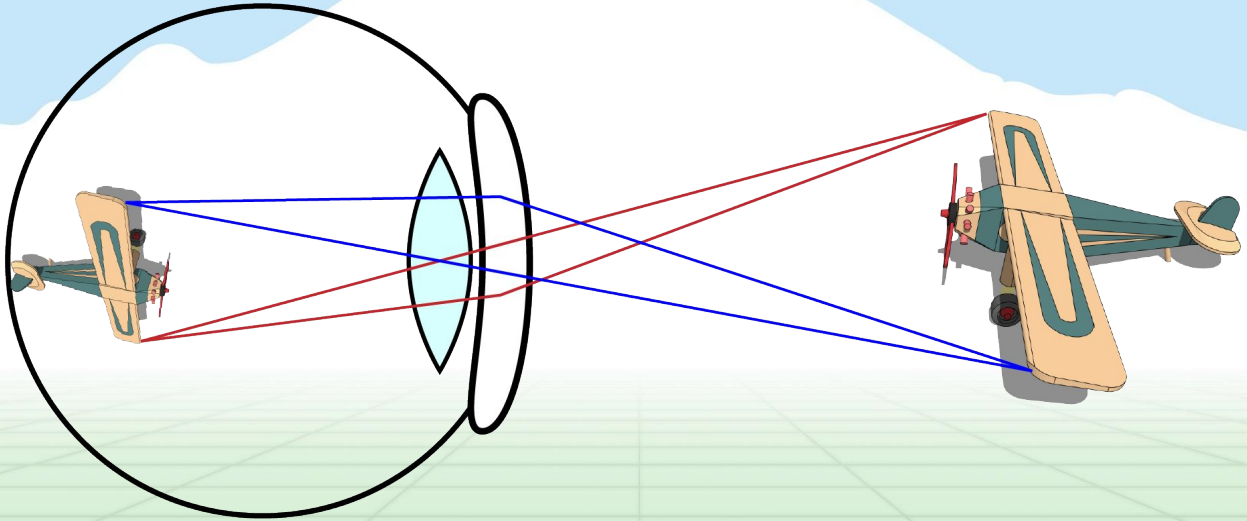




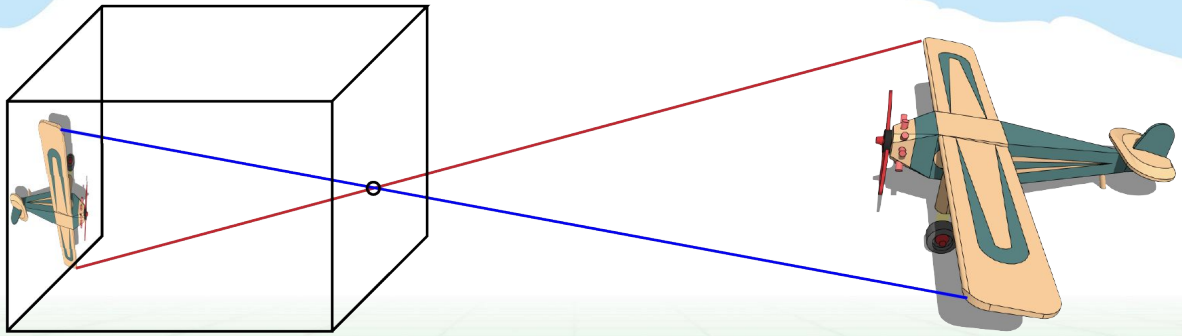

Chapter Three
Image Basics

So what is an image?

Eyes: projection onto retina



Model: pinhole camera



Model: pinhole camera

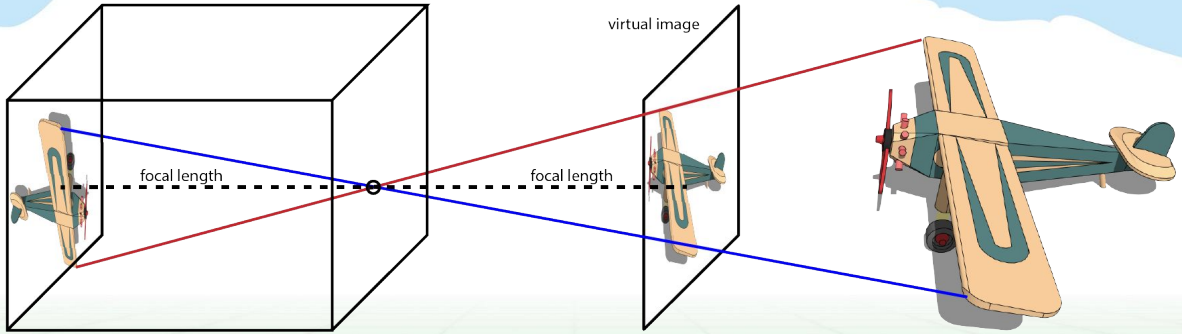


Image: 3d -> 2d projection of the world

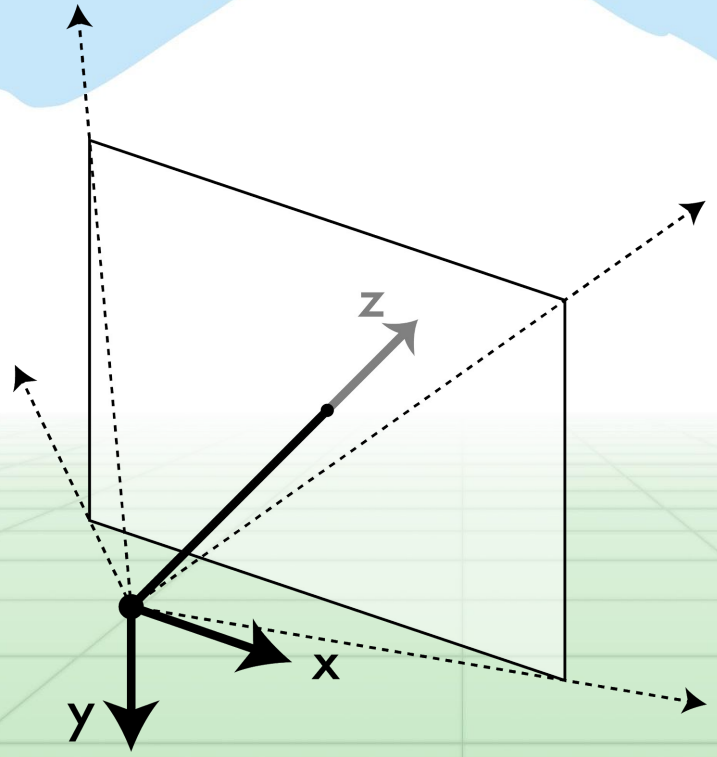


Image: 3d -> 2d projection of the world

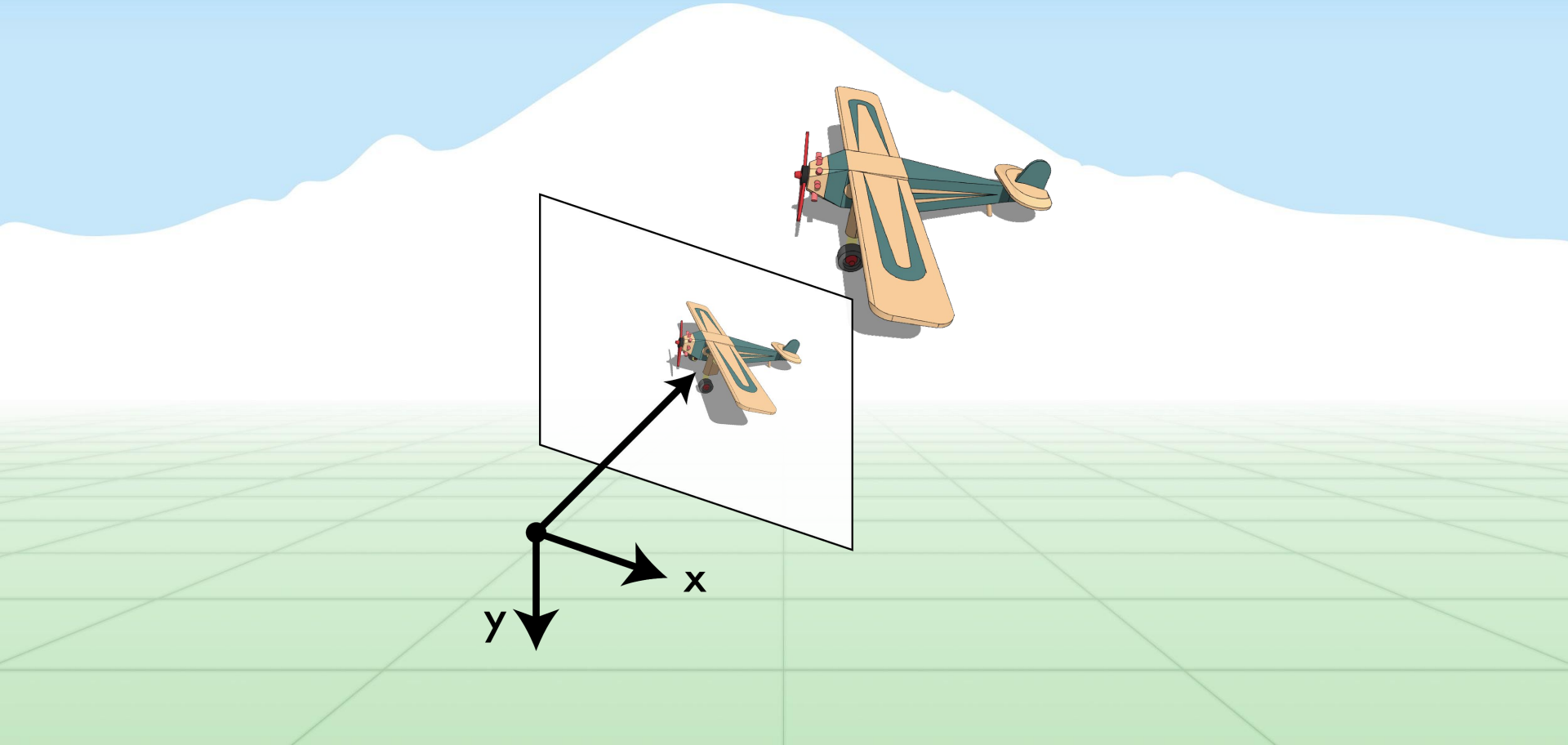


Image: 3d -> 2d projection of the world

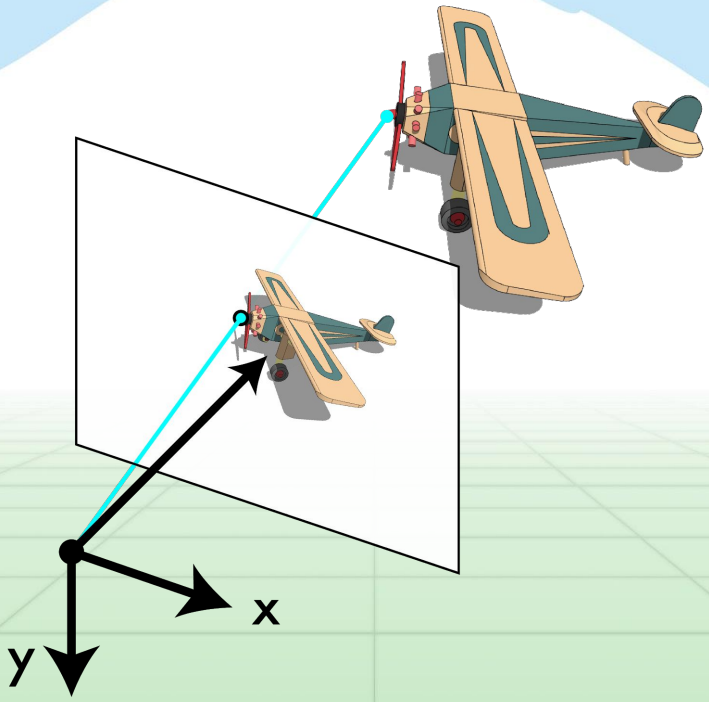


Image: 3d -> 2d projection of the world

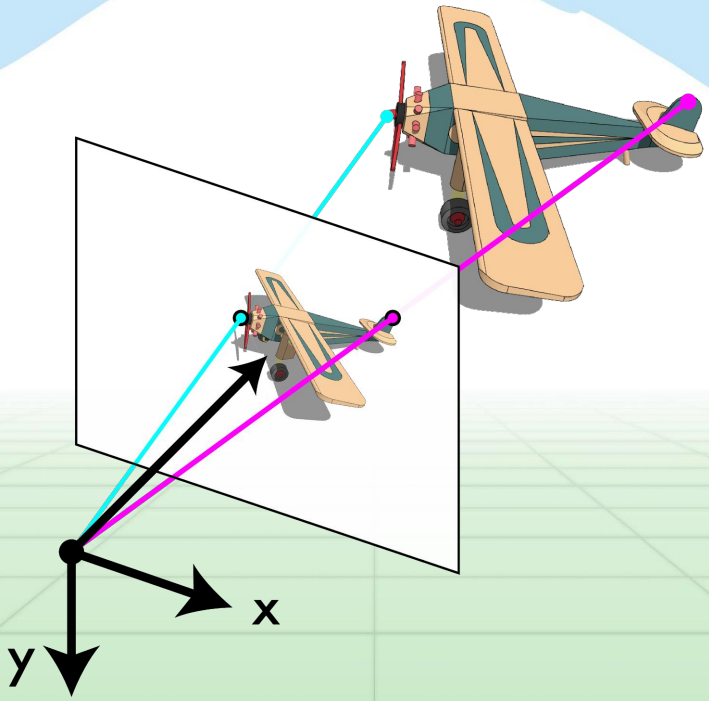


Image: 3d -> 2d projection of the world

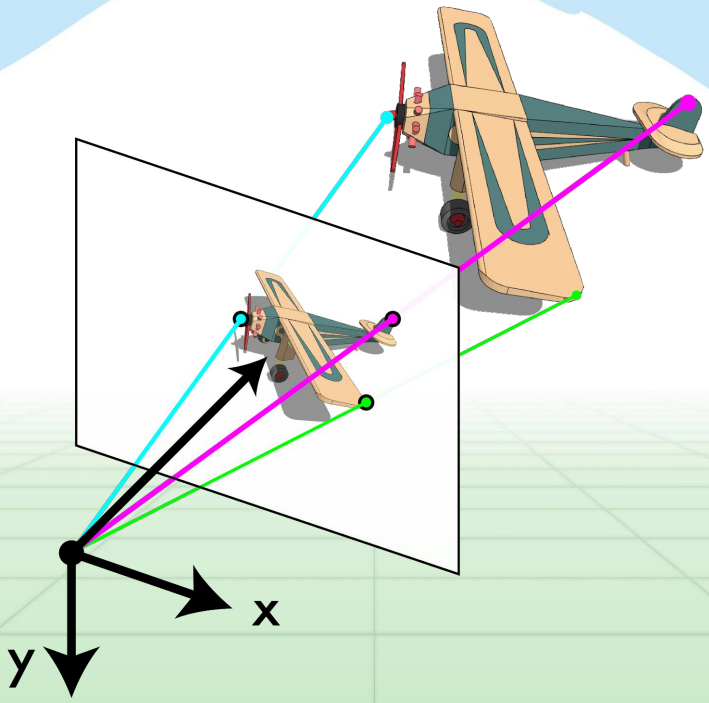
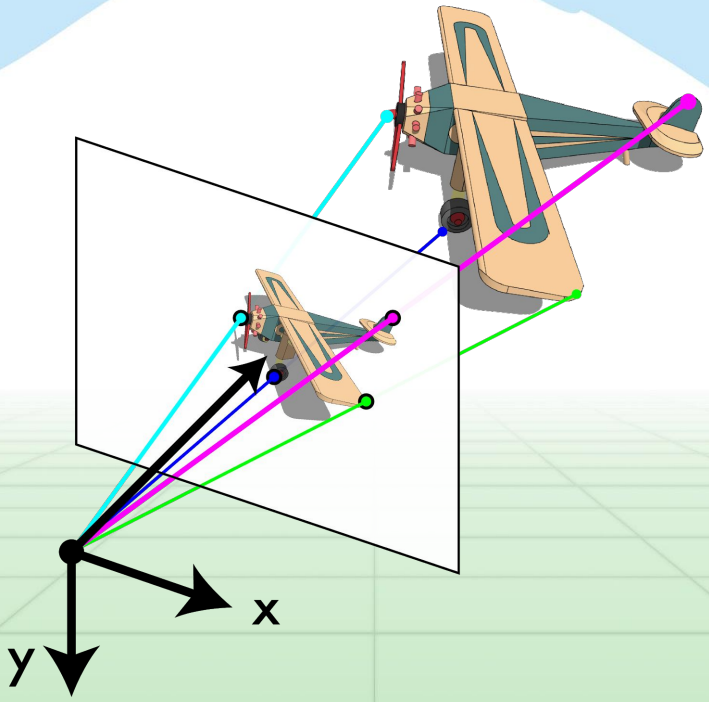
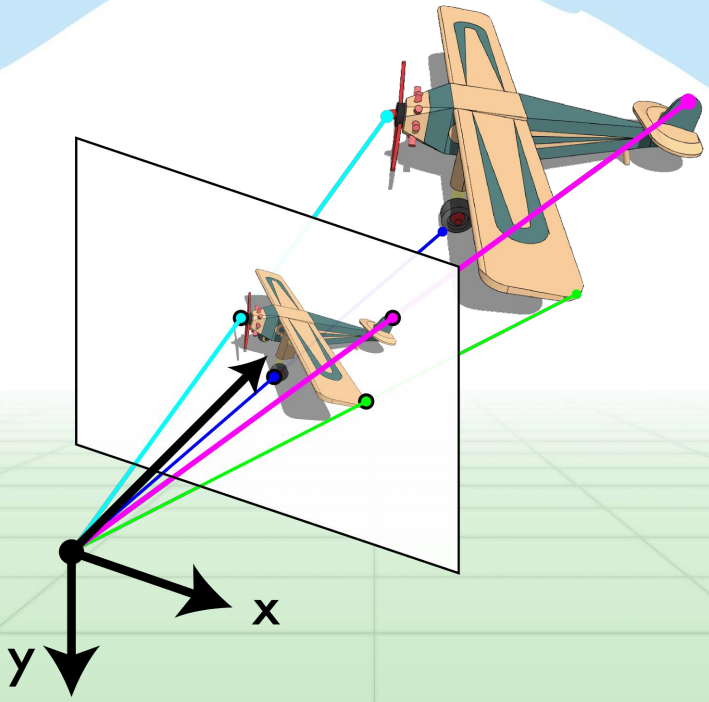


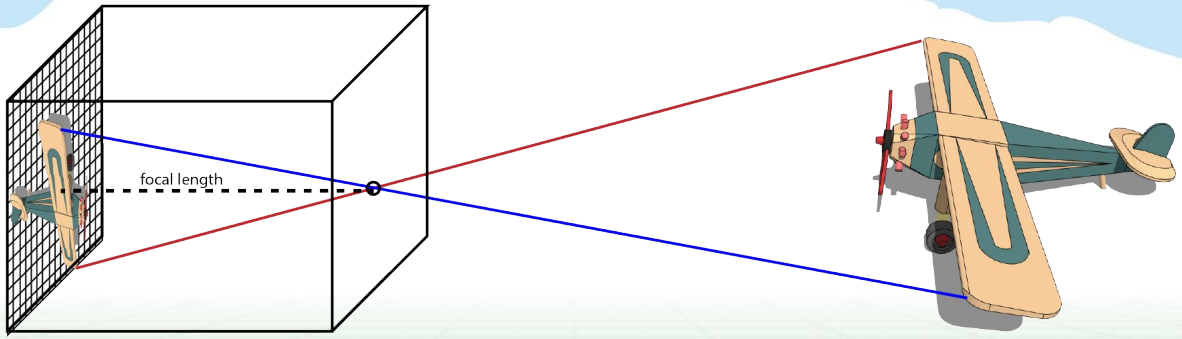
Image: 3d -> 2d projection of the world



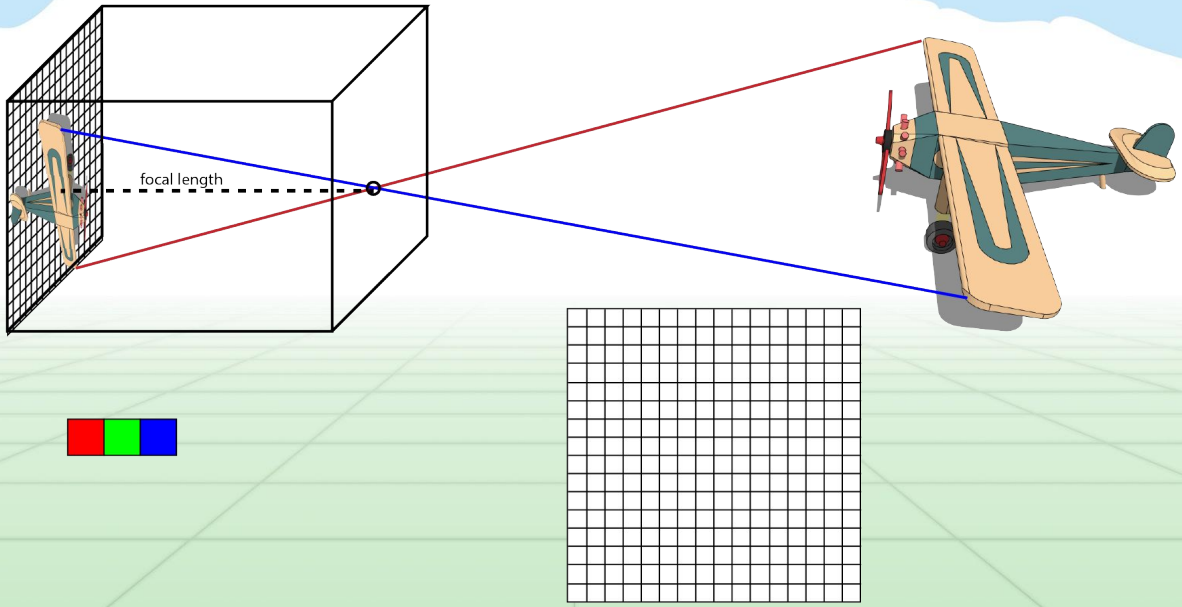
At each point we record incident light



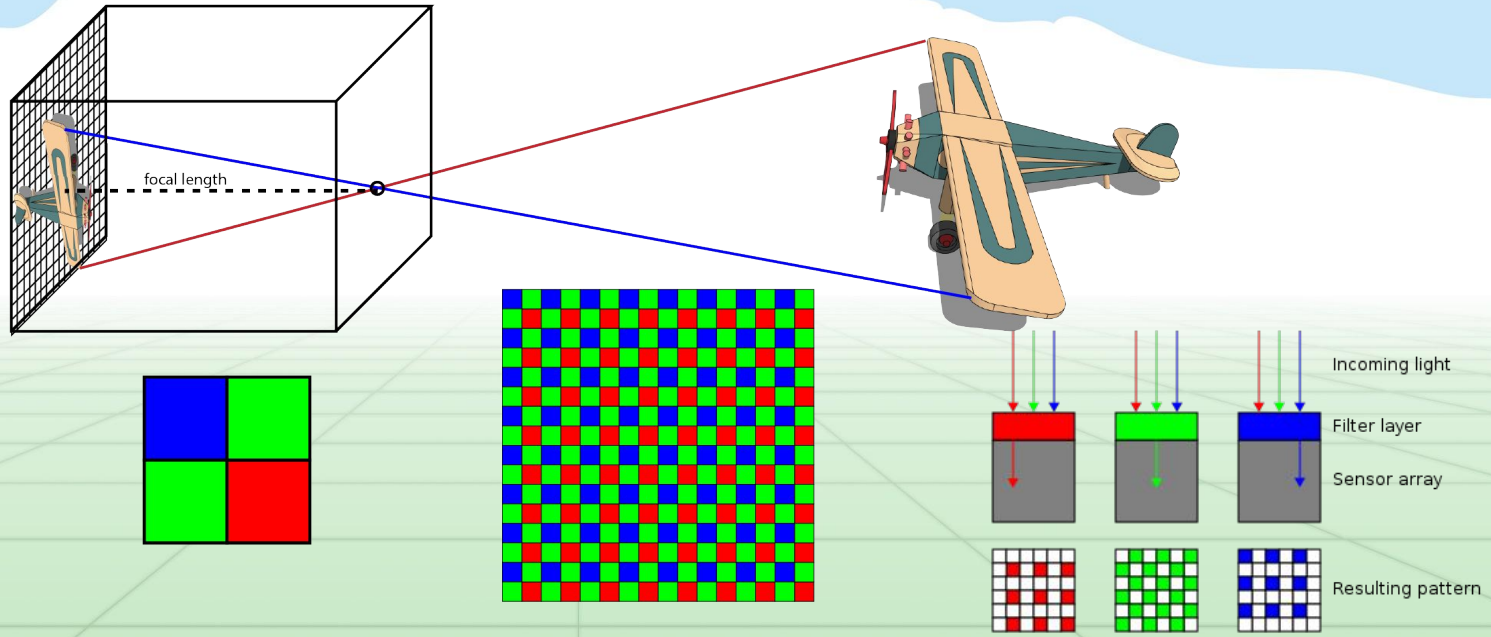
At each point we record incident light



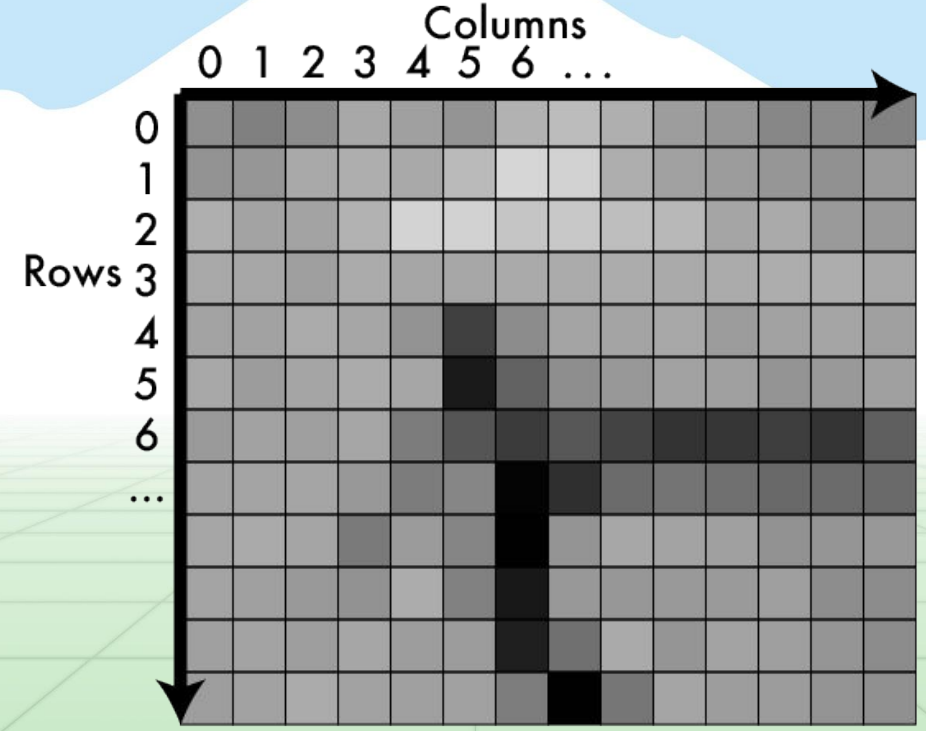
How do we record color?



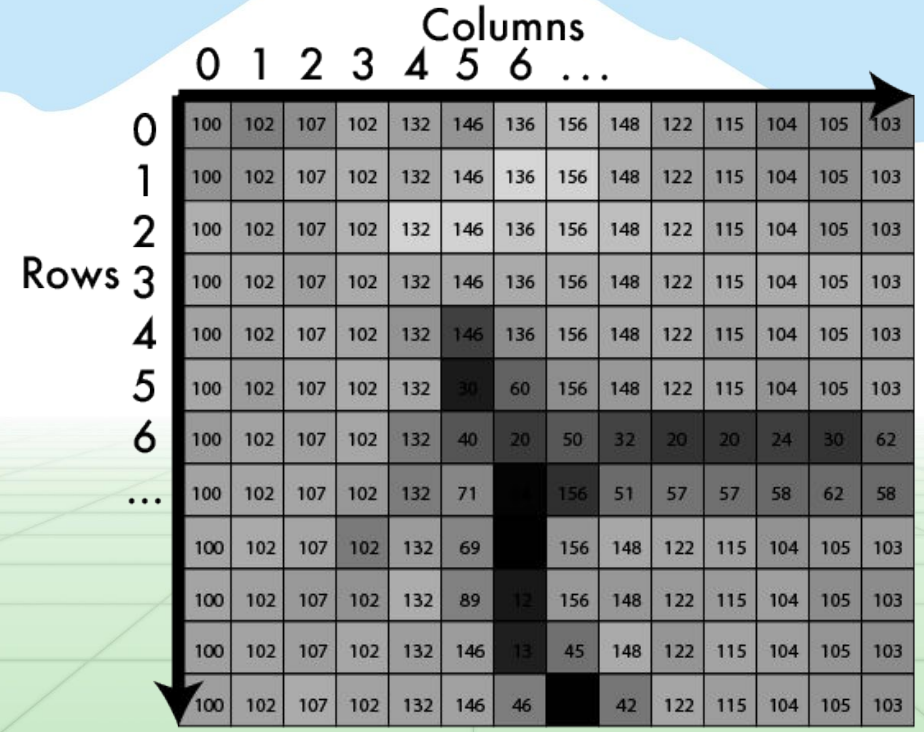
Bayer pattern for CMOS sensors



An image is a matrix of light

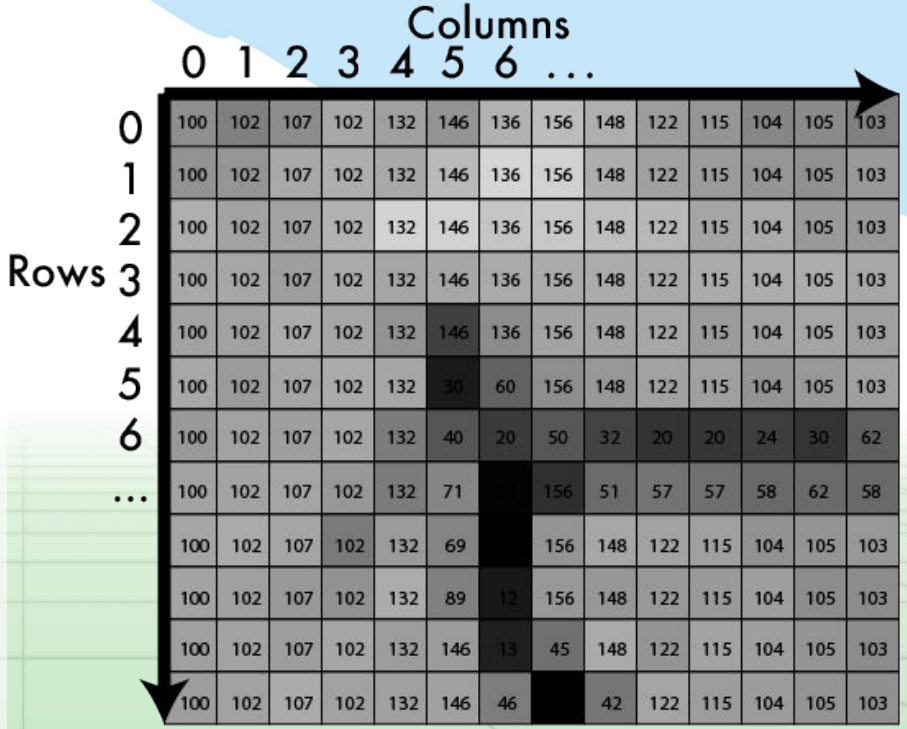


Values in matrix = how much light



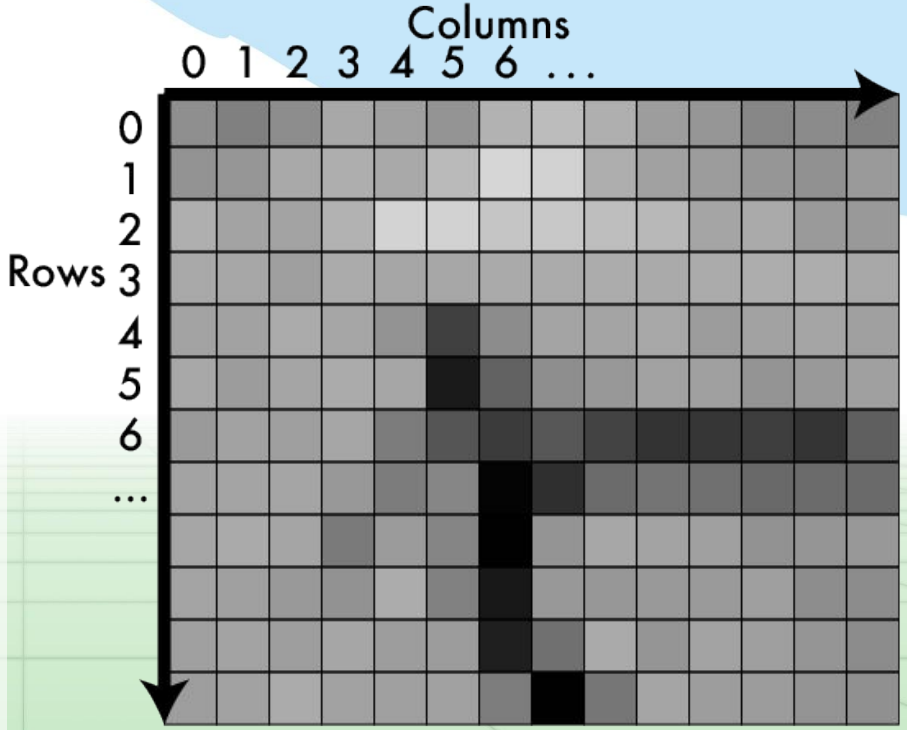
Values in matrix = how much light

- Higher = more light
- Lower = less light
- Bounded
  - No light = 0
  - Sensor/device limit = max
  - Typical ranges:
    - [0-255], fit into byte
    - [0-1], floating point
- Called pixels

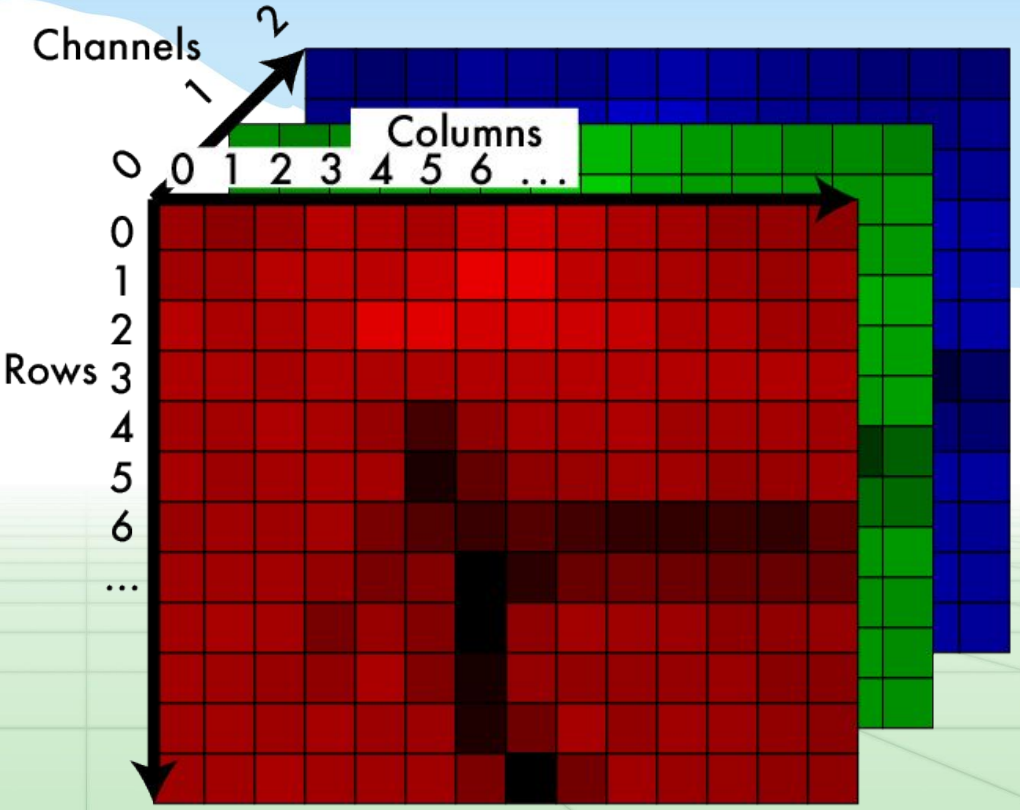


Addressing pixels

- Ways to index:
  - (x,y)
    - Like cartesian coordinates
    - (3,6) is column 3 row 6
  - (r,c)
    - Like matrix notation
    - (3,6) is row 3 column 6
- I use (x,y)
  - So does your homework!
  - Arbitrary
  - Only thing that matters is consistency



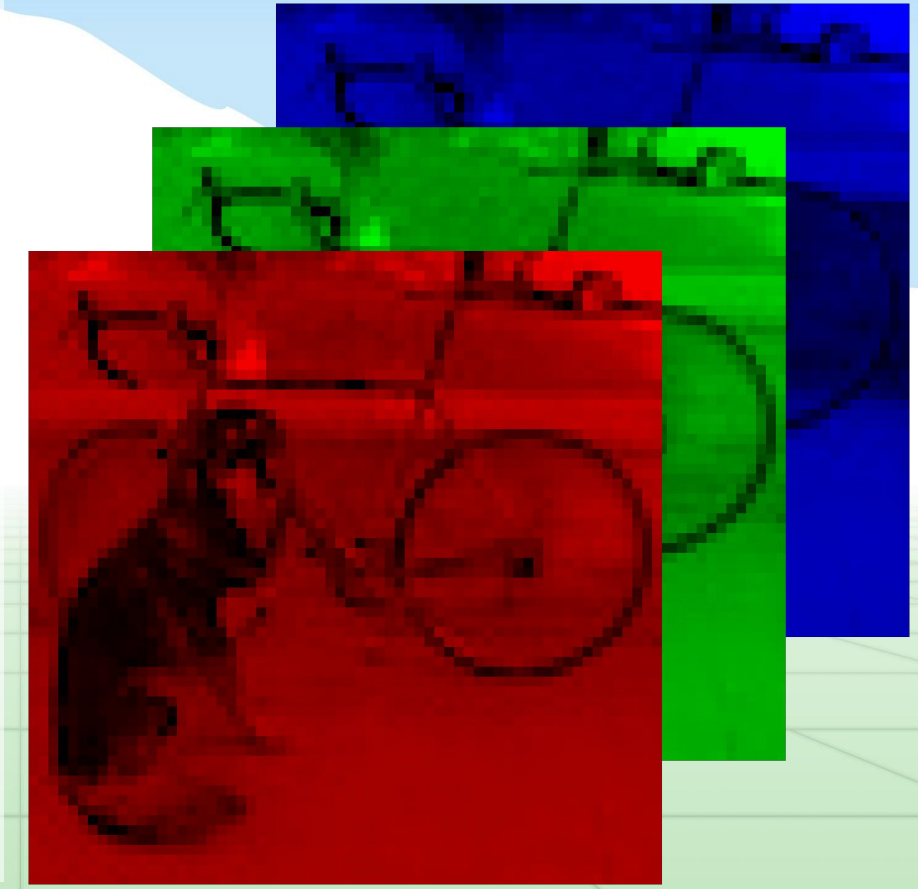
Color image: 3d tensor in colorspace



RGB information in separate “channels”

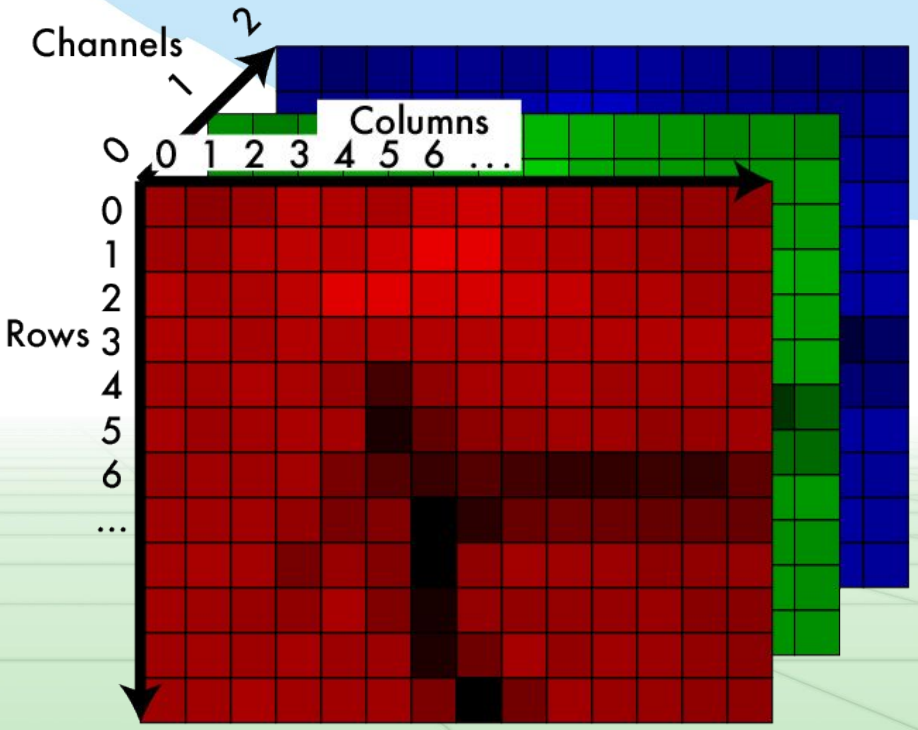
Remember: we can match “real” colors using a mix of primaries.

Each channel encodes one primary. Adding the light produced from each primary mimics the original color.

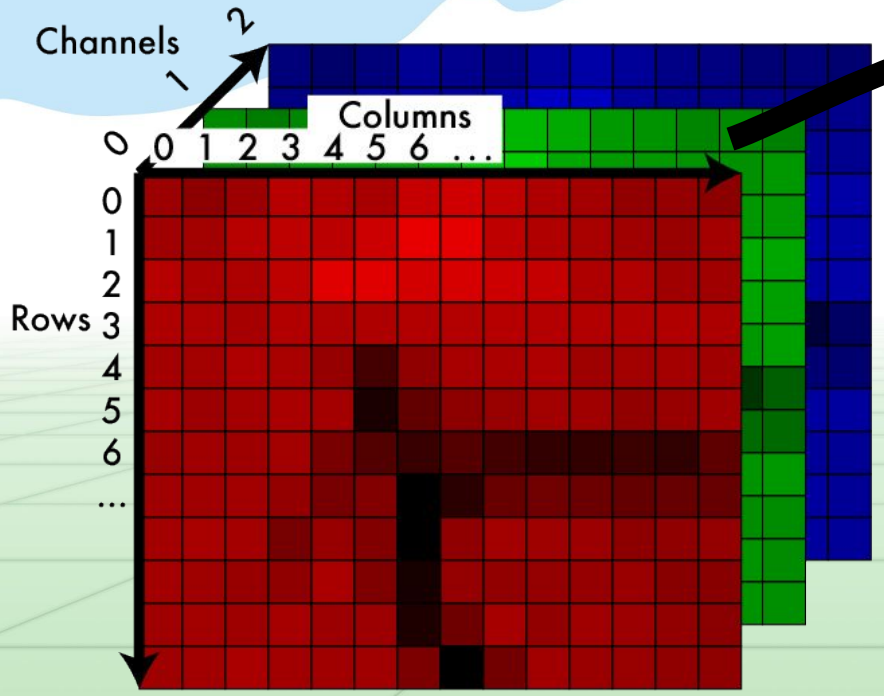


Addressing pixels

- I use (x,y,c)
  - (1,2,0):
    - column 1, row 2, channel 0
- Still doesn’t matter, just be consistent
- But do what I do for homeworks :-)
- Also for size:
  - 1920 x 1080 x 3 image:
    - 1920 px wide
    - 1080 px tall
    - 3 channels

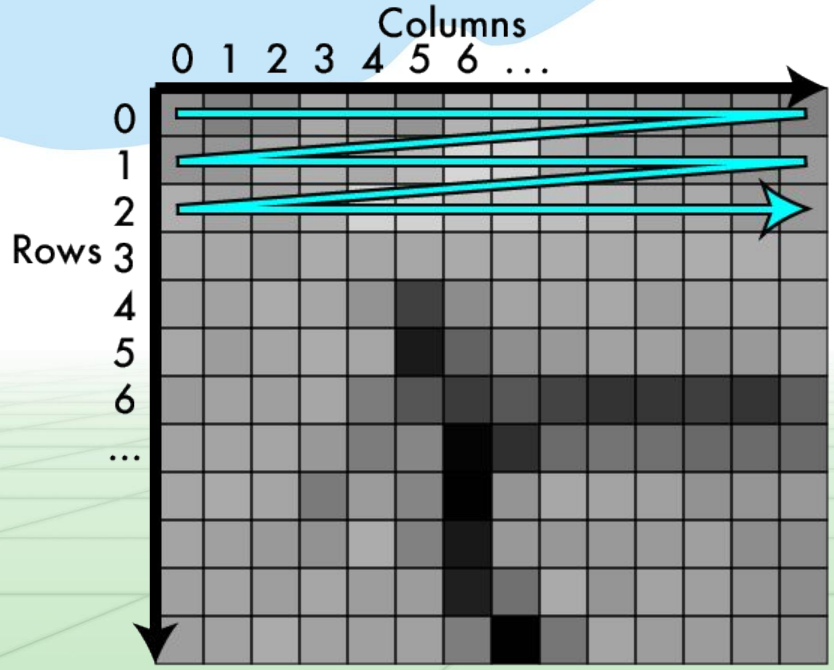


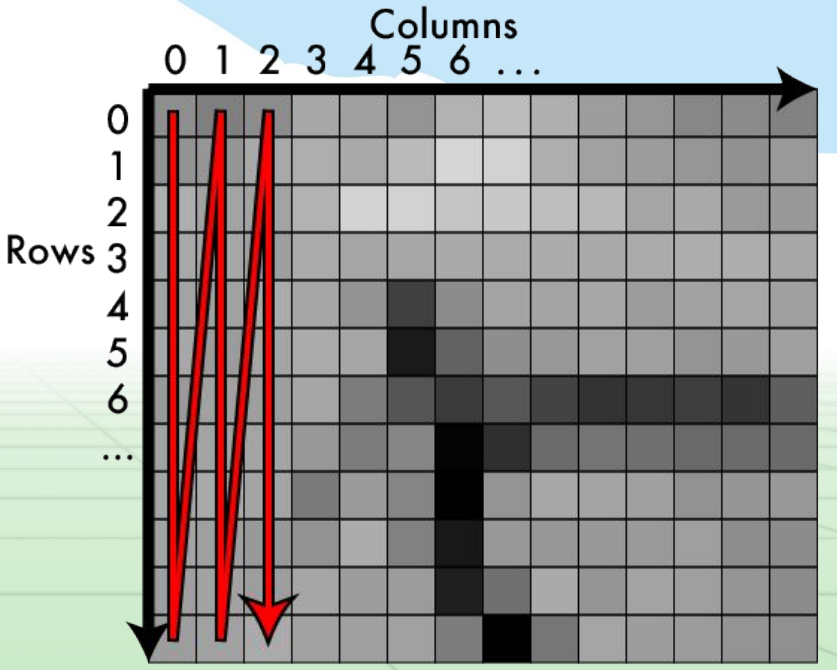
How do we store them?



129	131	152	159	135	163	142	82	182	...

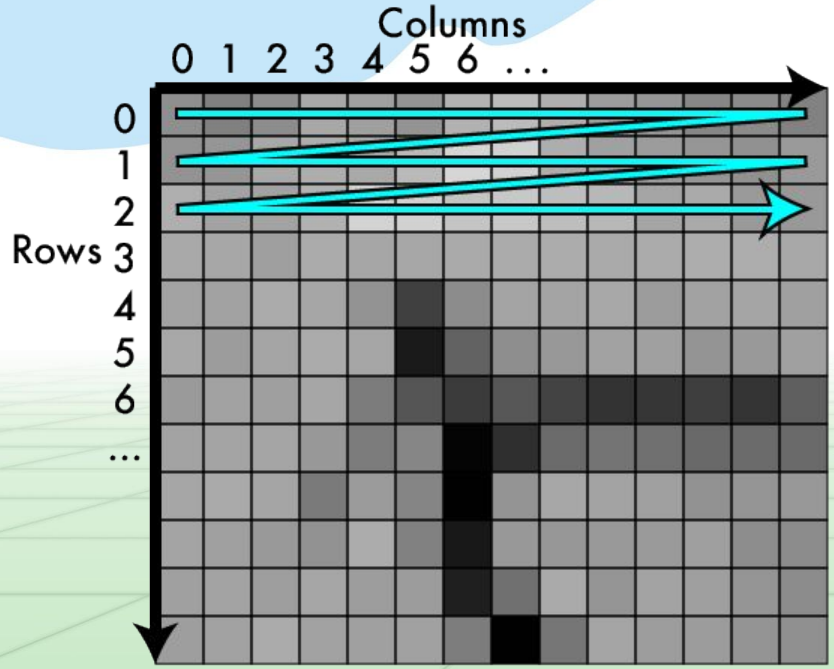
Storage: row major vs column major



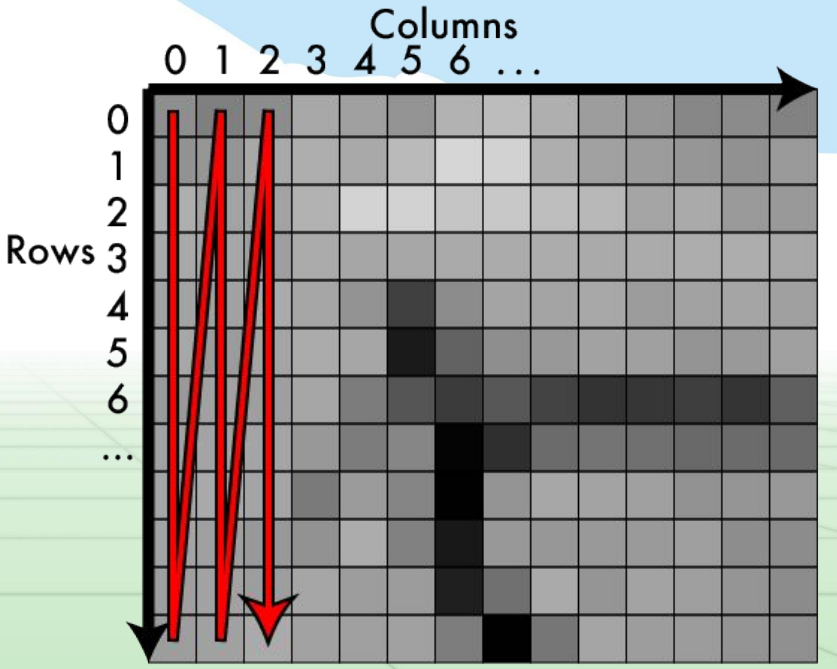


Storage: row major vs column major

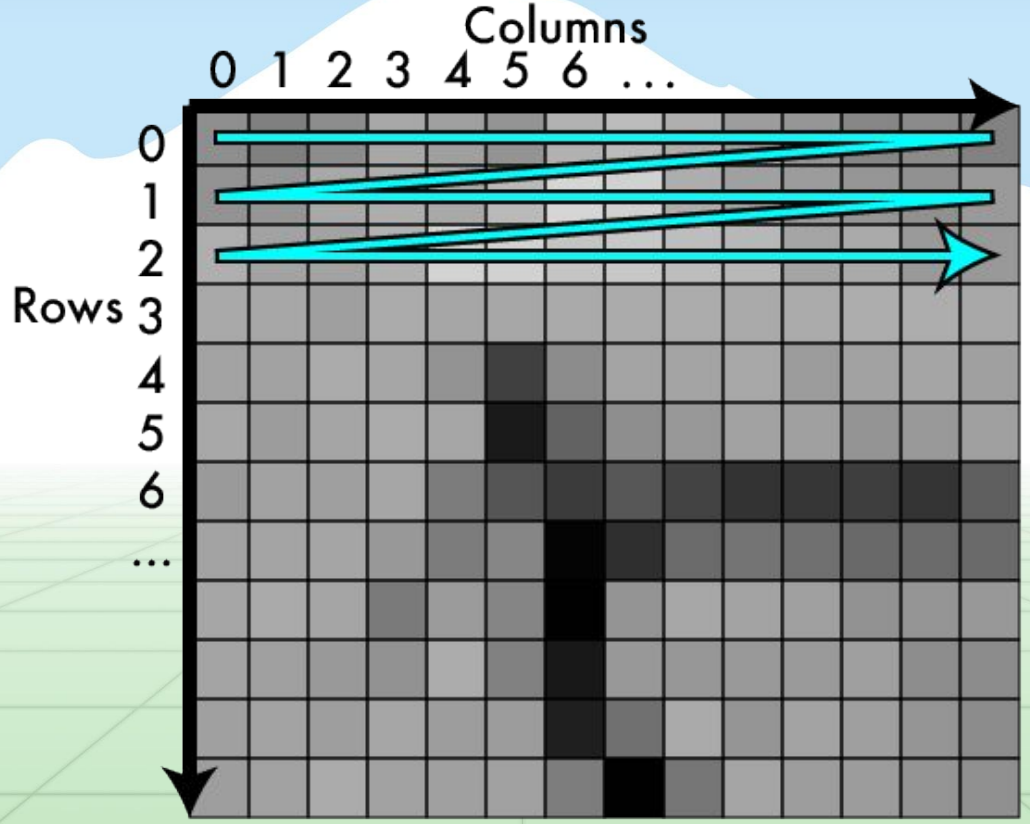
HW



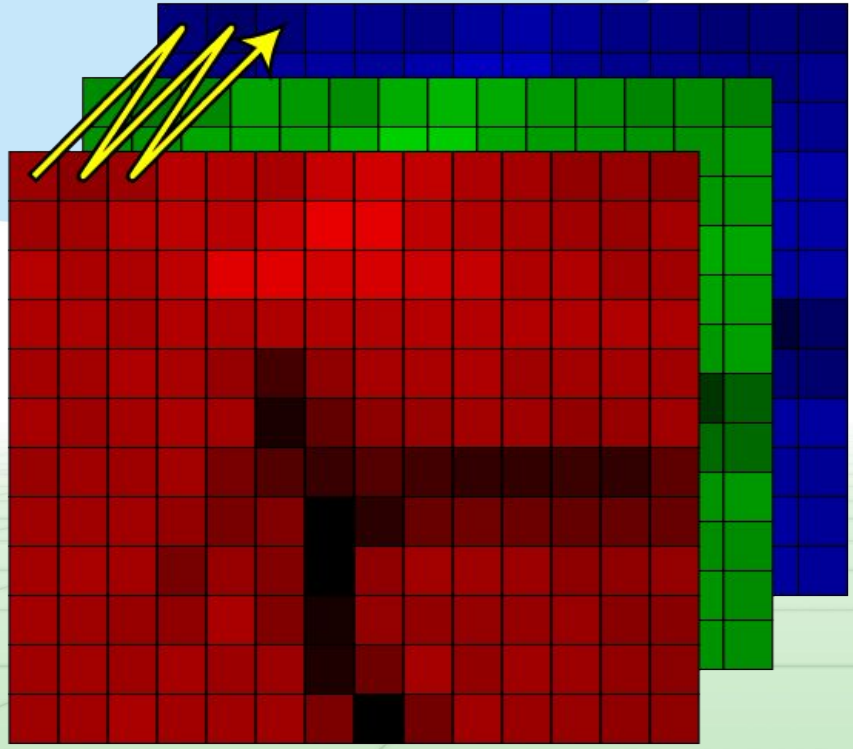
WH

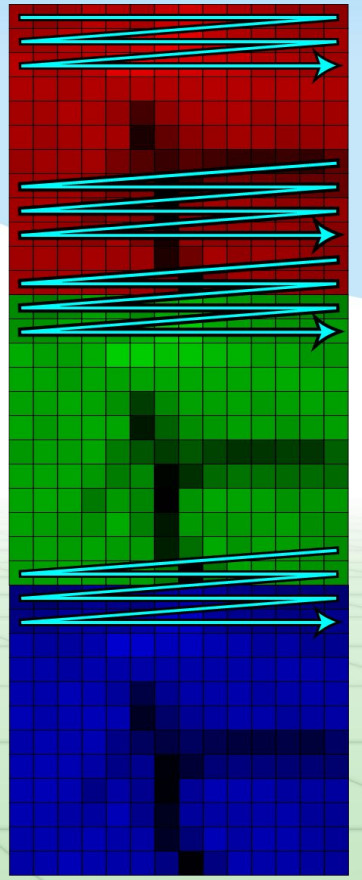


Typically use row-major or HW

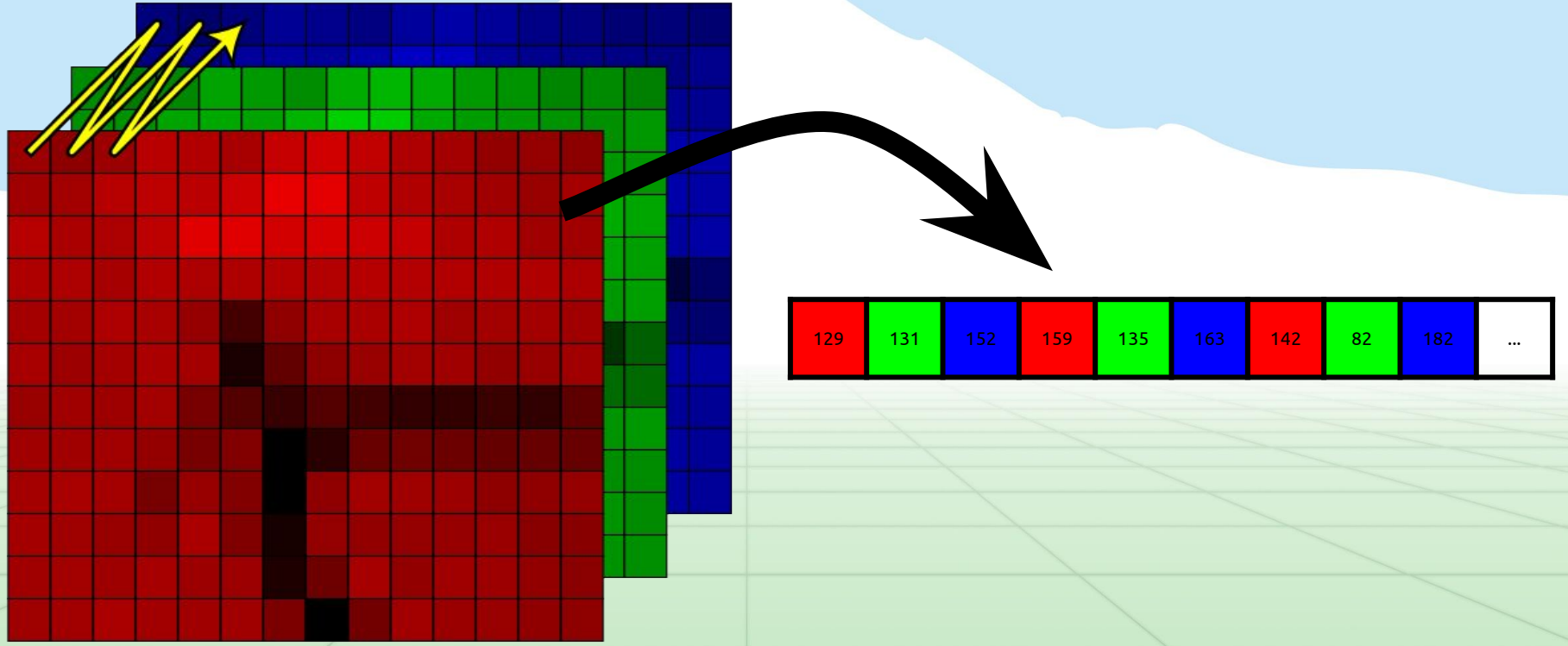


In 3d we have more choices!

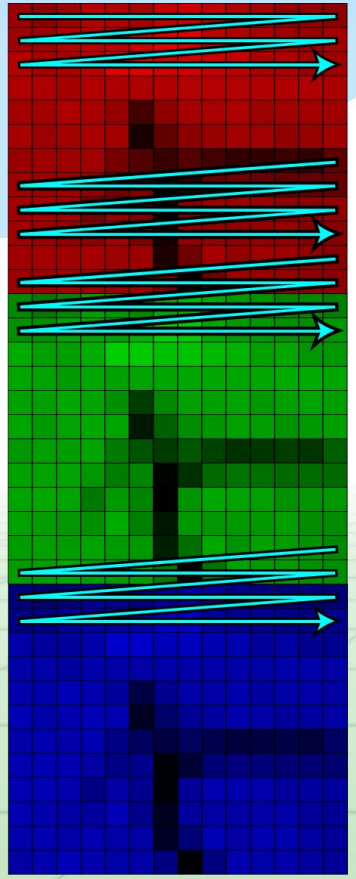


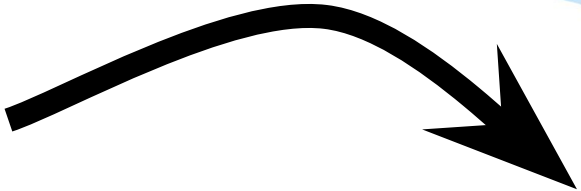


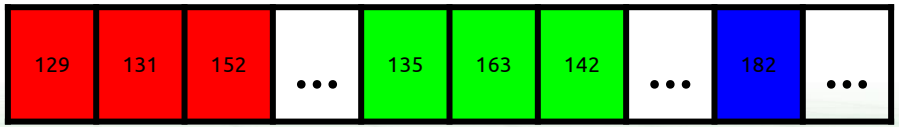
HWC: channels interleaved



CHW: channels separated





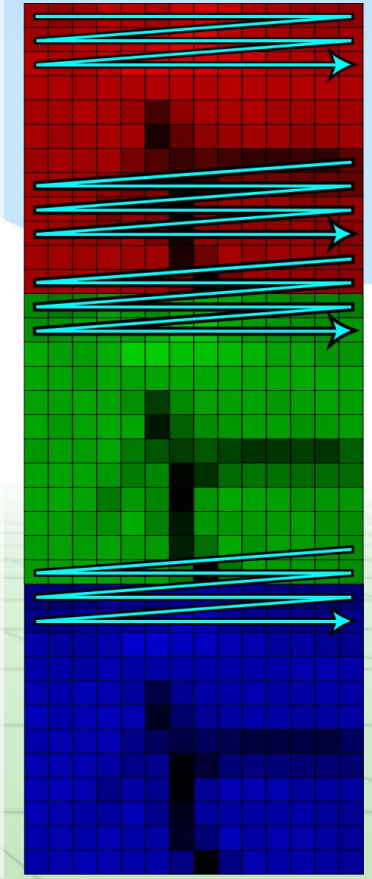


CHW Pop quiz

We'll use CHW, it's what a lot of other libraries use.

In an array for a 1920 x 1080 x 3 image what entry would contain the pixel (15,192,2)?

In groups, discuss for 2 minutes.



CHW Pop quiz

In an array for a 1920 x 1080 x 3 image what entry would contain the pixel (15,192,2)?

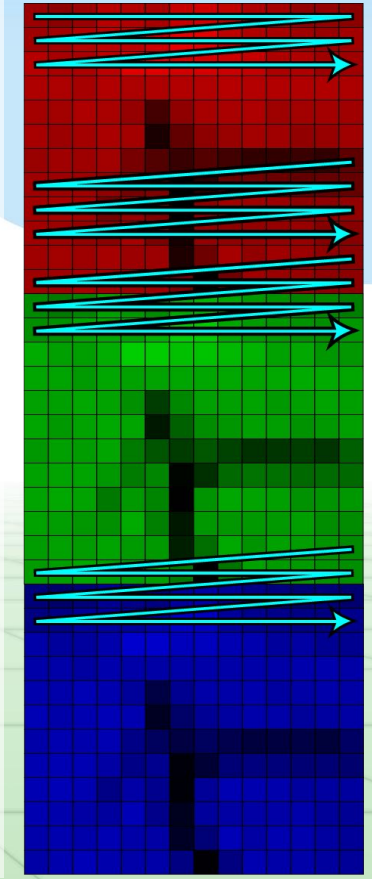
In general for (x,y,z) of image (W,H,C)

x + y*W + z*W*H

15 + 192*1920 + 2*1920*1080 = 4,515,855

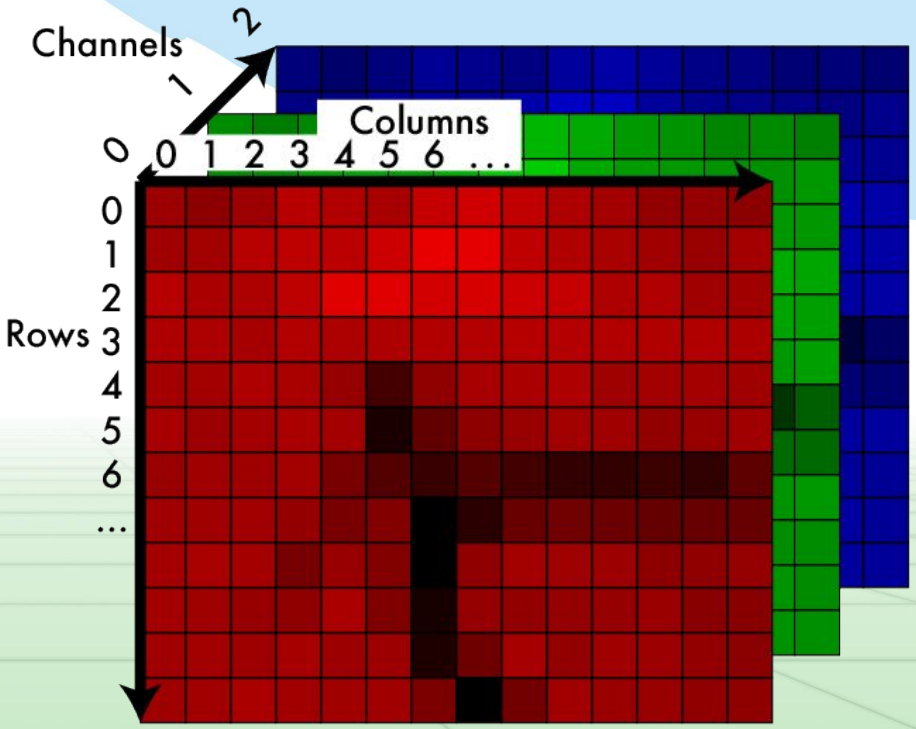
Remember, everything is 0 indexed

This isn’t MATLAB 🙃



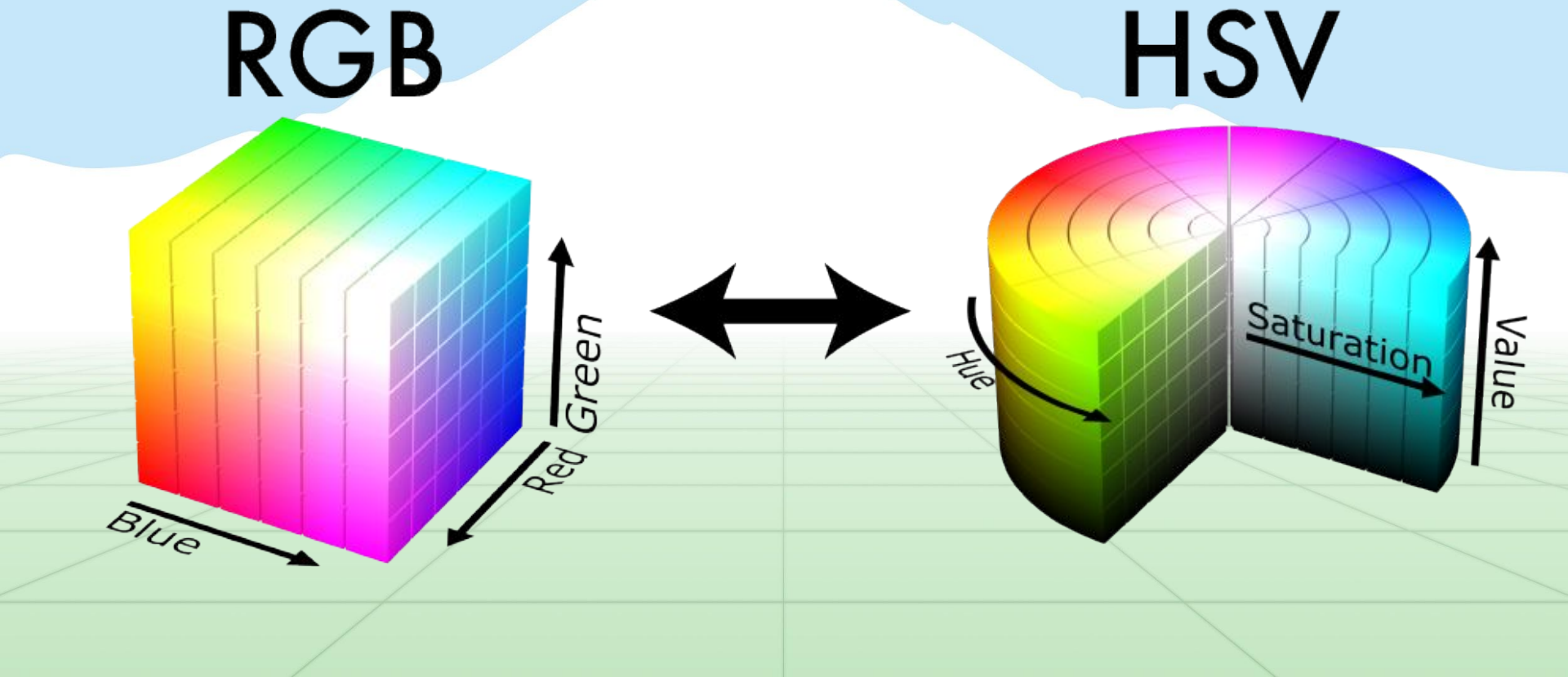
In your homework

```
typedef struct {
    int w,h,c;
    float *data;
} image;
```

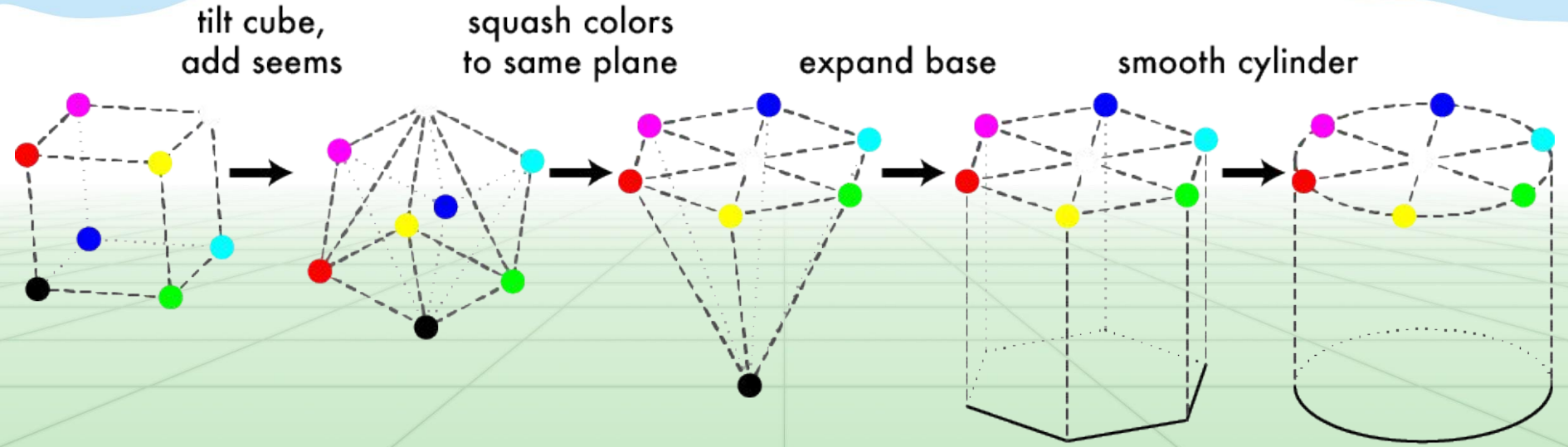


Fun with other colorspaces!

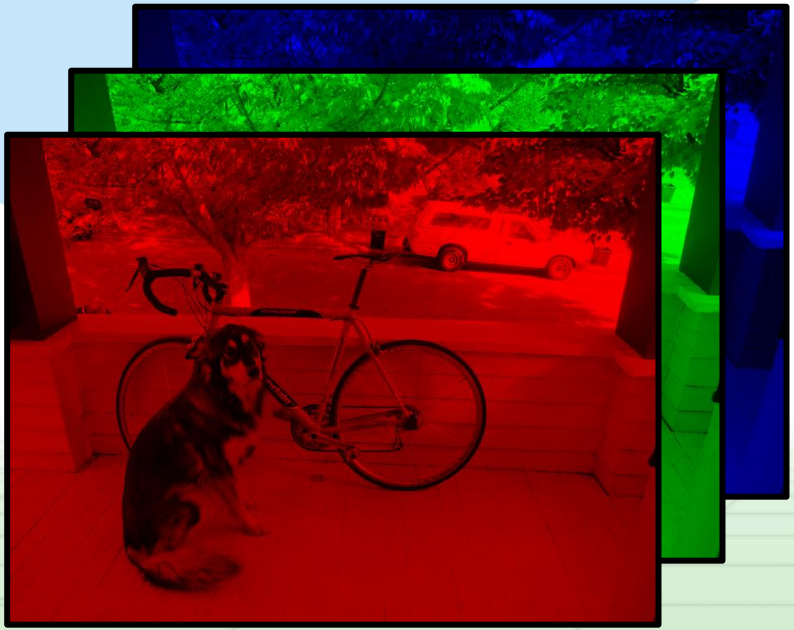
Other colorspaces are fun!

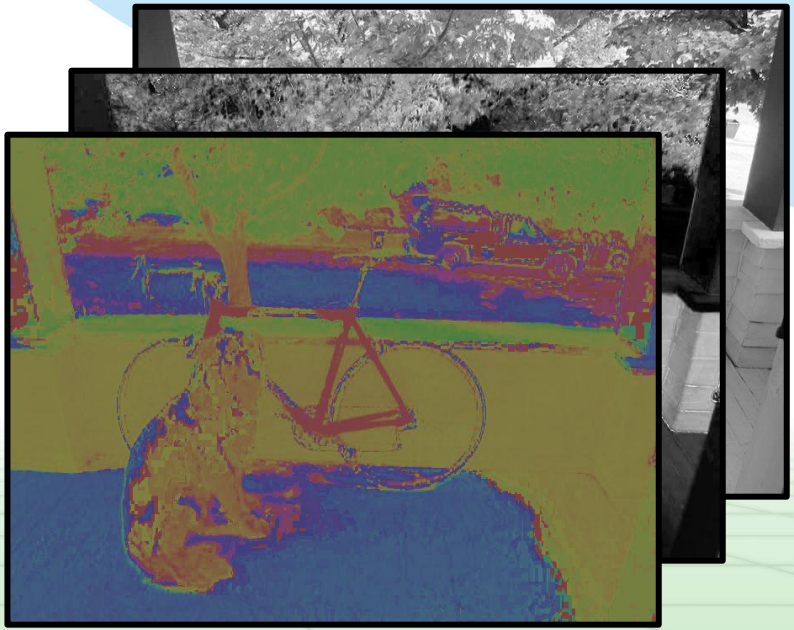


Geometric HSV to RGB:

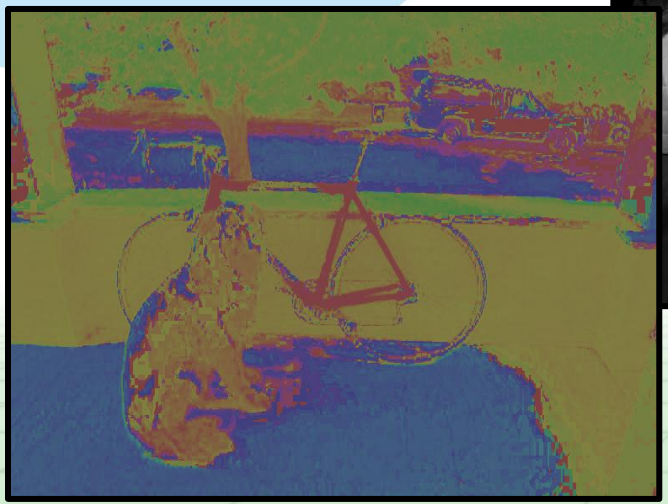


Still 3d tensor, different info

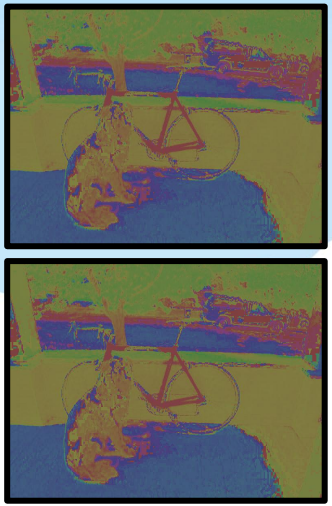




Hue Saturation Value

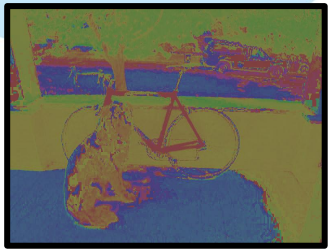


More saturation = intense colors



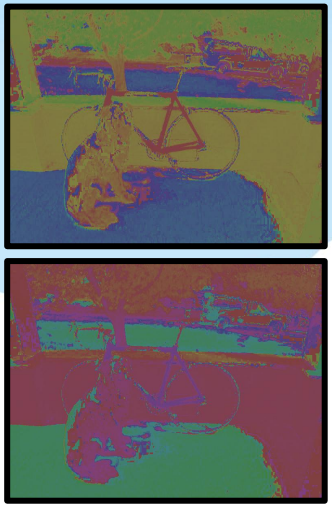
2x

More value = lighter image

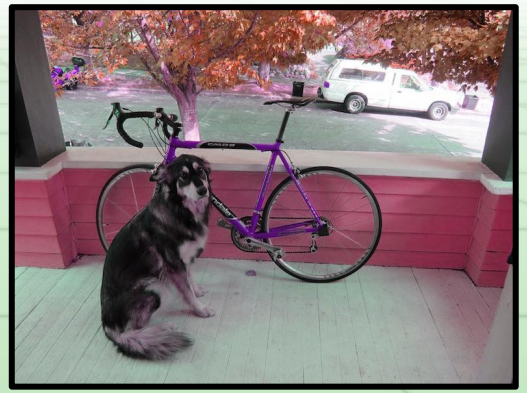


2x

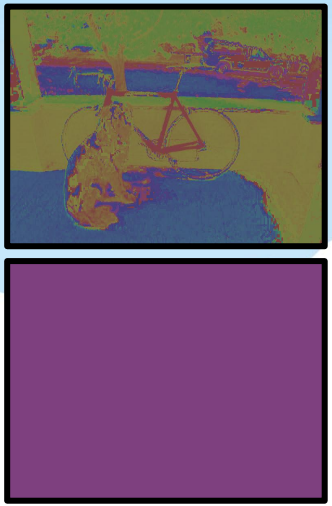
Shift hue = shift colors



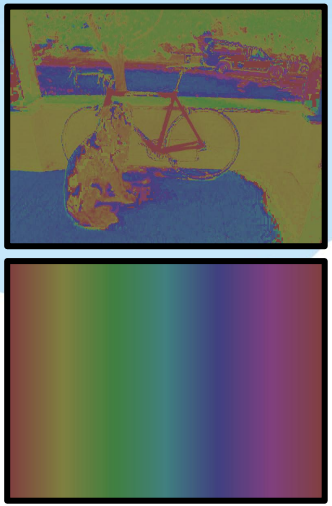
-.2



Set hue to your favorite color!



Or pattern...



Increase and threshold saturation

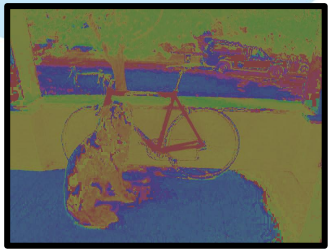


Image interpolation and resizing

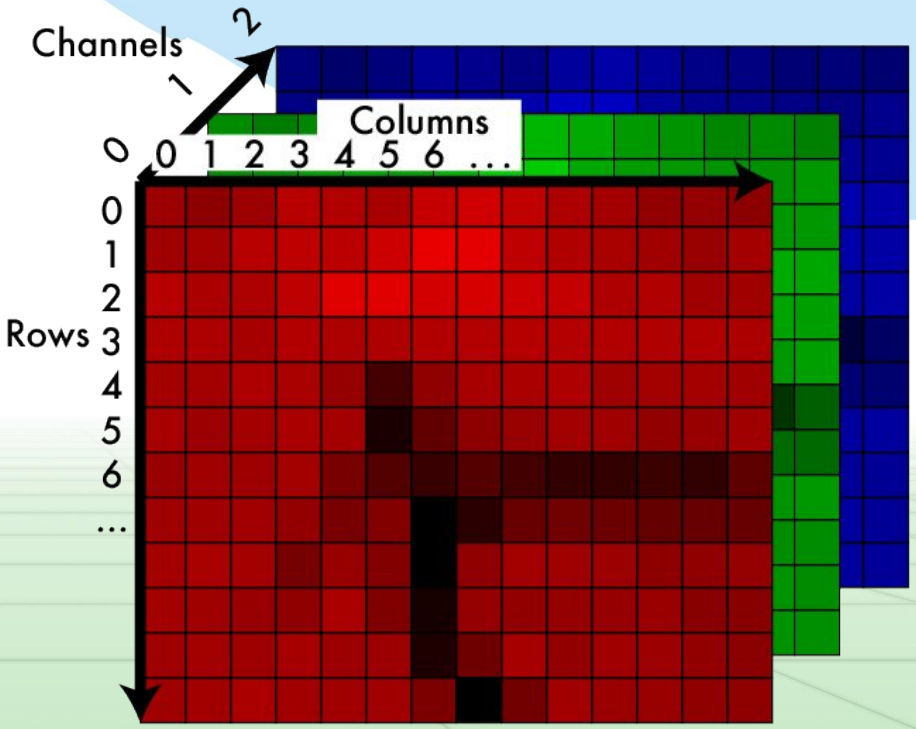
An image is kinda like a function

An image is a mapping from indices to pixel value:

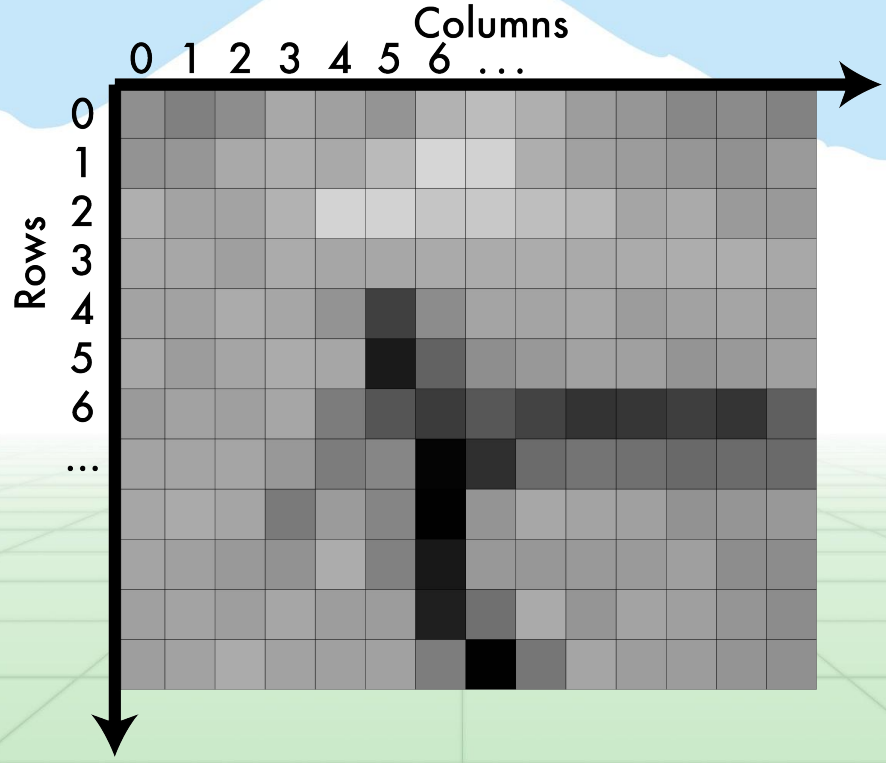
- Im: I x I x I -> R

We may want to pass in non-integers:

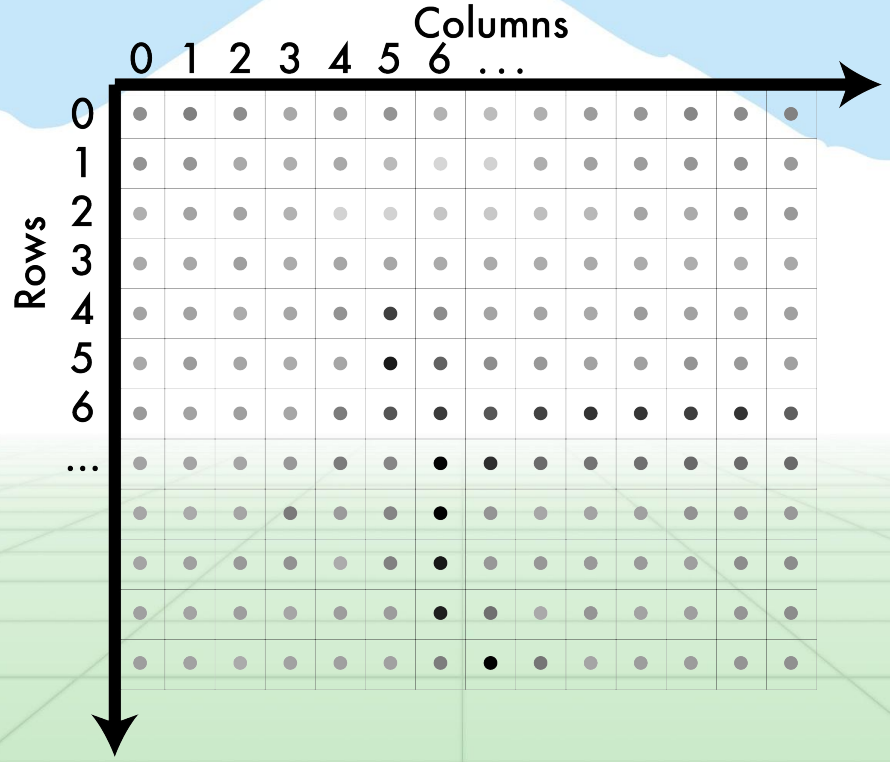
- Im’: R x R x I -> R



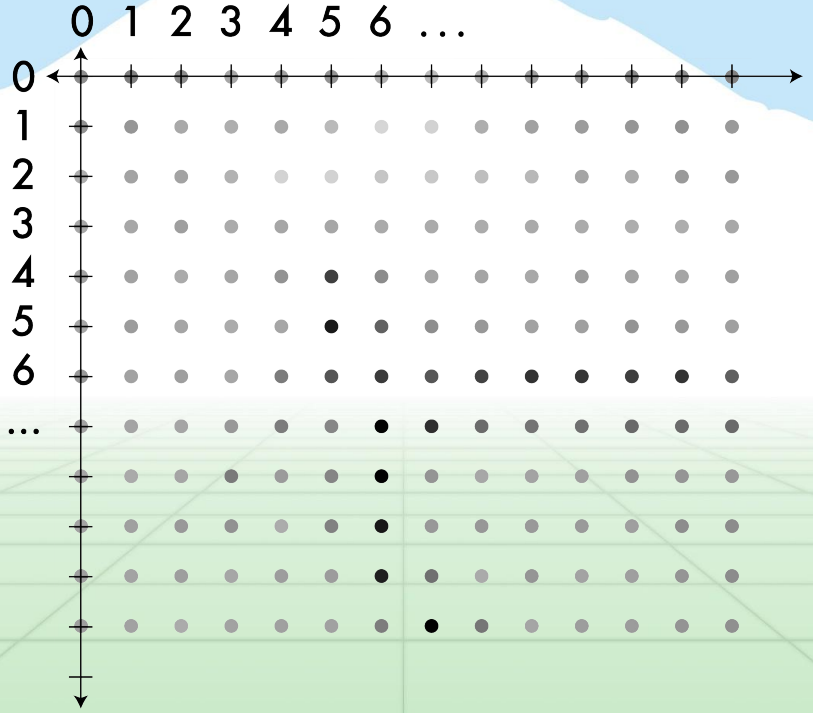
A note on coordinates in images



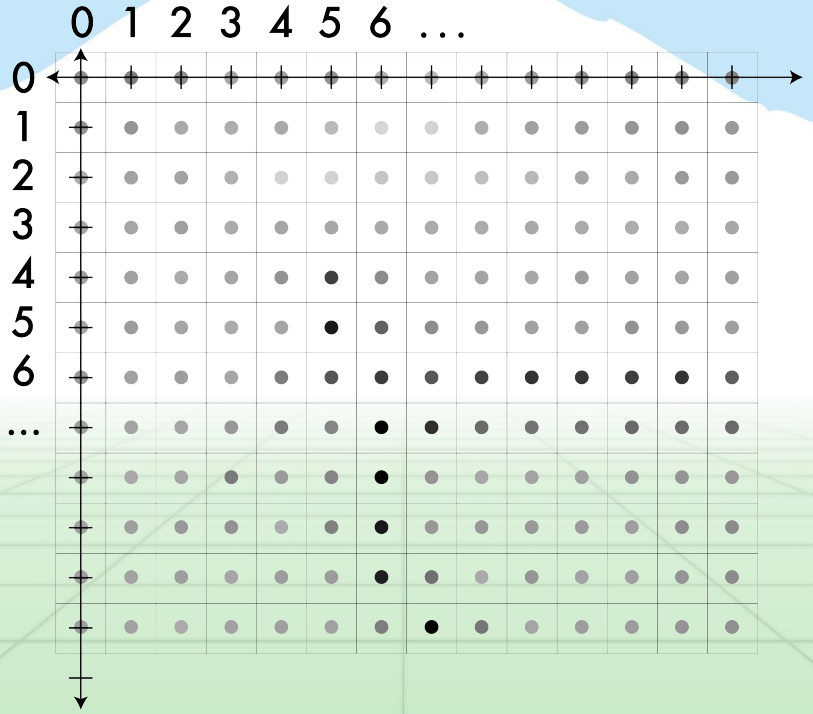
A note on coordinates in images



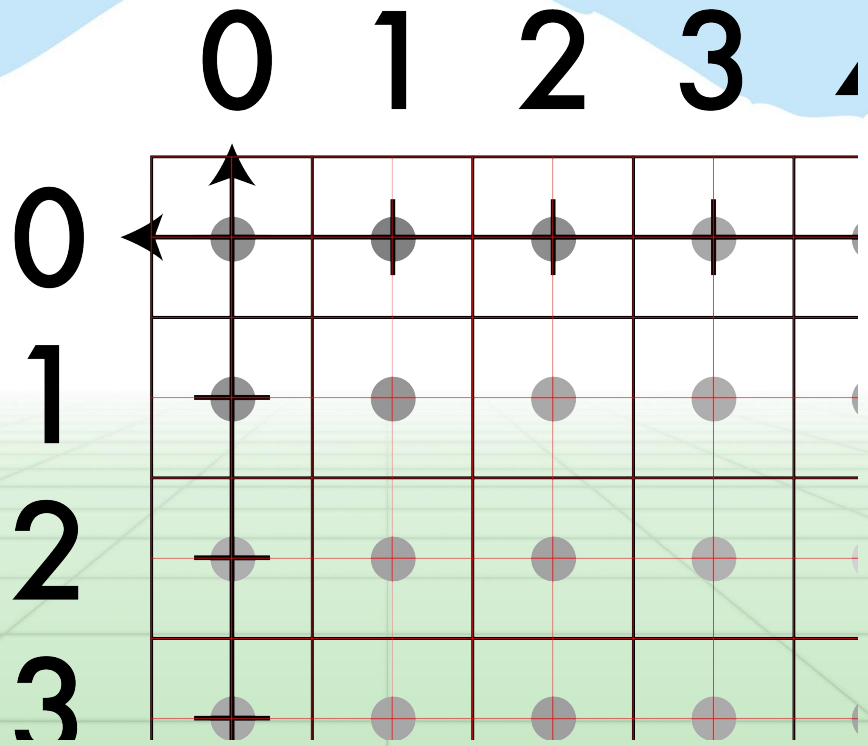
A note on coordinates in images



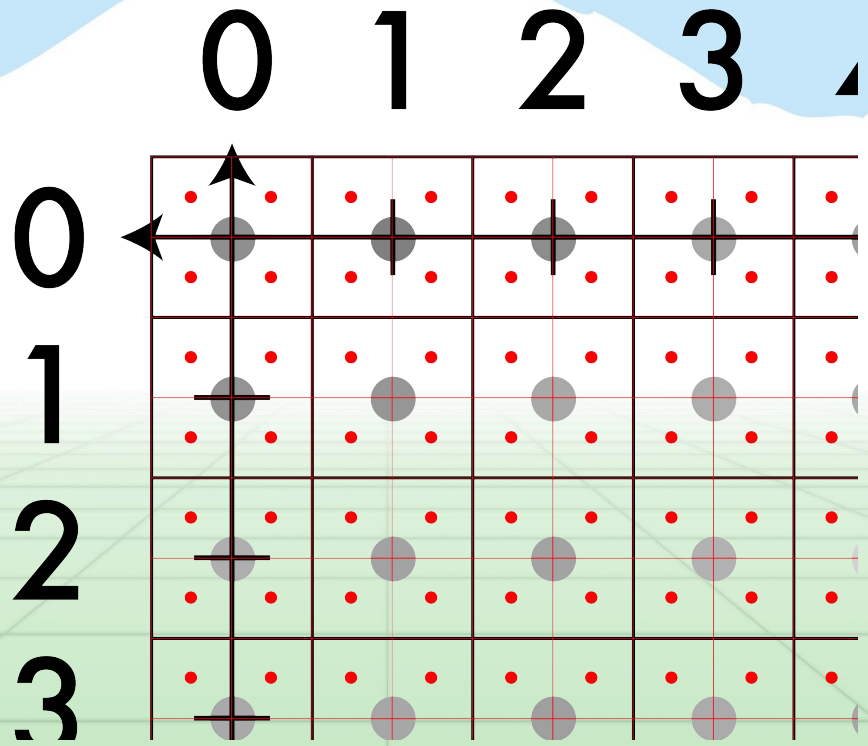
A note on coordinates in images



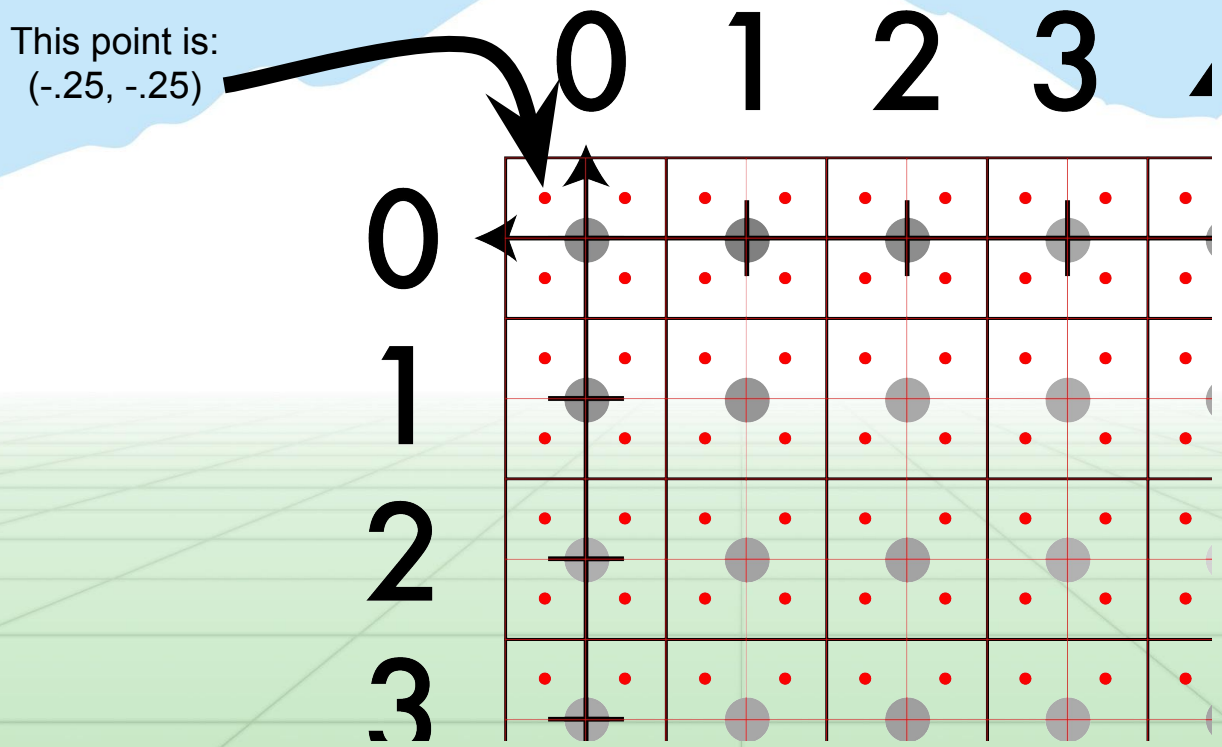
A note on coordinates in images



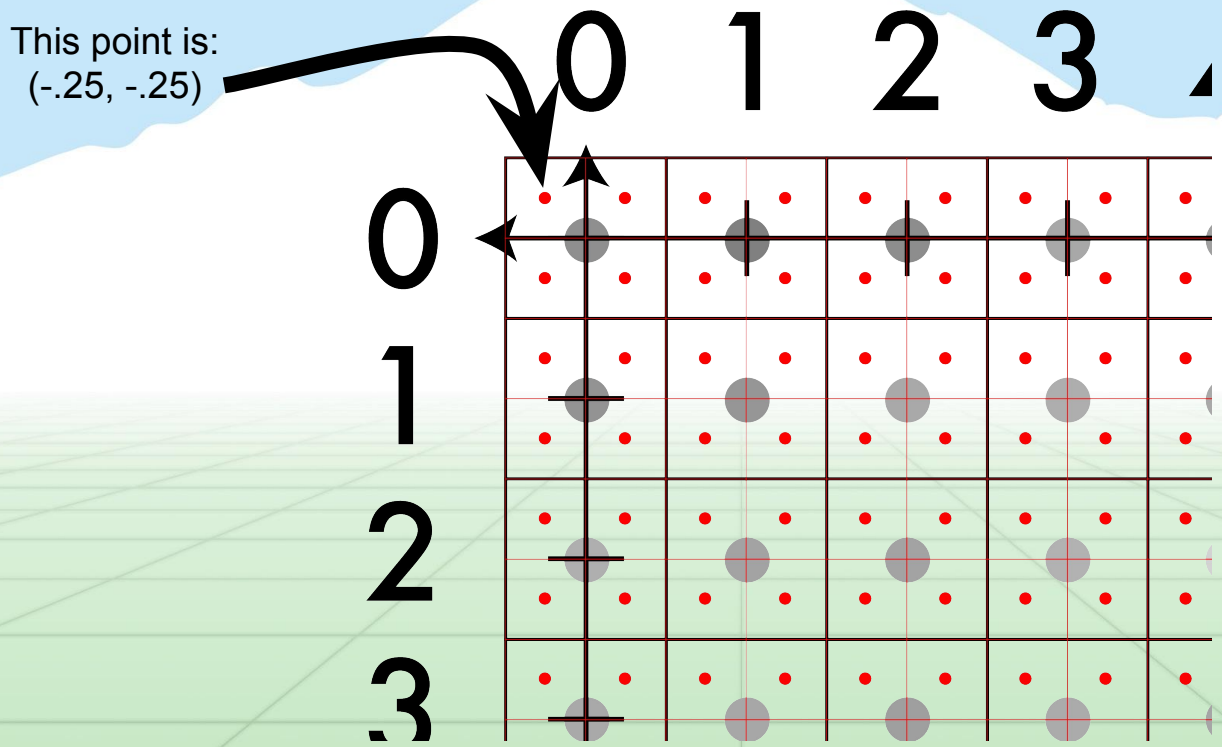
A note on coordinates in images



A note on coordinates in images



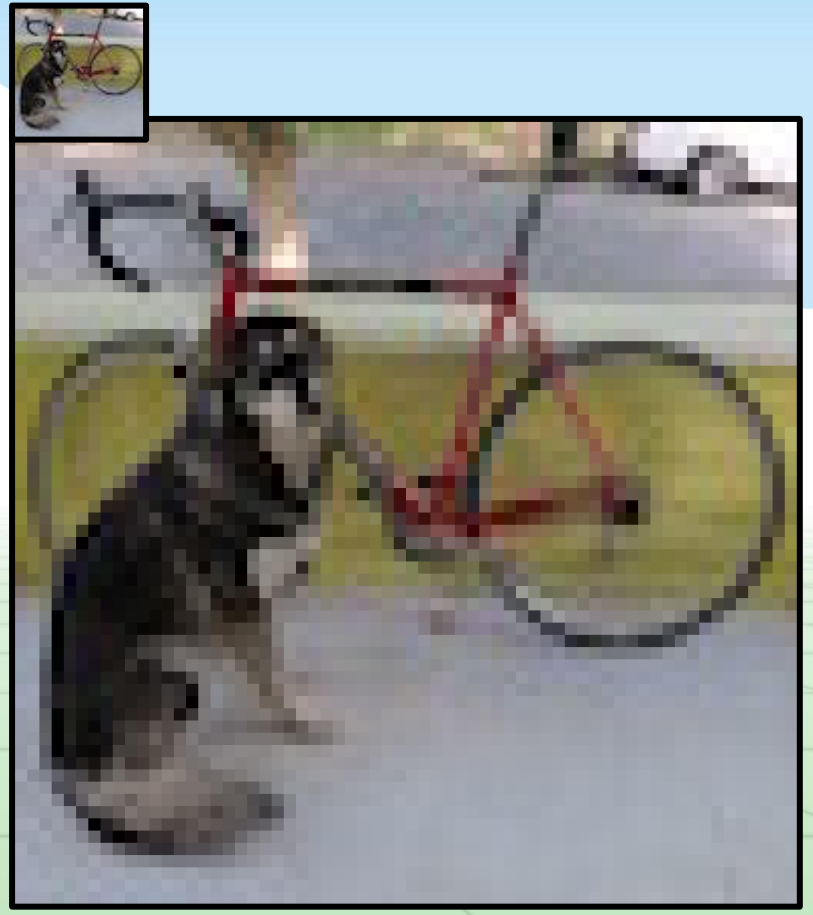
Just be careful, lots of pitfalls



Nearest neighbor: what it sounds like

```
f(x,y,z) = Im(round(x), round(y), z)
```

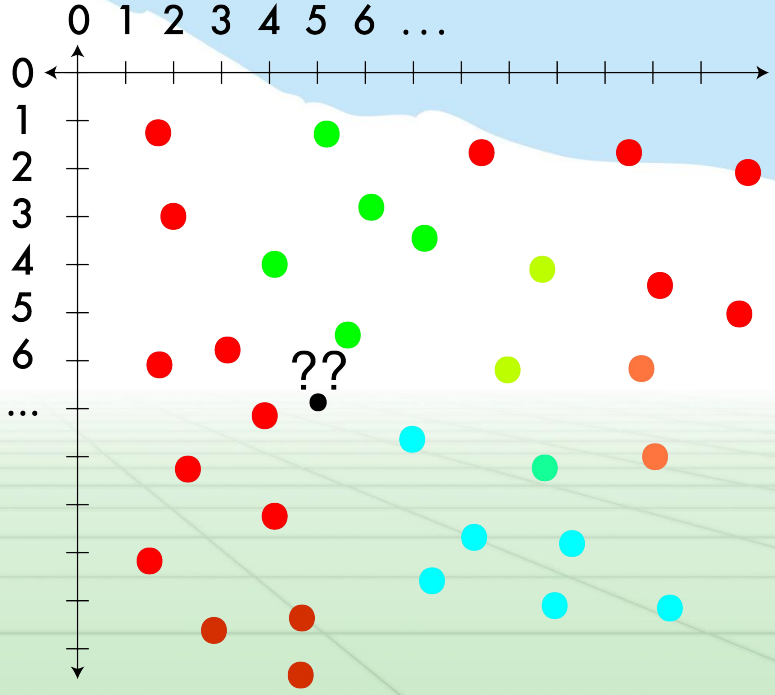
- Looks blocky
- Common pitfall:
  Integer division
  rounds down in C
- Note: z is still int



Triangle interpolation: for less structured image

Sometimes you have a regular grid, sometimes you don’t.

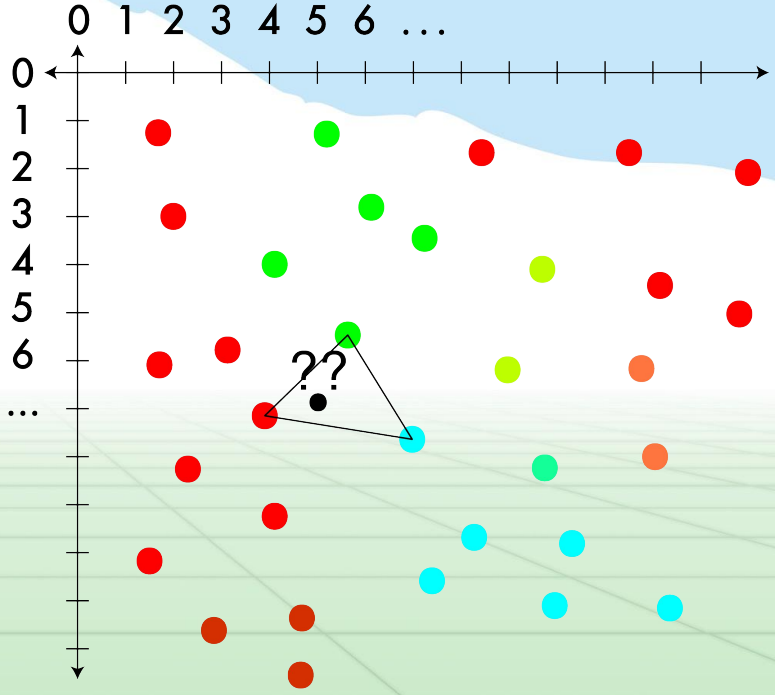
When you don’t look for triangles!



Triangle interpolation: for less structured image

Sometimes you have a regular grid, sometimes you don’t.

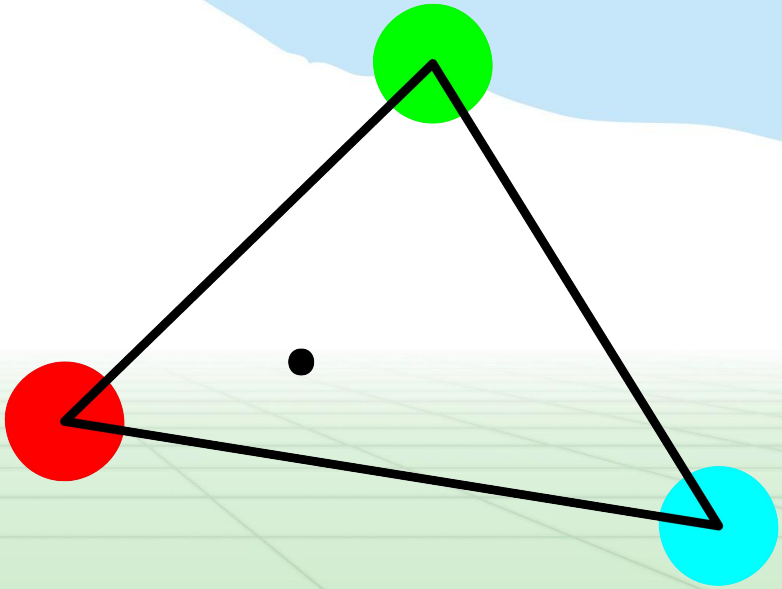
When you don’t look for triangles!



Triangle interpolation: for less structured image

Sometimes you have a regular grid, sometimes you don’t.

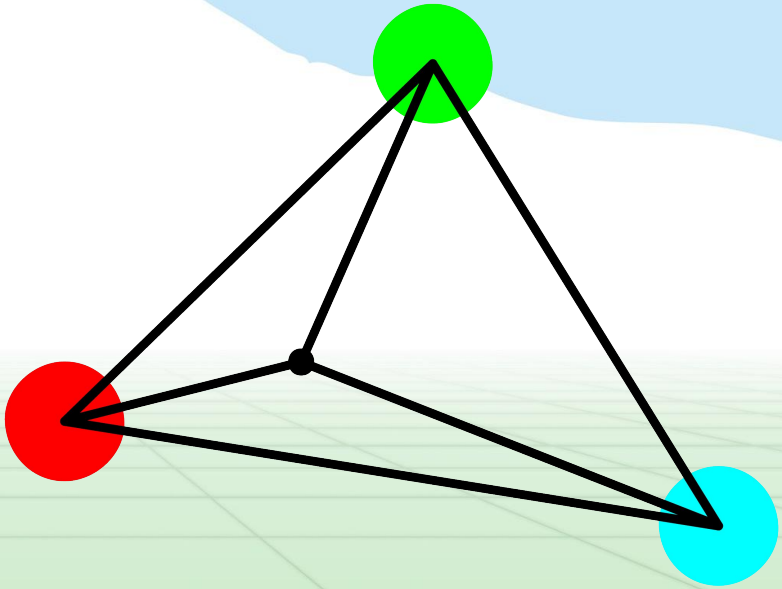
When you don’t look for triangles!



Triangle interpolation: for less structured image

Sometimes you have a regular grid, sometimes you don’t.

When you don’t look for triangles!

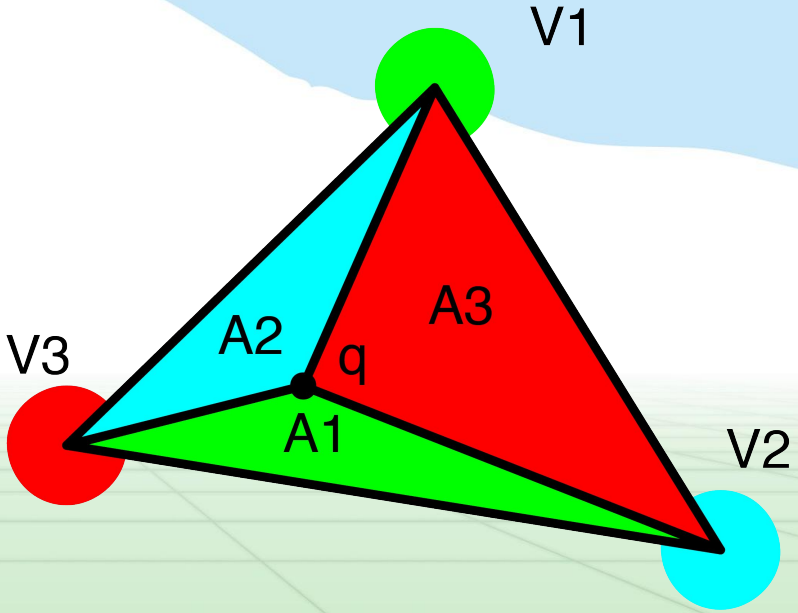


Triangle interpolation: for less structured image

Weighted sum using of triangles:

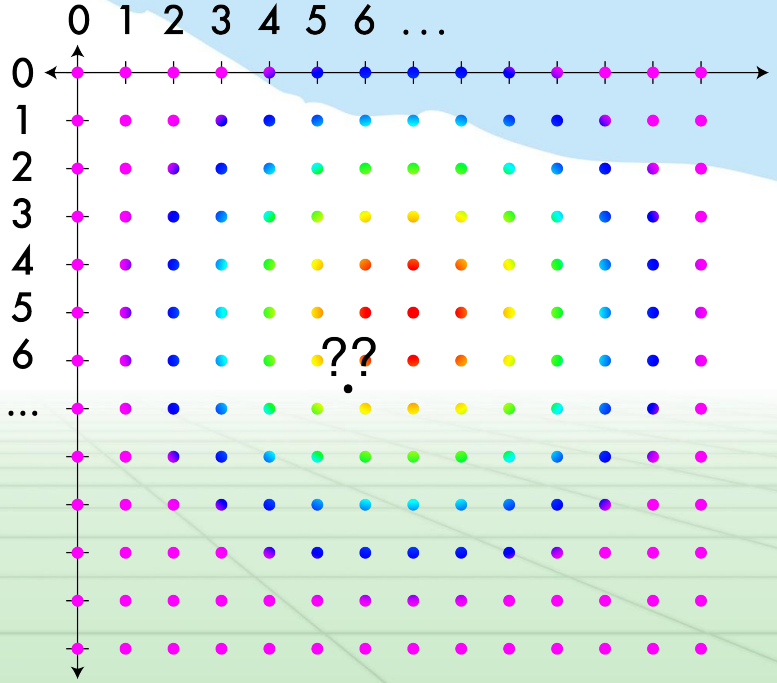
Q = V1*A1 + V2*A2 + V3*A3

Should normalize this based on total area



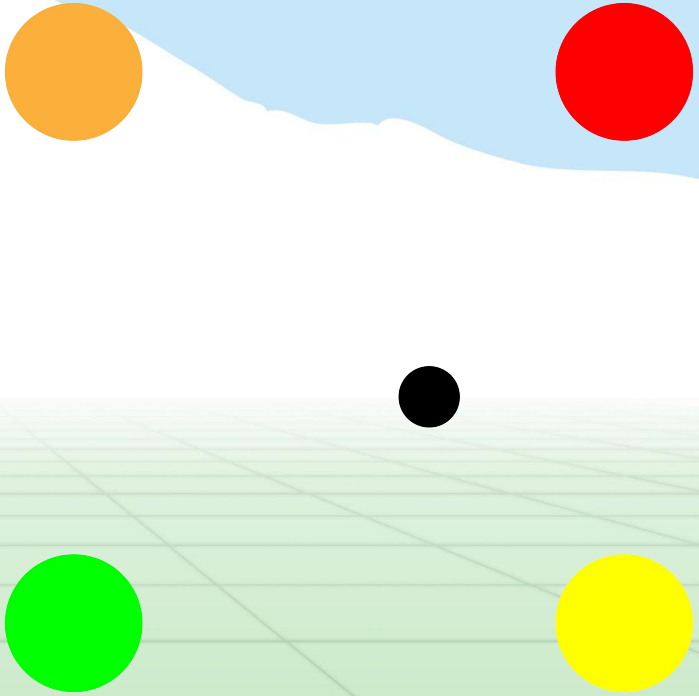
Bilinear interpolation: for grids, pretty good

This time find the closest pixels in a box



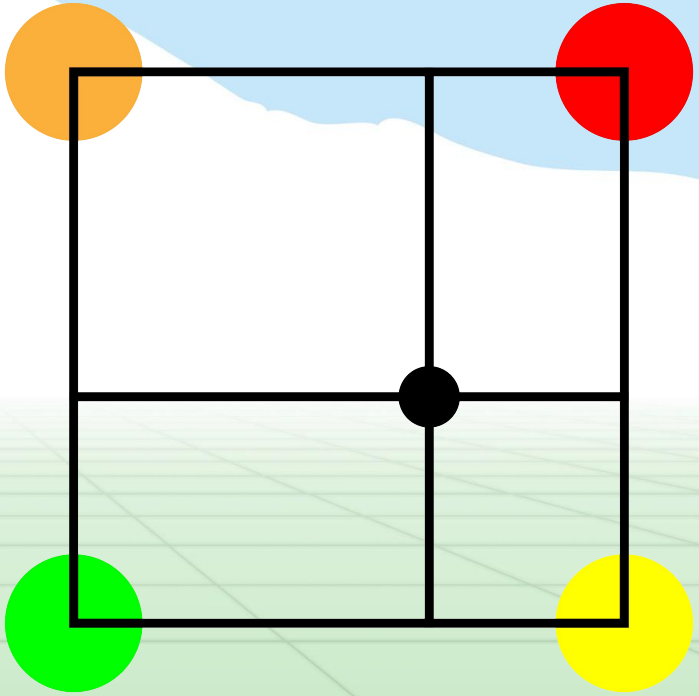
Bilinear interpolation: for grids, pretty good

This time find the
closest pixels in a box



Bilinear interpolation: for grids, pretty good

This time find the closest pixels in a box



Bilinear interpolation: for grids, pretty good

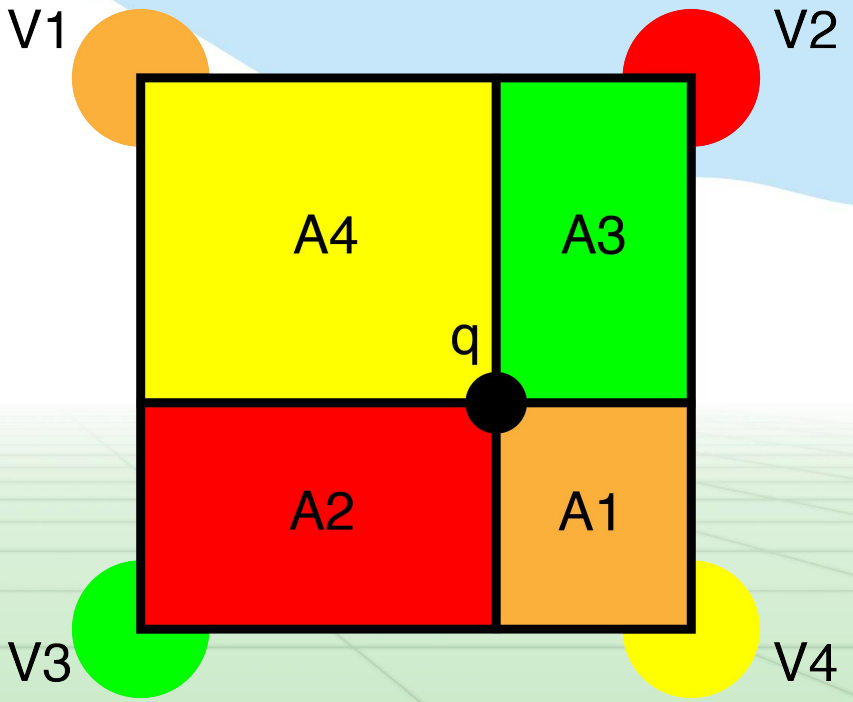
This time find the closest pixels in a box

Same plan, weighted sum based on area of opposite rectangle

$$Q = V1*A1 + V2*A2 + V3*A3 + V4*A4$$

Still need to normalize!

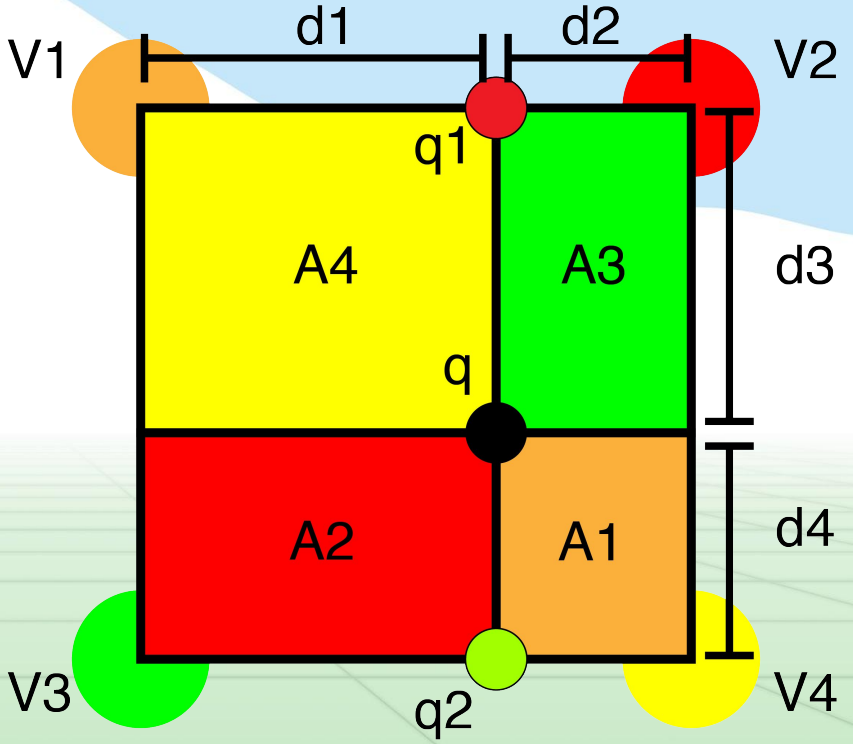
Or do we?



Bilinear interpolation: for grids, pretty good

Alternatively, linear interpolation of linear interpolates

```
q1 = V1*d2 + V2*d1
q2 = V3*d2 + V4*d1
q = q1*d4 + q2*d3
```

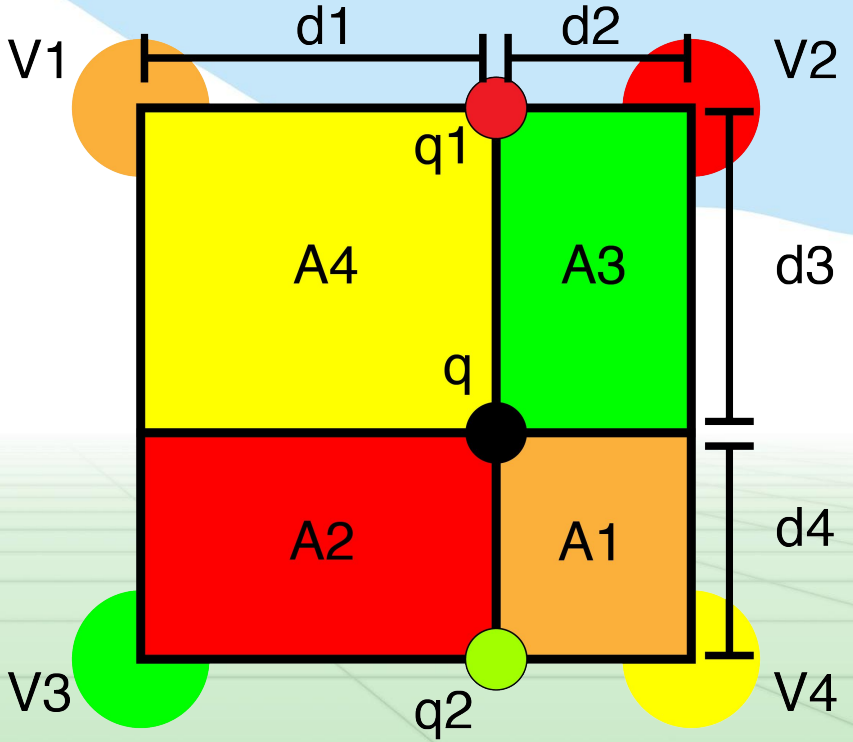


Bilinear interpolation: for grids, pretty good

```
q1 = V1*d2 + V2*d1
q2 = V3*d2 + V4*d1
q = q1*d4 + q2*d3
```

Equivalent:

```
q = q1*d4 + q2*d3
```

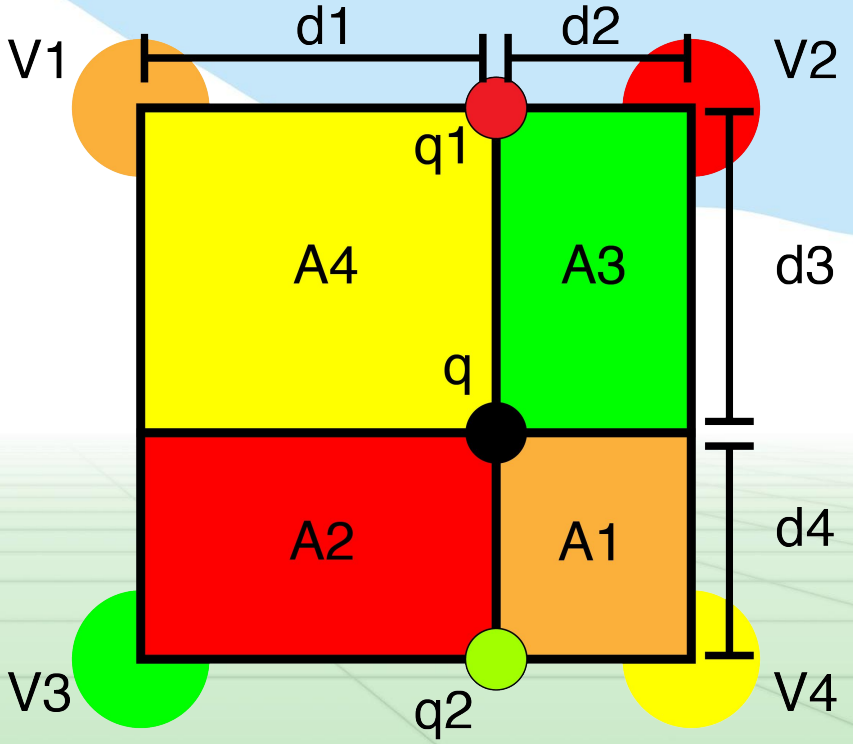


Bilinear interpolation: for grids, pretty good

```
q1 = V1*d2 + V2*d1
q2 = V3*d2 + V4*d1
q = q1*d4 + q2*d3
```

Equivalent:

```
q = q1*d4 + q2*d3
q = (V1*d2 + V2*d1)*d4 + (V3*d2 + V4*d1)*d3 (subst)
```

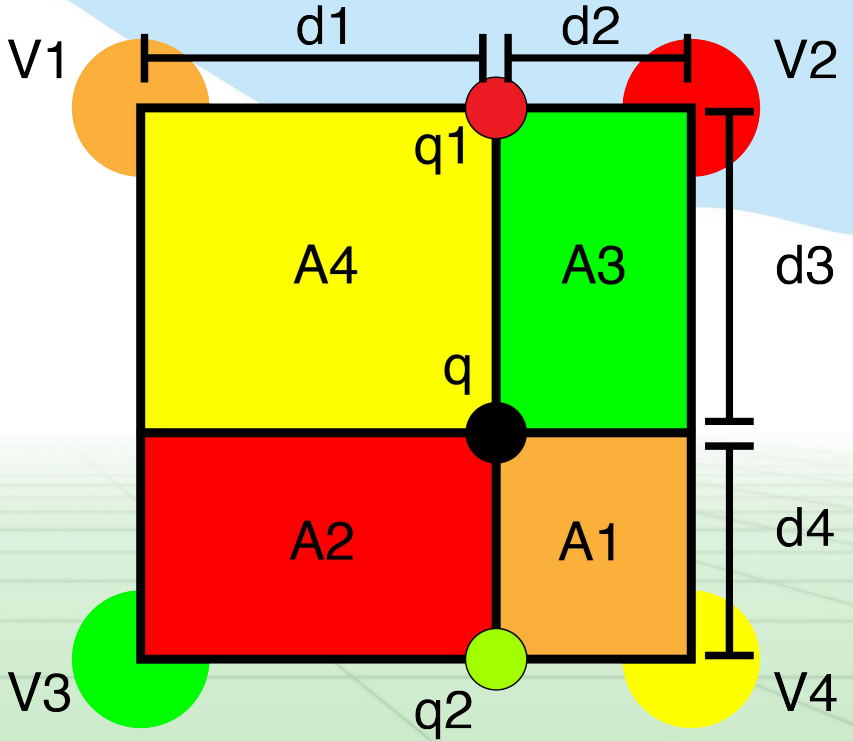


Bilinear interpolation: for grids, pretty good

```
q1 = V1*d2 + V2*d1
q2 = V3*d2 + V4*d1
q = q1*d4 + q2*d3
```

Equivalent:

```
q = q1*d4 + q2*d3
q = (V1*d2 + V2*d1)*d4 + (V3*d2 + V4*d1)*d3 (subst)
q = V1*d2*d4 + V2*d1*d4 + V3*d2*d3 + V4*d1*d3 (distribution)
```



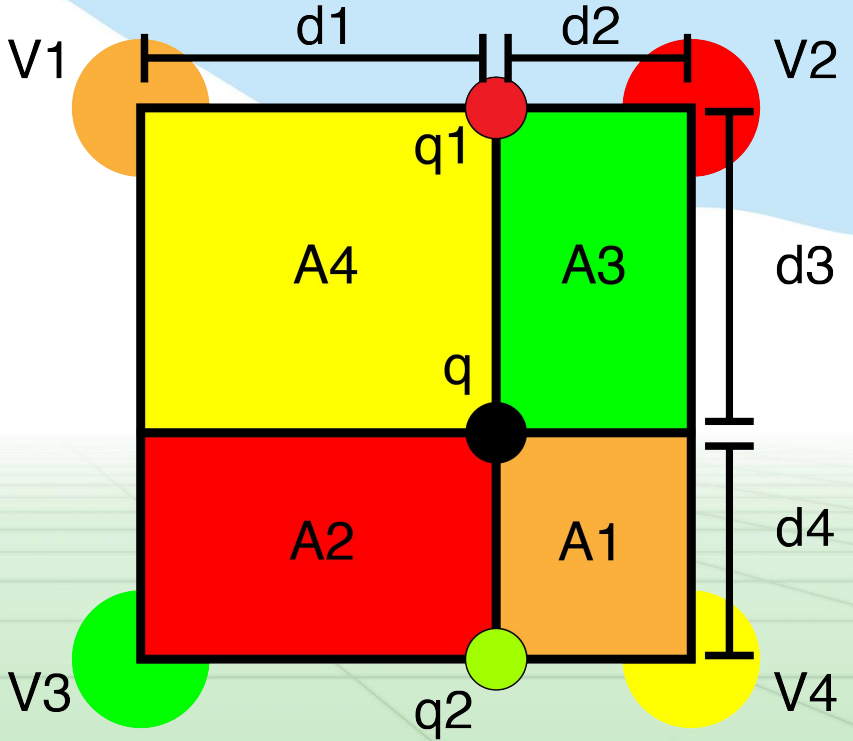
Bilinear interpolation: for grids, pretty good

```
q1 = V1*d2 + V2*d1
q2 = V3*d2 + V4*d1
q = q1*d4 + q2*d3
```

Equivalent:

```
q = q1*d4 + q2*d3
q = (V1*d2 + V2*d1)*d4 + (V3*d2 + V4*d1)*d3 (subst)
q = V1*d2*d4 + V2*d1*d4 + V3*d2*d3 + V4*d1*d3 (distribution)

Recall:
A1 = d2*d4
A2 = d1*d4
A3 = d2*d3
A4 = d1*d3
```



Bilinear interpolation: for grids, pretty good

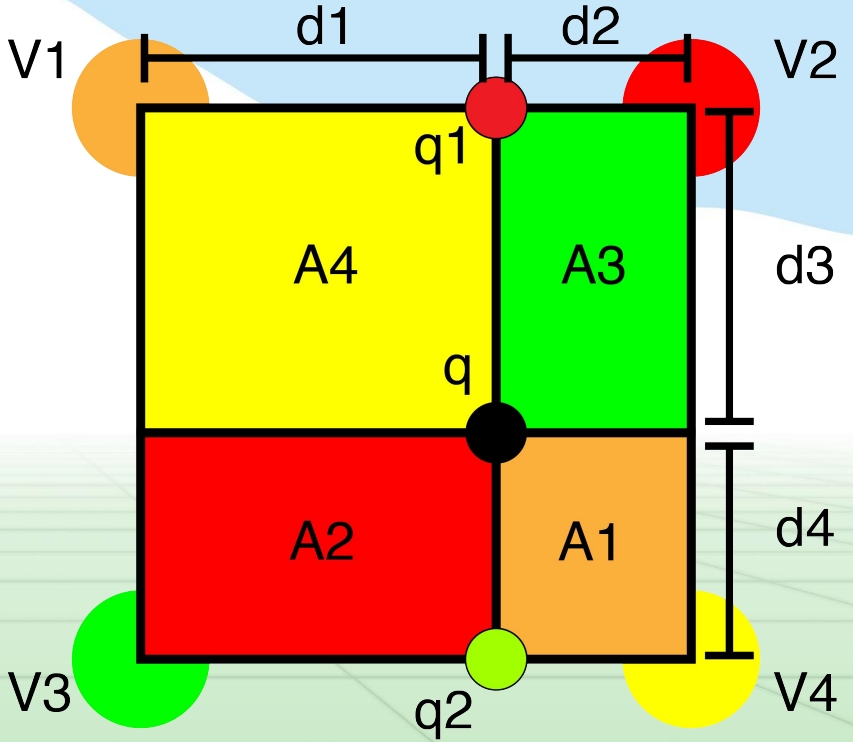
```
q1 = V1*d2 + V2*d1
q2 = V3*d2 + V4*d1
q = q1*d4 + q2*d3
```

Equivalent:

```
q = q1*d4 + q2*d3
q = (V1*d2 + V2*d1)*d4 + (V3*d2 + V4*d1)*d3 (subst)
q = V1*d2*d4 + V2*d1*d4 + V3*d2*d3 + V4*d1*d3 (distribution)

Recall:
A1 = d2*d4
A2 = d1*d4
A3 = d2*d3
A4 = d1*d3
```

```
q = V1*A1 + V2*A2 + V3*A3 + V4*A4
```



Bilinear interpolation: for grids, pretty good

```
q1 = V1*d2 + V2*d1
q2 = V3*d2 + V4*d1
q = q1*d4 + q2*d3
```

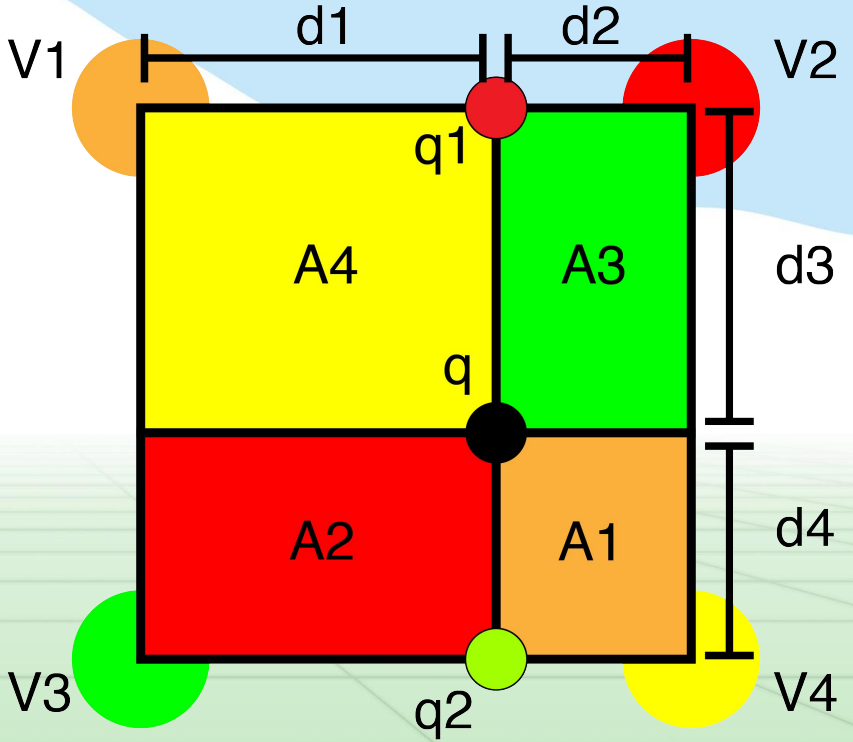
Equivalent:

```
q = q1*d4 + q2*d3
q = (V1*d2 + V2*d1)*d4 + (V3*d2 + V4*d1)*d3 (subst)
q = V1*d2*d4 + V2*d1*d4 + V3*d2*d3 + V4*d1*d3 (distribution)

Recall:
A1 = d2*d4
A2 = d1*d4
A3 = d2*d3
A4 = d1*d3
```

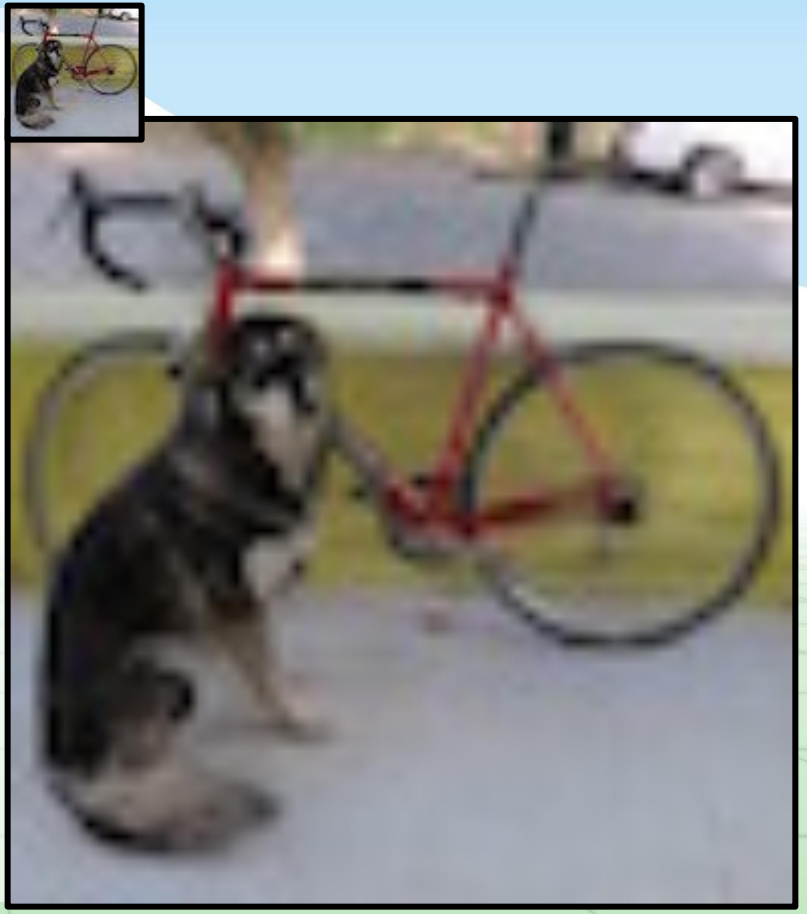
```
q = V1*A1 + V2*A2 + V3*A3 + V4*A4
```

yay!



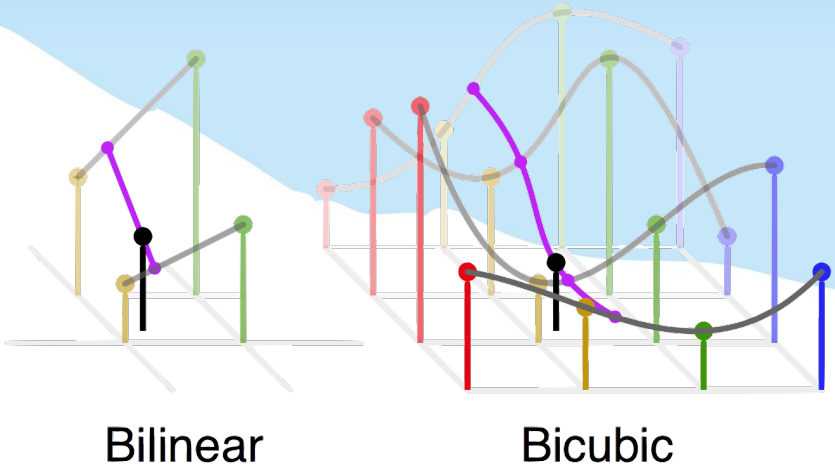
Bilinear interpolation: for grids, pretty good

- Smoother than NN
- More complex
  - 4 lookups
  - Some math
- Often the right tradeoff of speed vs final result



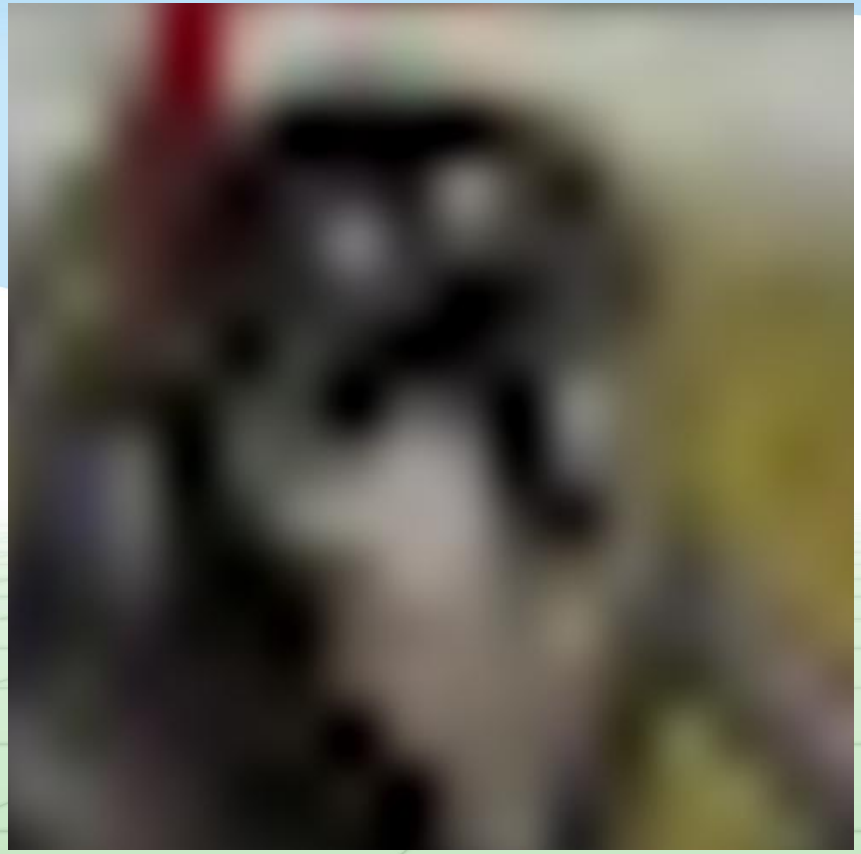
Bicubic sampling: more complex, maybe better?

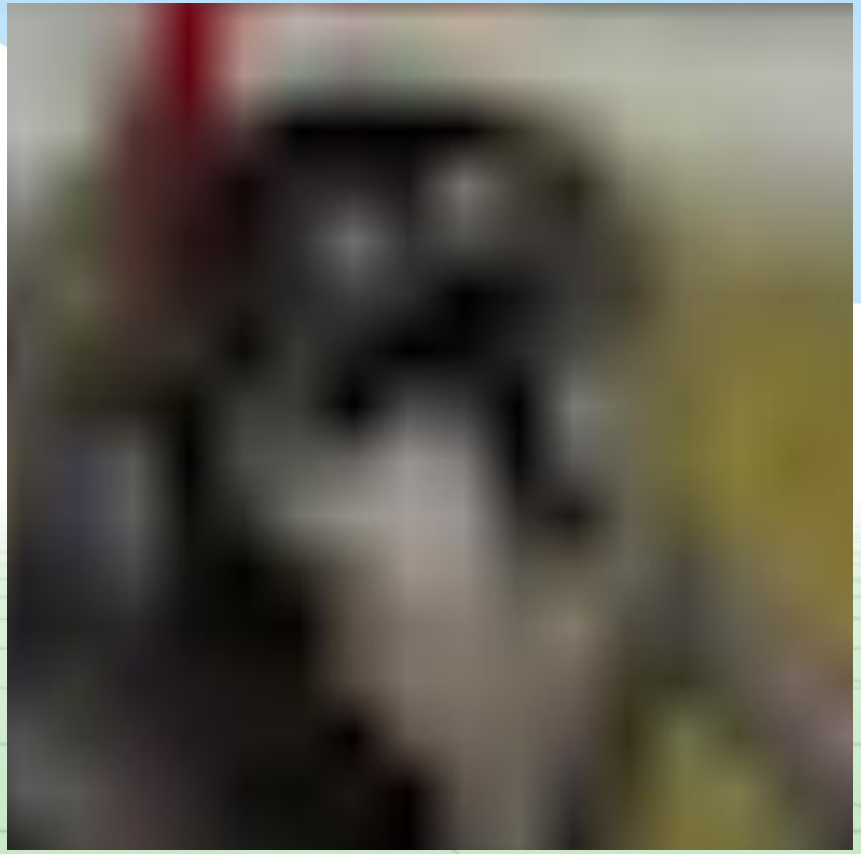
- A cubic interpolation of 4 cubic interpolations
- Smoother than bilinear, no “star”
- 16 nearest neighbors
- Fit 3rd order poly:
  - f(x) = a + bx + cx^2 + dx^3
- Interpolate along axis
- Fit another poly to interpolated values



Bicubic vs bilinear

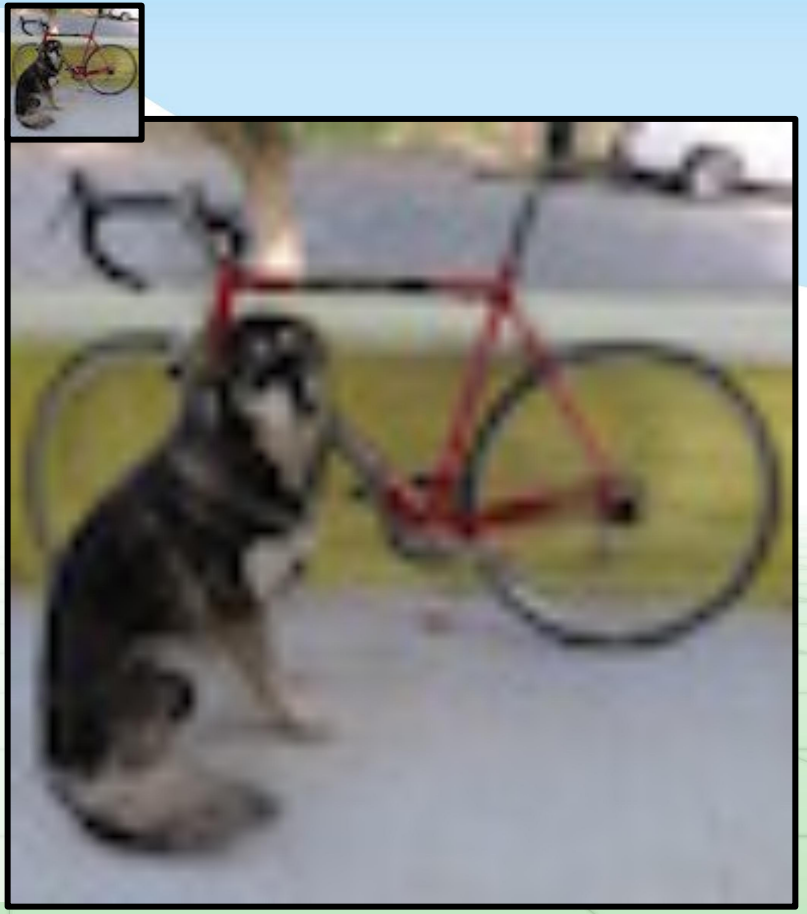
Bicubic vs bilinear





Resize algorithm:

- For each pixel in new image:
  - Map to old im coordinates
  - Interpolate value
  - Set new value in image



What about shrinking?

- NN and Bilinear only look at small area
- Lots of artifacting
- Staircase pattern on diagonal lines
- We'll fix this next class with filters!