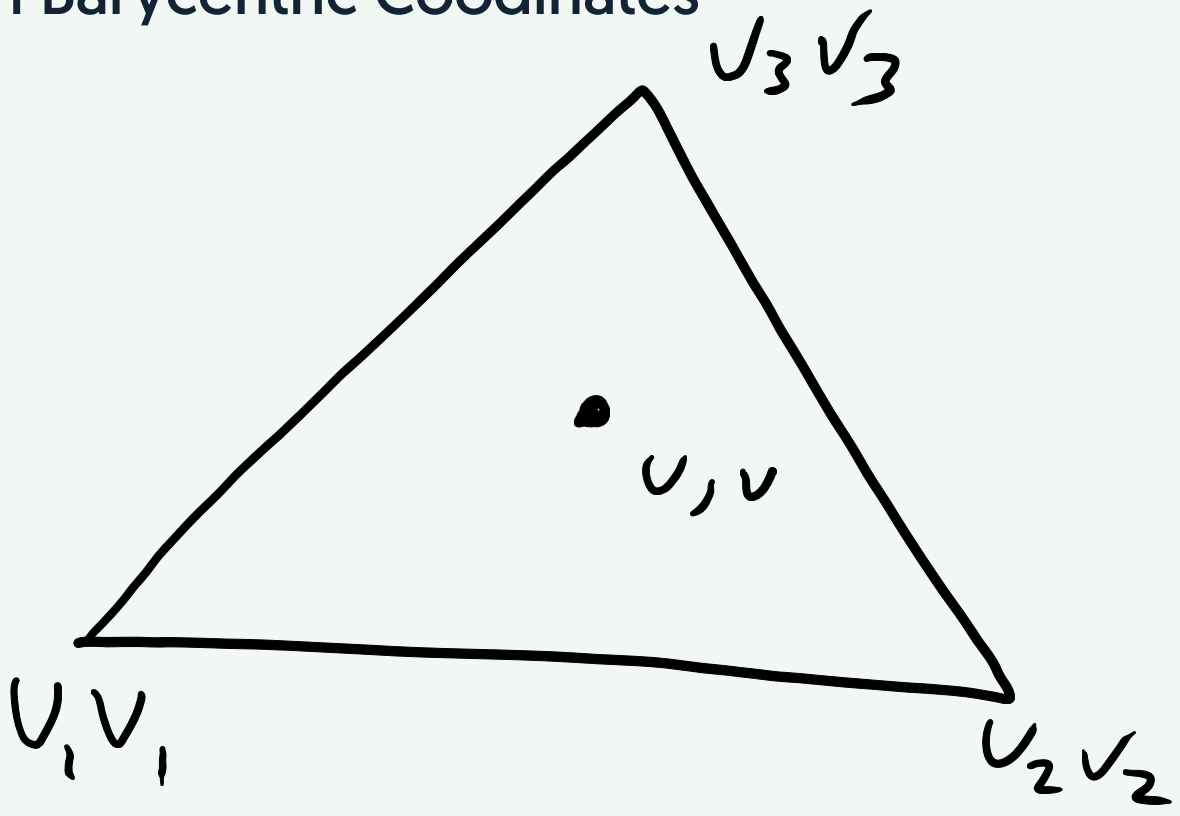
Lecture 18 Part D:
How Texture Mapping Works

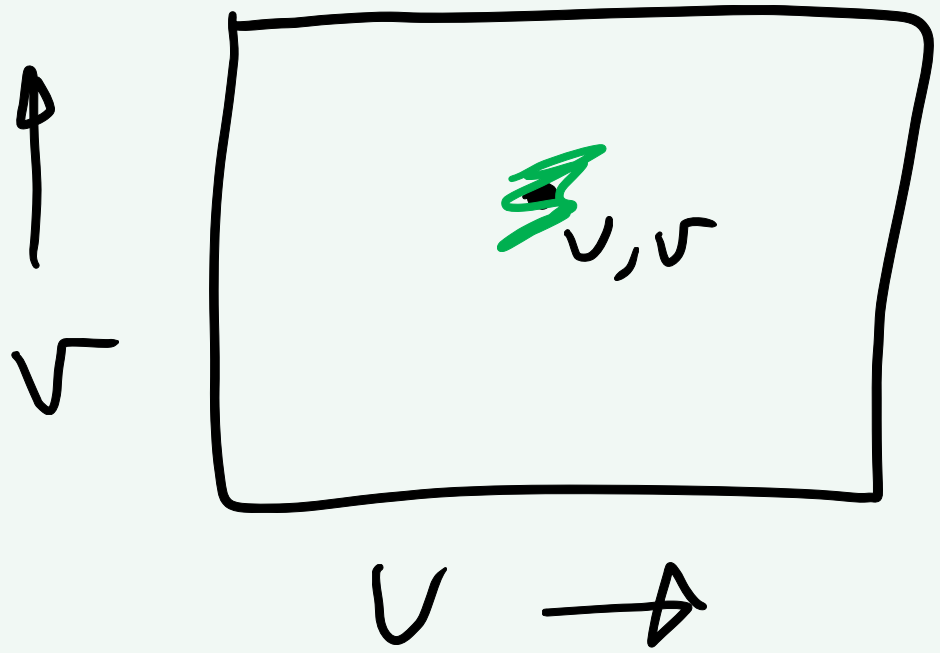
The Texture Coordinate

UV Mapping with Barycentric Coodinates



The Texture Lookup

Given UV what is the color?

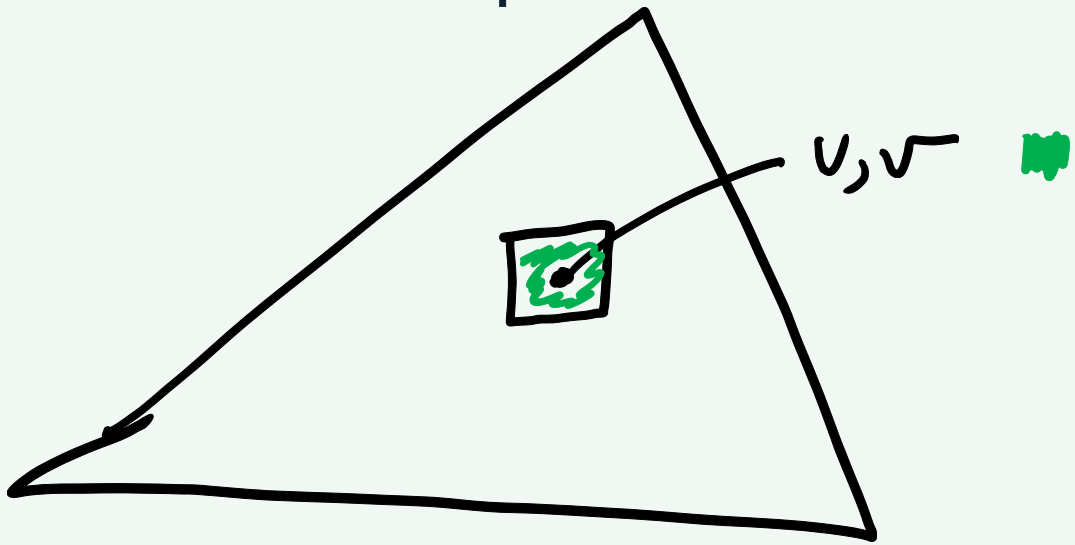


Where does the color go?

Pixel is a little square?

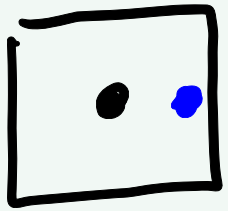
(no, but useful for thinking about it)

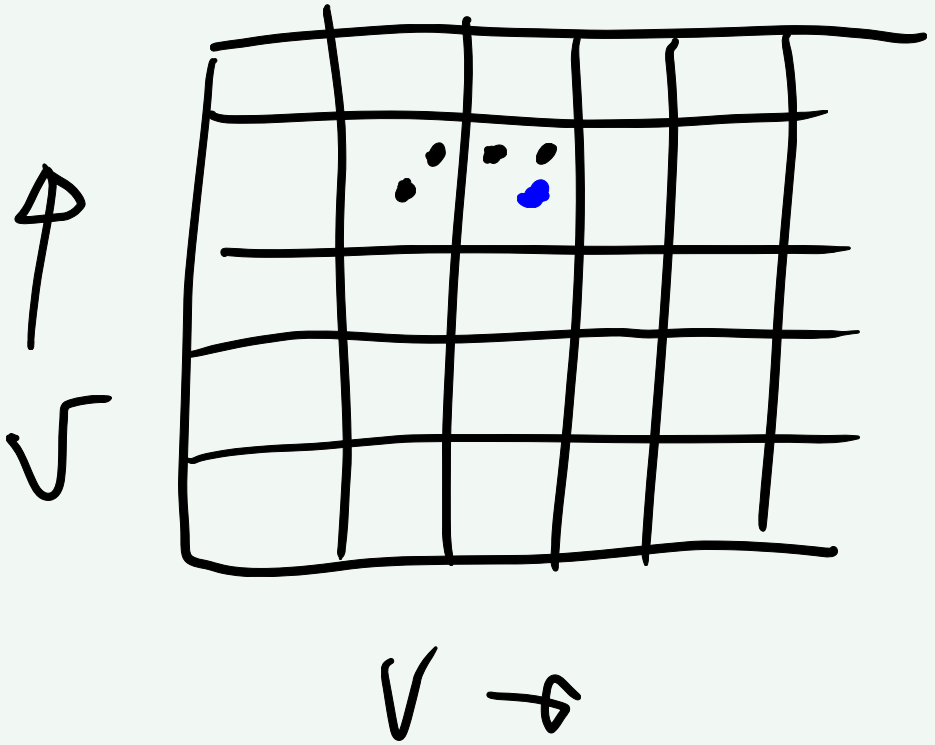
Assume UV is the center of the pixel



The problems

1. U,V might not be integers
2. Target Area of the pixel
3. Target Areas next to each other





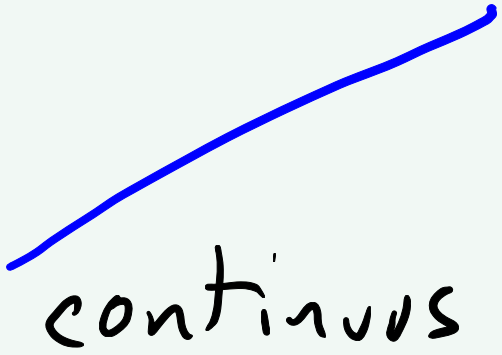
The general problem...

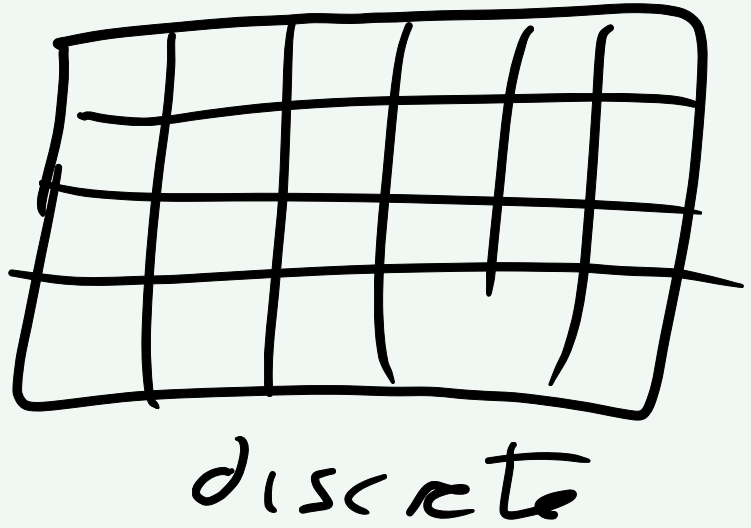
The world/image in continuous

The pixel grid is discrete

This is a fundamental problem in graphics

We'll come back to it

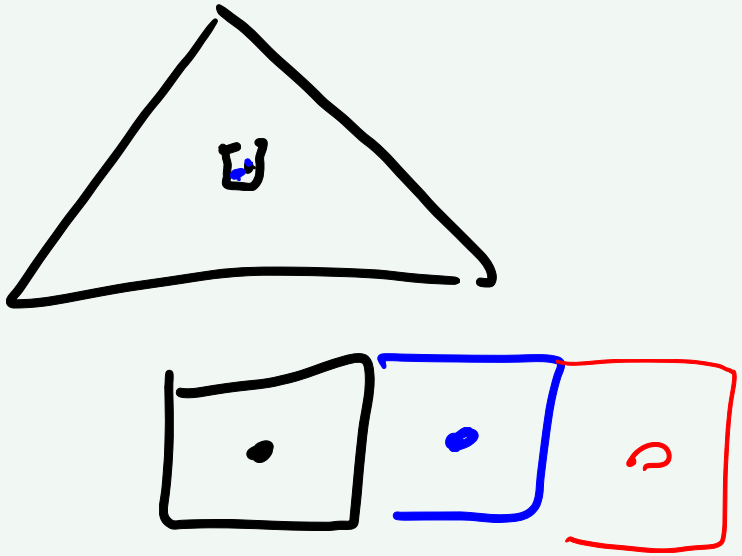


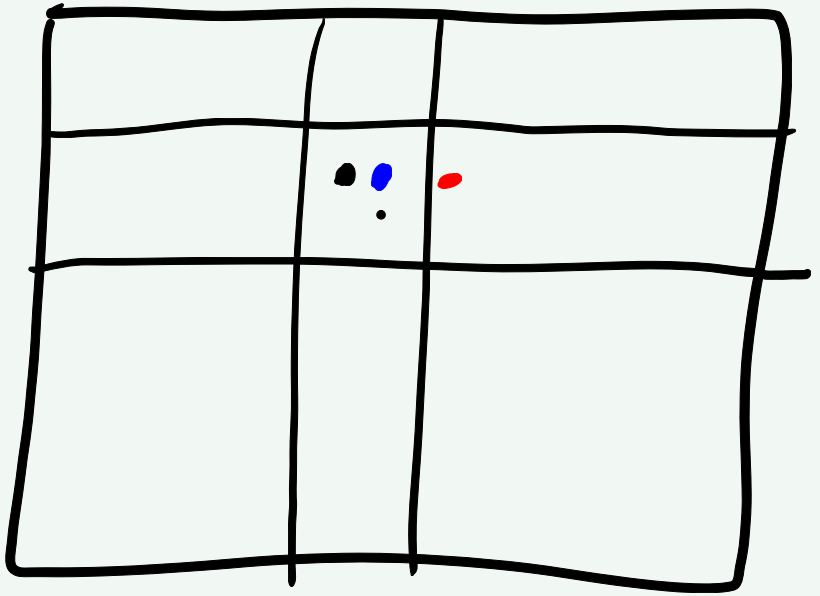


Texture Lookups: Magnification

A pixel covers less than a pixel in the image

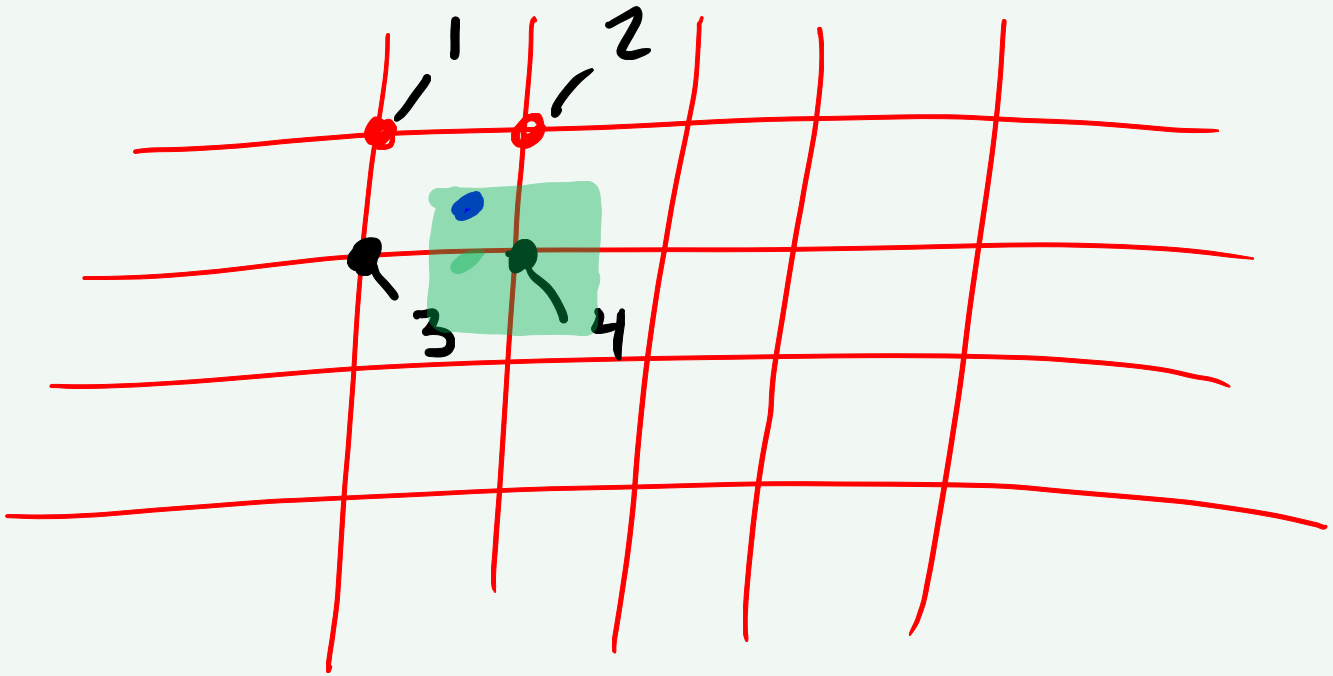
Note: this same reasoning works for "point sampling" (the center of an area)





Basic Solution: Nearest-Neighbor

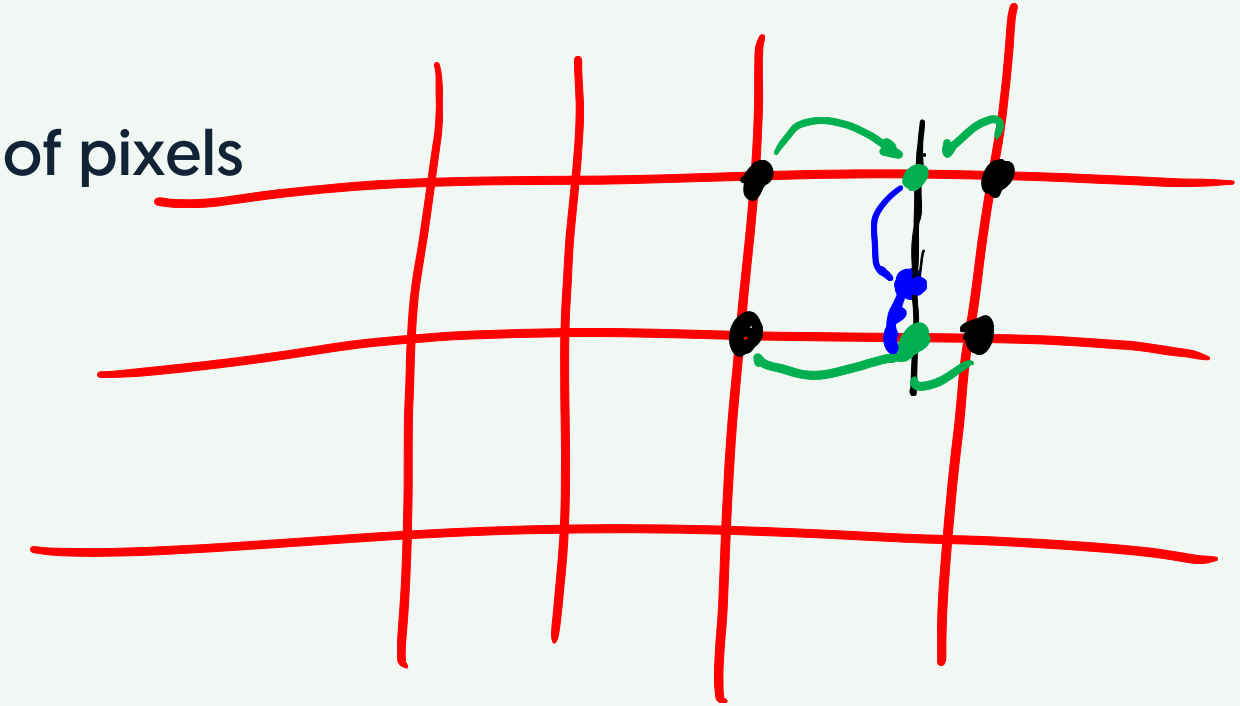
Think of colors at grid corners (points) not filling squares



Standard Solution: Bi-Linear Interpolation

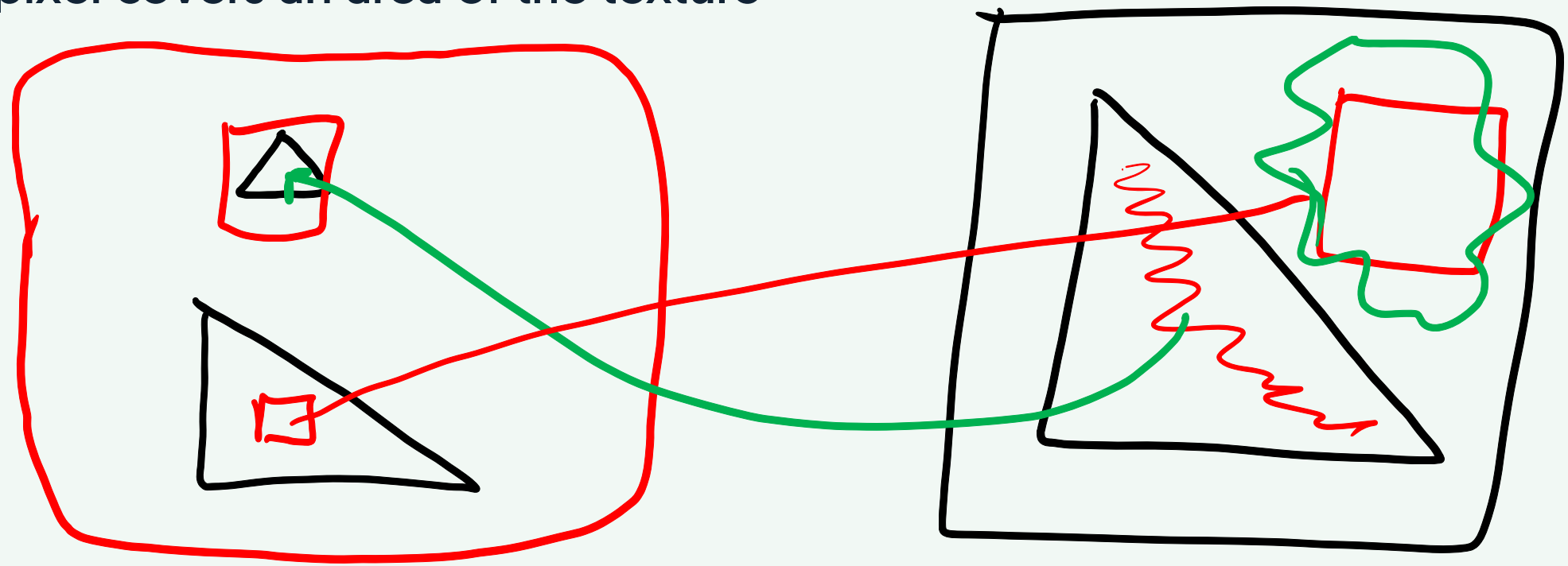
Treat the pixel as a point

Does not consider edges of pixels

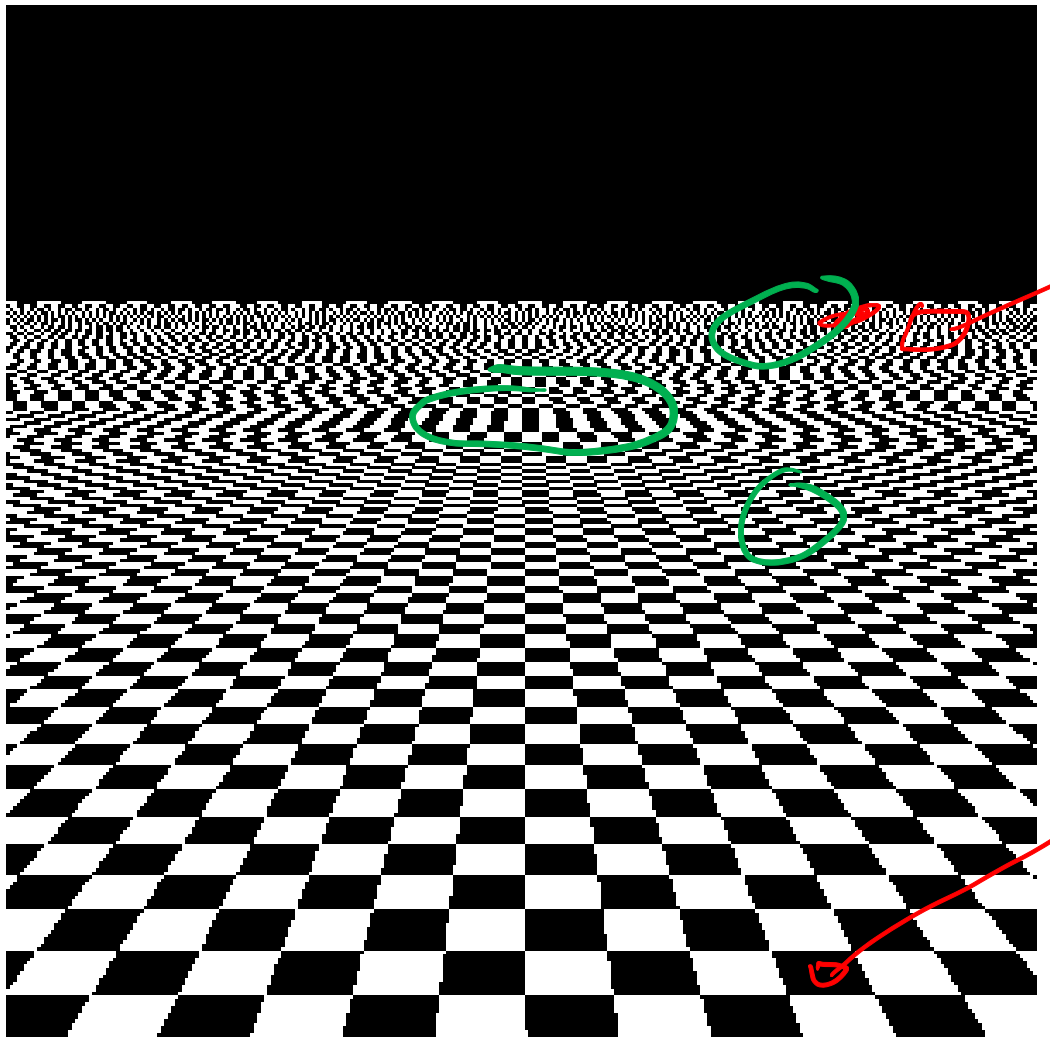


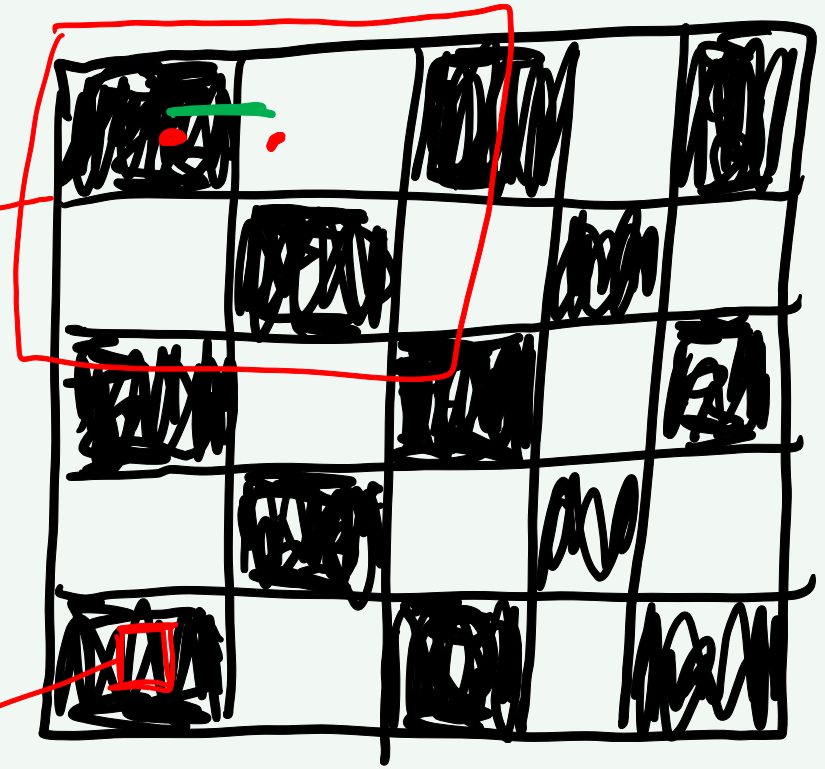
Texture Lookups: Minification

One pixel covers an area of the texture



What if we just look up the color?

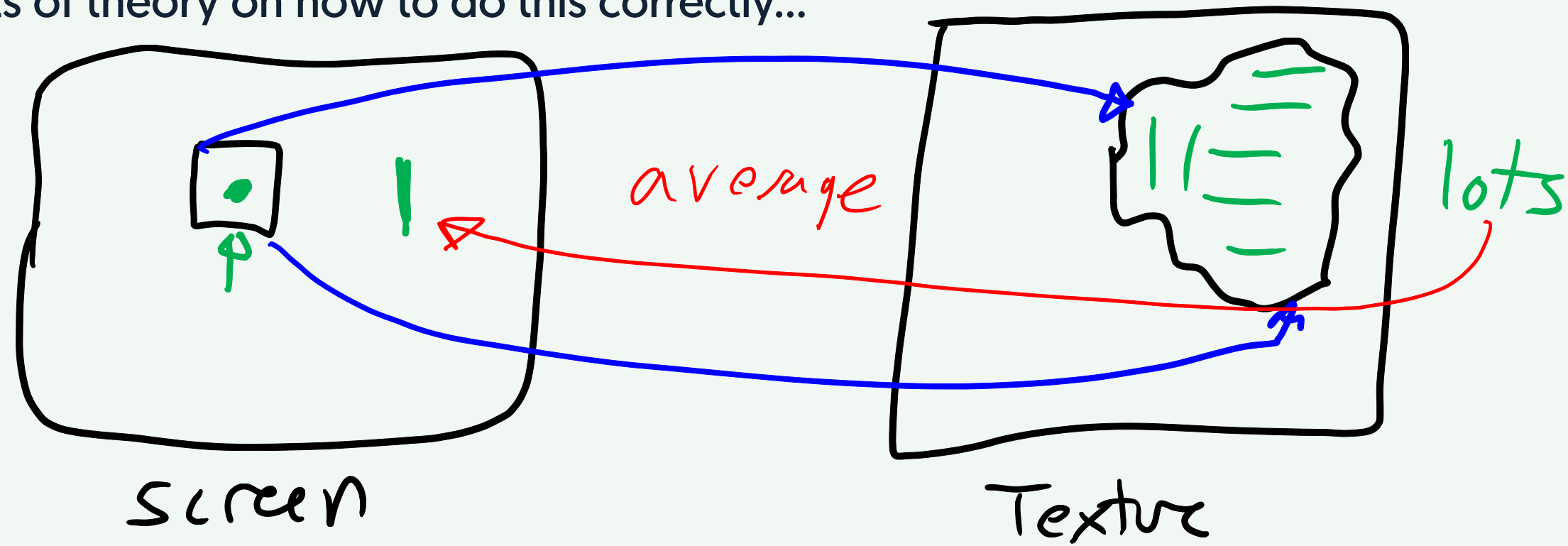




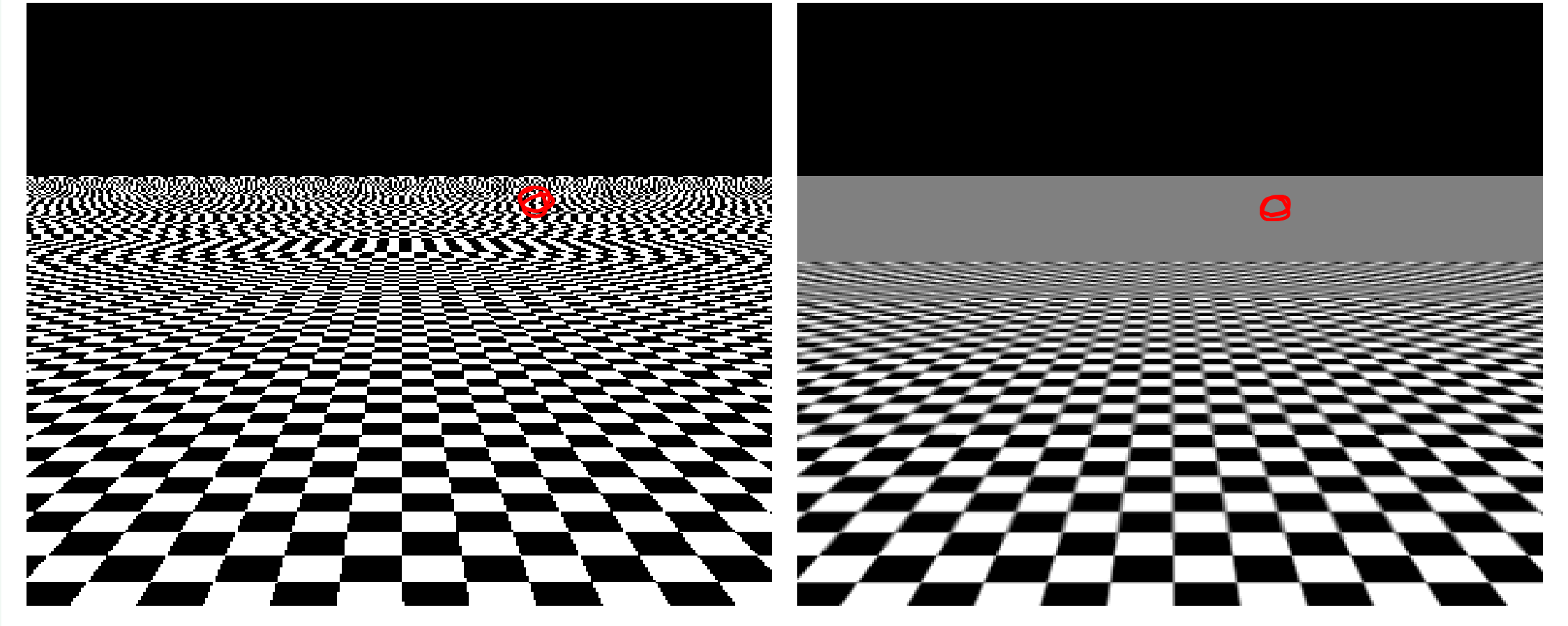
Filtering: The Basic Idea

We need to average together all the texture pixels (texels) the pixel covers

Lots of theory on how to do this correctly...



Simple Filtering



Filtering: The practical problem

We need to average together a lot of texels

What to average?

How to average it **fast**?

The Secrets of performance...

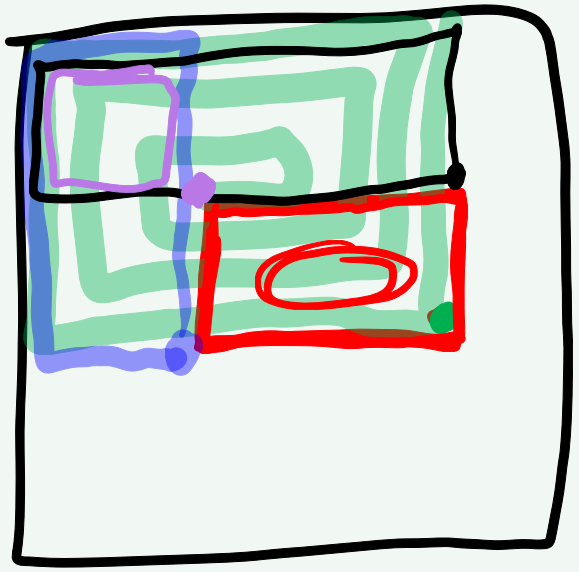
1. Pre-computation
2. Amortization
3. Approximation

We'll see all of these...

Solution 1: Summed Area Table

This is a historical solution - not really used in practice

1. Estimate Shape as a rectangle
2. Pre-compute Summed Area Table
3. Fast lookups (4 values)



Texture Map

Summed Area Table

Key Idea: Amortization and Pre-Computation

We save time per-pixel by pre-computing the summed area table

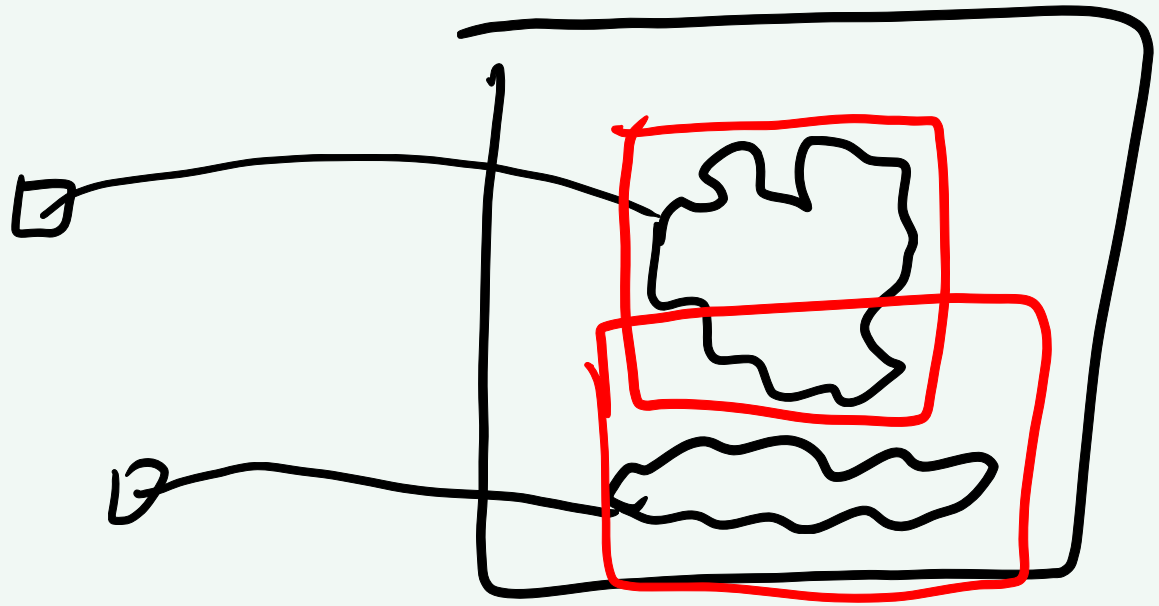
Solution 2: Mip-Maps

De-Facto Standard in Graphics

Modern hardware can do fancier stuff

1. Approximate Filter as a Square
2. Pre-Compute Multiple Sizes of Map
3. Look up in correct sized maps

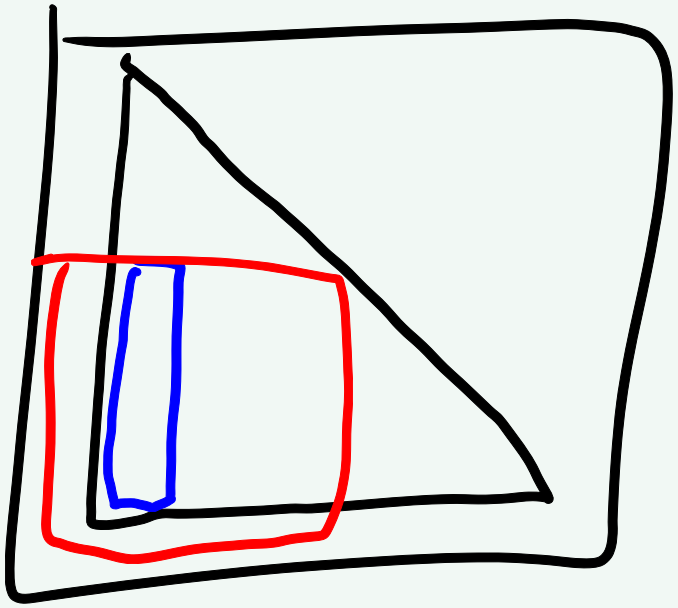
Mip Maps 1: Approximate as square

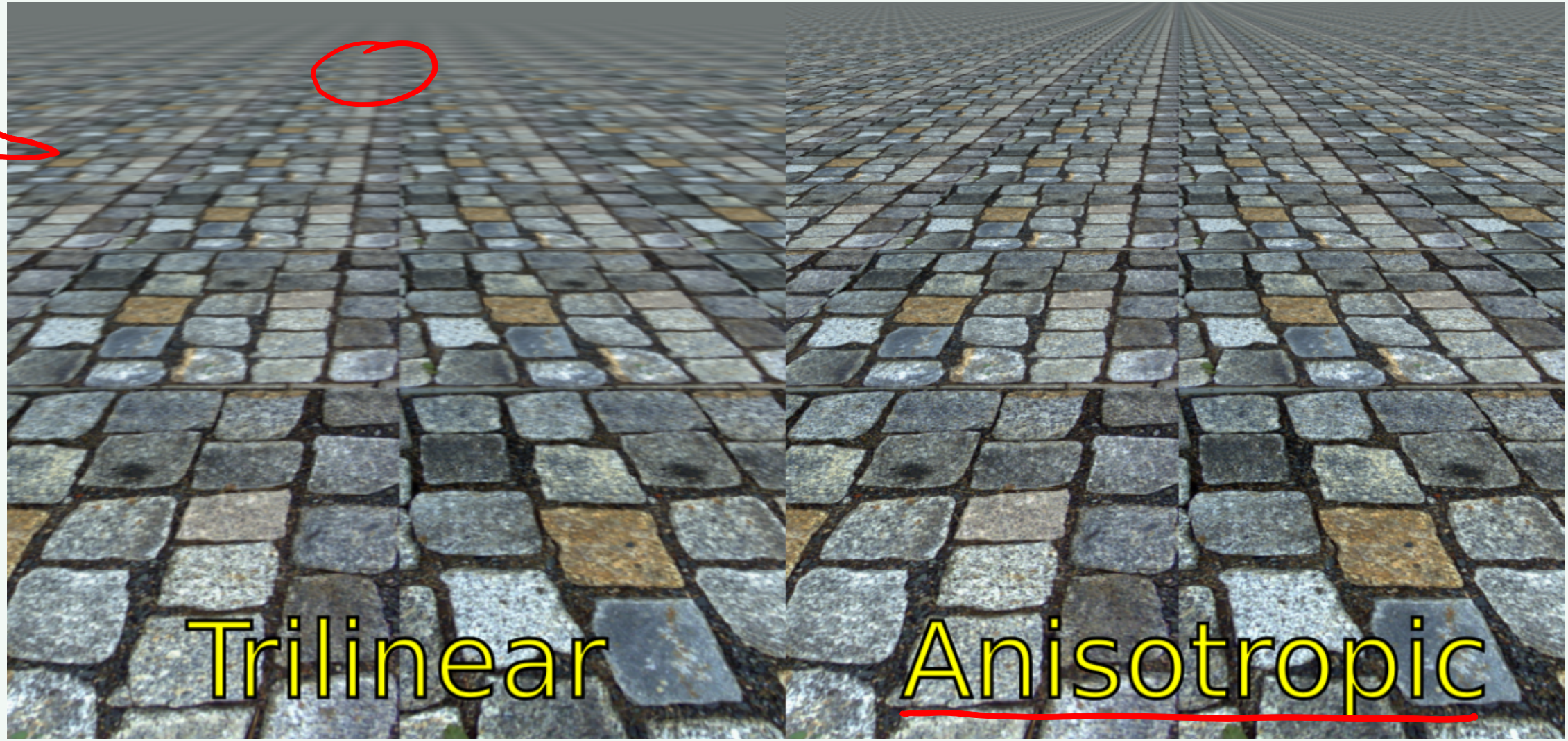


Mip Maps 1: Approximate as square

Problem: Anisotropy

The areas are not square!

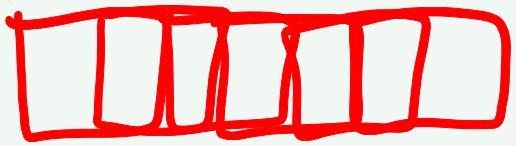


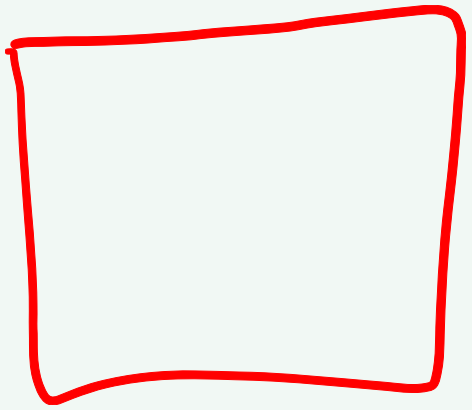

Trilinear
Anisotropic

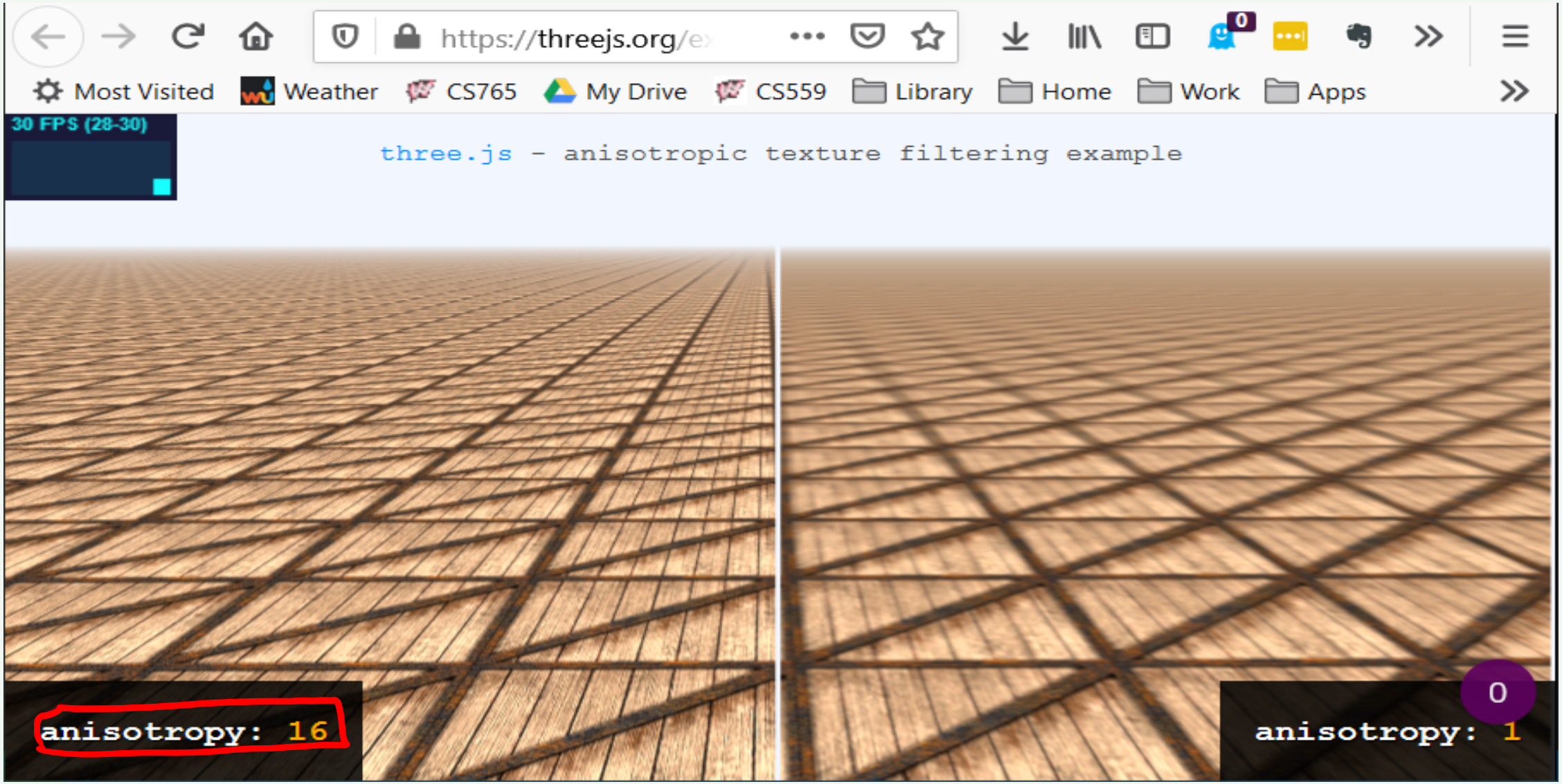
Anisotropic Filtering

Hack version:

Use multiple squares (three supports this)



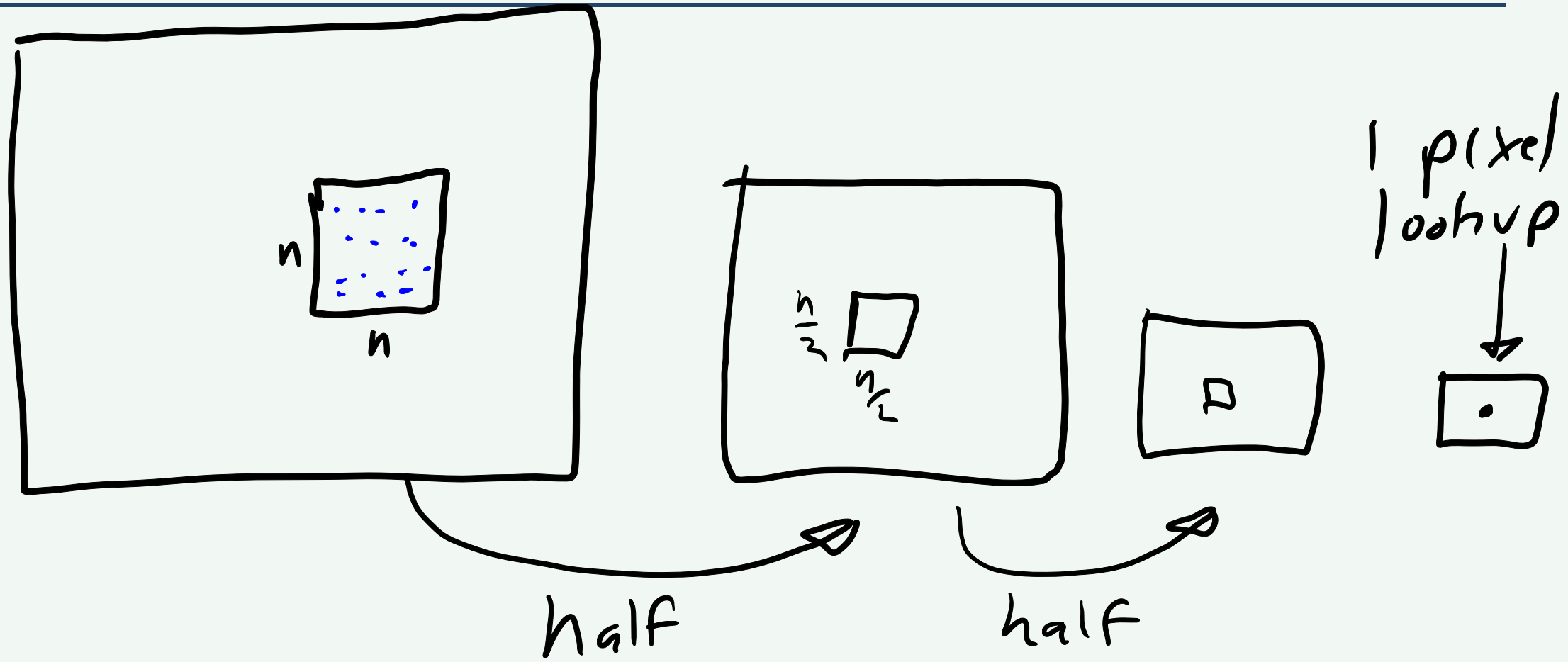




16 squares

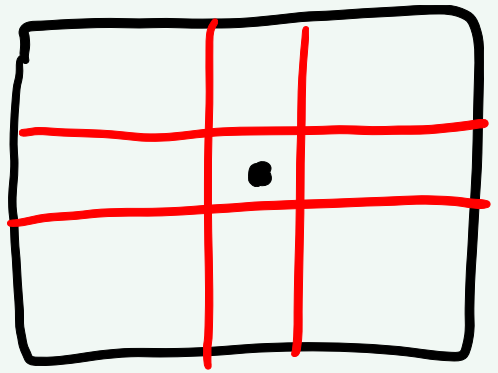
no - anisotropy

Mip Maps 2: Lookup in smaller image



Mip Maps 3: Lookup in smallest image

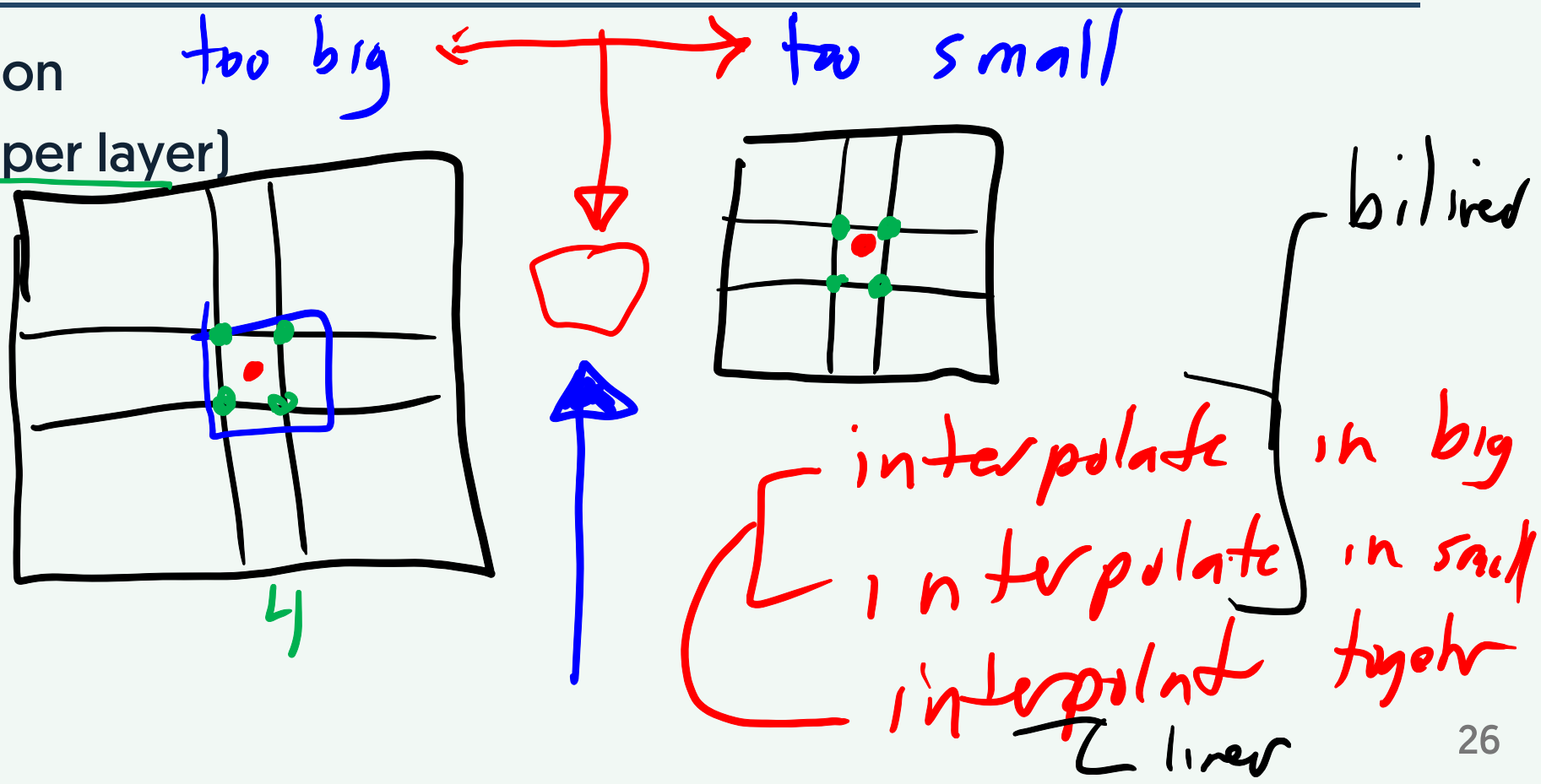
Bi-Linear Interpolation



Mip Maps 4: Lookup in-between images

Tri-Linear Interpolation

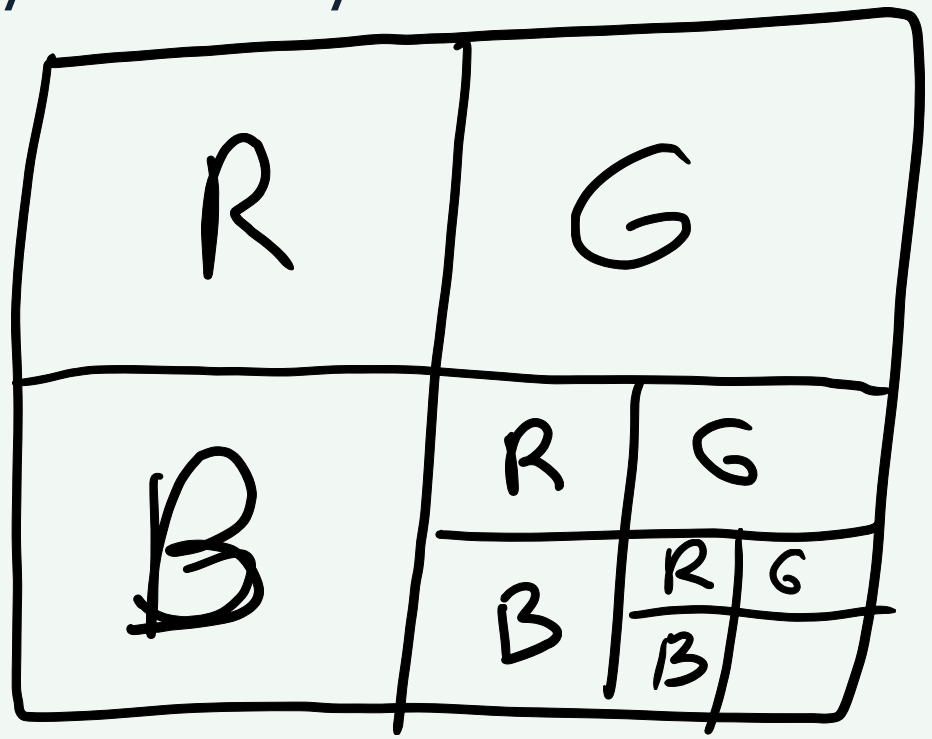
requires 8 values (4 per layer)

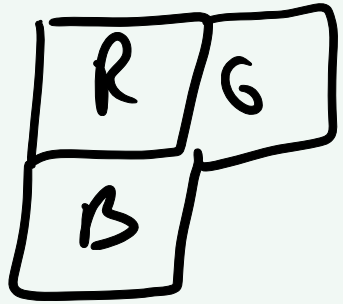


Mip Maps 5: A Historic Storage Trick

Why is it called "MIP"?

We don't actually do this any more...





Mip Maps Summary

1. Pre-Compute MIP-Map (images of various sizes)
2. Aproximate area with square (center is easy, size is harder)
3. Figure out which image to sample from
4. Tri-Linear Interpolate between levels

In Practice...

#1 must be done at texture loading (THREE does it for us by default)

#2-#4 is done by hardware - per pixel

THREE Options

Minification

THREE.NearestFilter

THREE.NearestMipMapNearestFilter

THREE.NearestMipMapLinearFilter

THREE.LinearFilter

THREE.LinearMipMapNearestFilter

THREE.LinearMipMapLinearFilter

tri

Magnification

THREE.NearestFilter

THREE.LinearFilter

bi

Texture Mapping Summary

1. Define UVs for Triangles
2. Lookup values in an image
3. Use images for coloring materials

And inside...

1. Use Barycentric interpolation for UVs
2. Use image sampling to lookup values
3. Use MIP Maps to filter efficiently

Next time...

Facier uses of texture maps!