CS559 Lecture 18: Texture (1)

Part A: Introduction

- A.1 Roadmap
- A.2 JS Tip of the Day
- A.3 Lighting and Meshes Review

Part B: Texture Mapping Concepts

Part C: Texture Mapping in THREE

Part D: How Texturing Works

Lecture 18 Part A: Starting Stuff

Announcements are better done via Canvas

Online Announcements are very important!

Roadmap

Last Time

Meshes (collections of triangles)

Lighting (what color do things appear)

Today: Texture

Many colors per triangle

Up Next...

More Texturing

Fancy versions of texturing

How to use it for effects

Graphics Hardware

How does 3D drawing work?

How do we program the hardware?

JavaScript Tip of the Day

Inheritance is important for the workbook

You will make your own **subclasses** of the framework class

(there is a tutorial on the course web, will post to Piazza)

Classes in Javascript

```
class Parent {
    constructor(a,b) {
        this.a = a;
        this.b = b;
        this.c = 10;
    }
    method() {
        console.log(this.a,this.c)
    }
};
```

```
let thing1 = new Parent(1,2);
thing1.method(); // prints 1,10
```

SubClasses in Javascript

```
class Parent {
    constructor(a,b) {
        this.a = a;
        this.b = b;
        this.c = 10;
    }
    method() {
        console.log(this.a,this.c);
    }
};
```

```
class Child extends Parent {
    constructor(b) {
        super(3,b);
        this.c = 20;
    }
}
```

```
let thing1 = new Parent(1,2);
thing1.method(); // prints 1,10
```

```
let thing2 = new Child(5);
thing1.method(); // prints 3,20
```

SubClasses in Javascript

Child class **extends** parent class

Child class has its own constructor

Child constructor calls parent

`super()` - takes parent arguments

`this` doesn't exist until `super()`

```
class Child extends Parent {
    constructor(b) {
        super(3,b);
        this.c = 20;
    }
}
```

Child class uses parent methods (unless it overrides them)

```
let thing2 = new Child(5);
thing1.method(); // prints 3,20
```

Why do you need to know this?

The CS559 Software Framework uses this!

You define types (subclasses) of `GrObject`

`GrObject` has a list of THREE `Object3D`

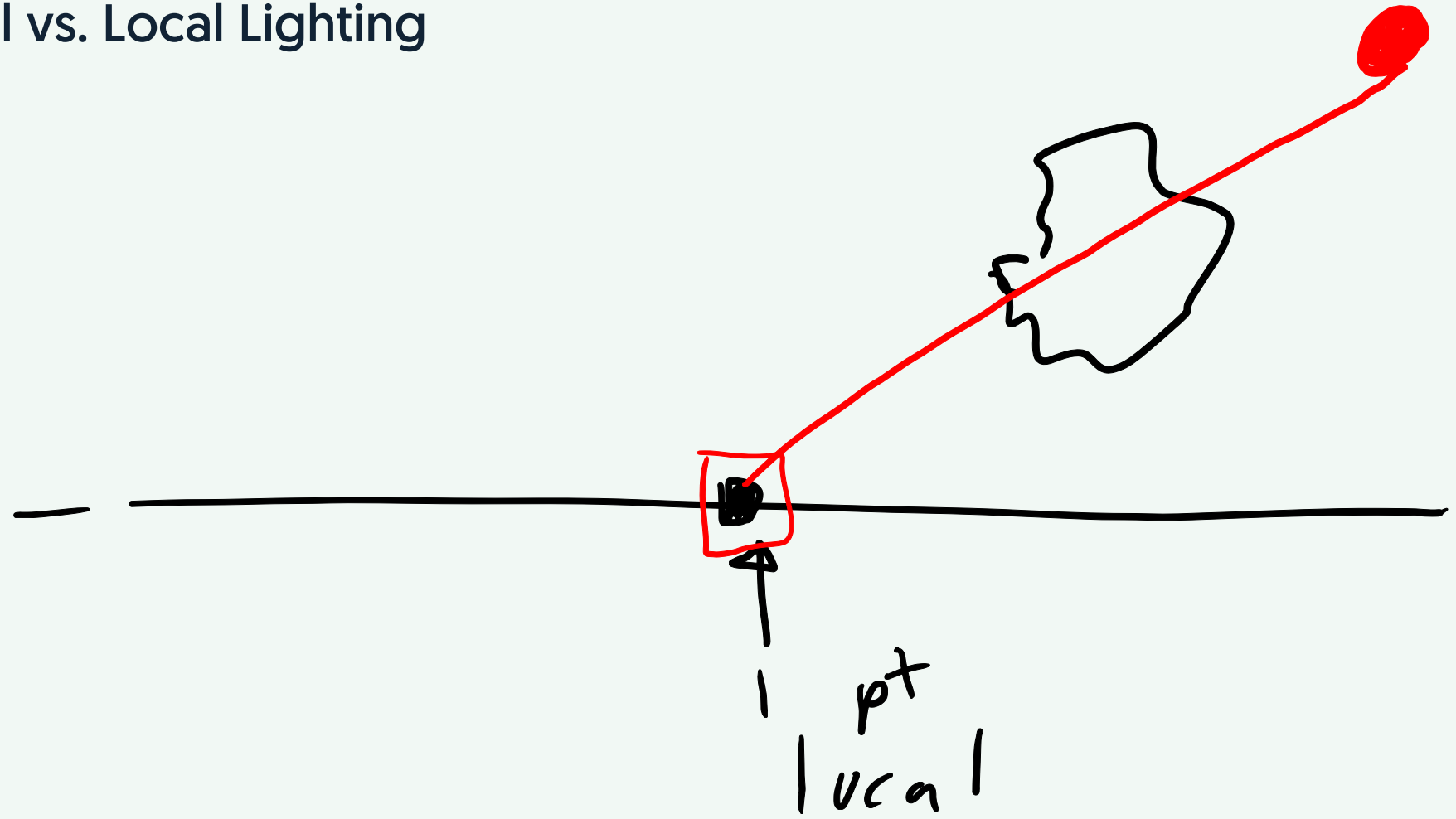
You pass the `GrObject` constructor the `Object3D` it should contain

```
class BasicSphere extends GrObject {
    constructor() {
        let geom = new T.SphereGeometry();
        let mat = new T.BasicMaterial({color:"green"});
        let mesh = new T.Mesh(geom,mat);
        super("Basic Sphere", mesh);
        this.mesh = mesh;
    }
}
```

Back to Graphics...

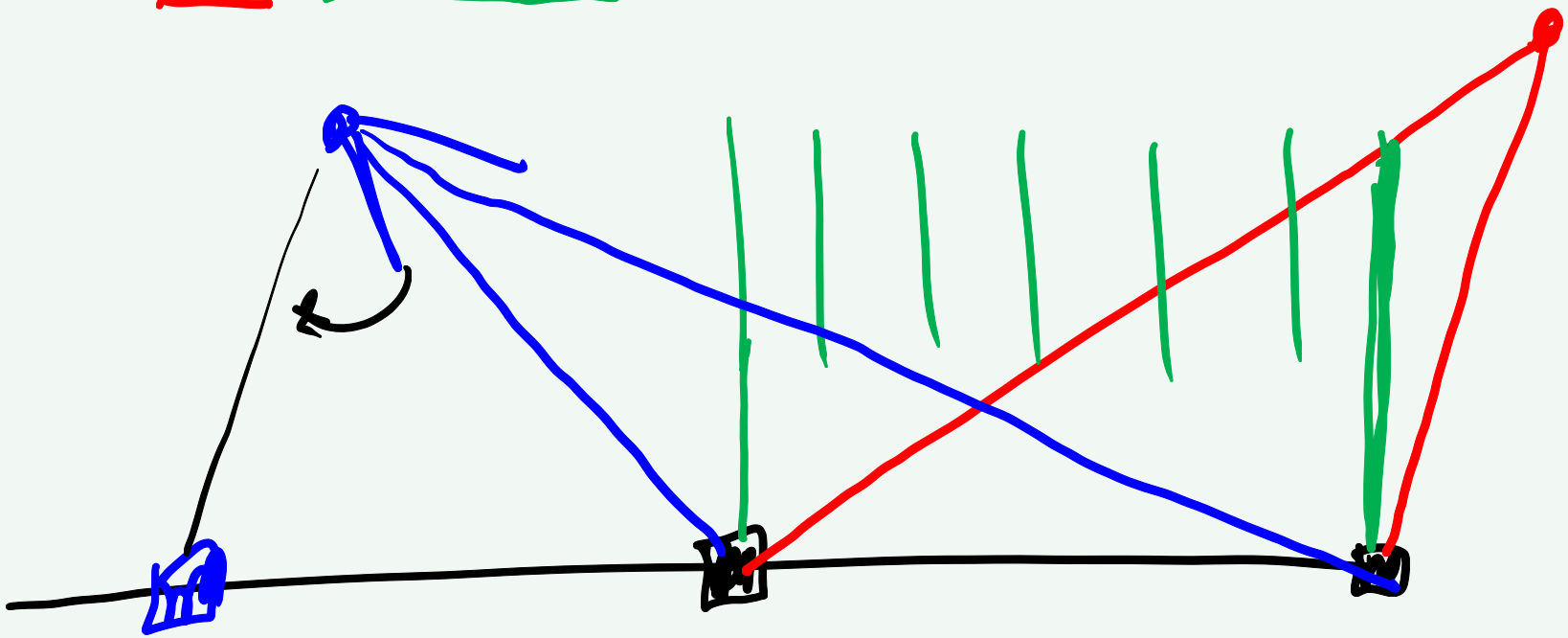
Lighting Review

Global vs. Local Lighting

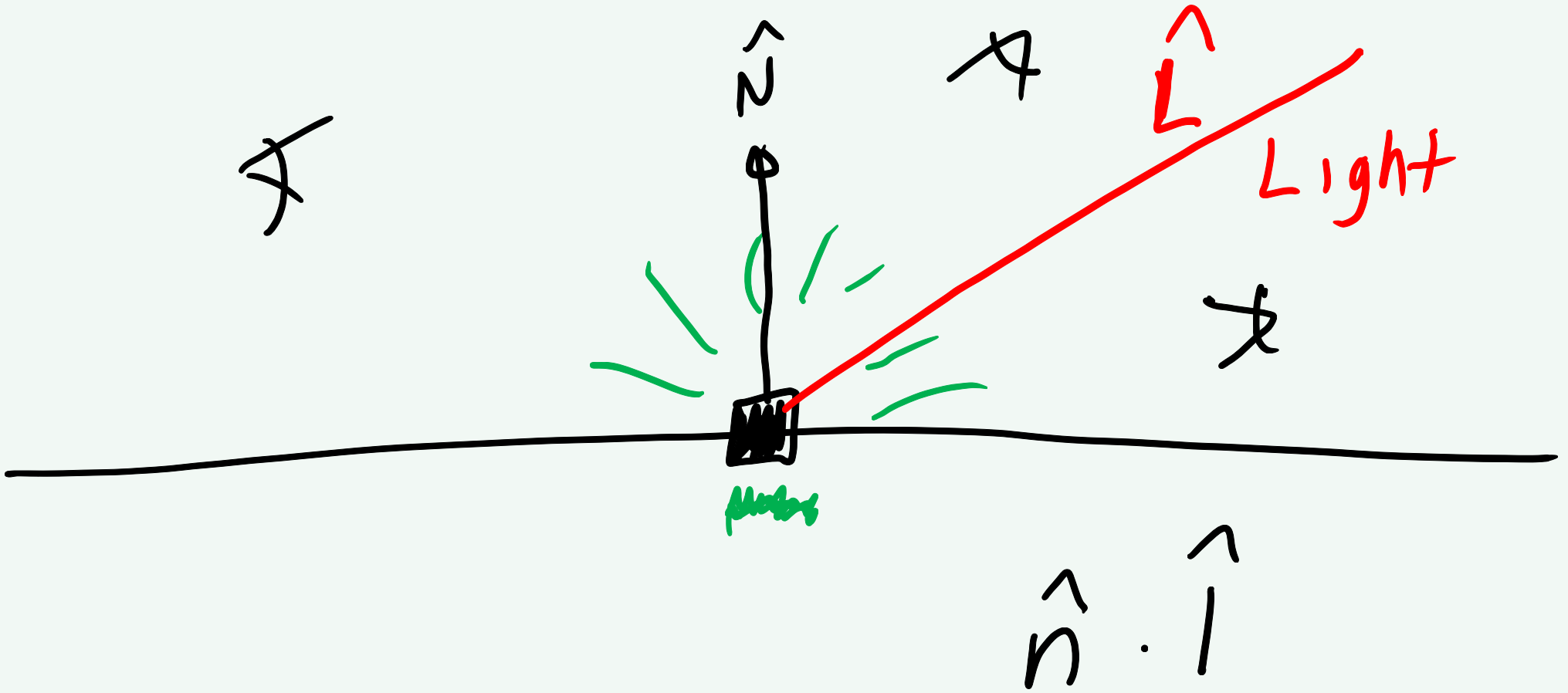


Lighting Review

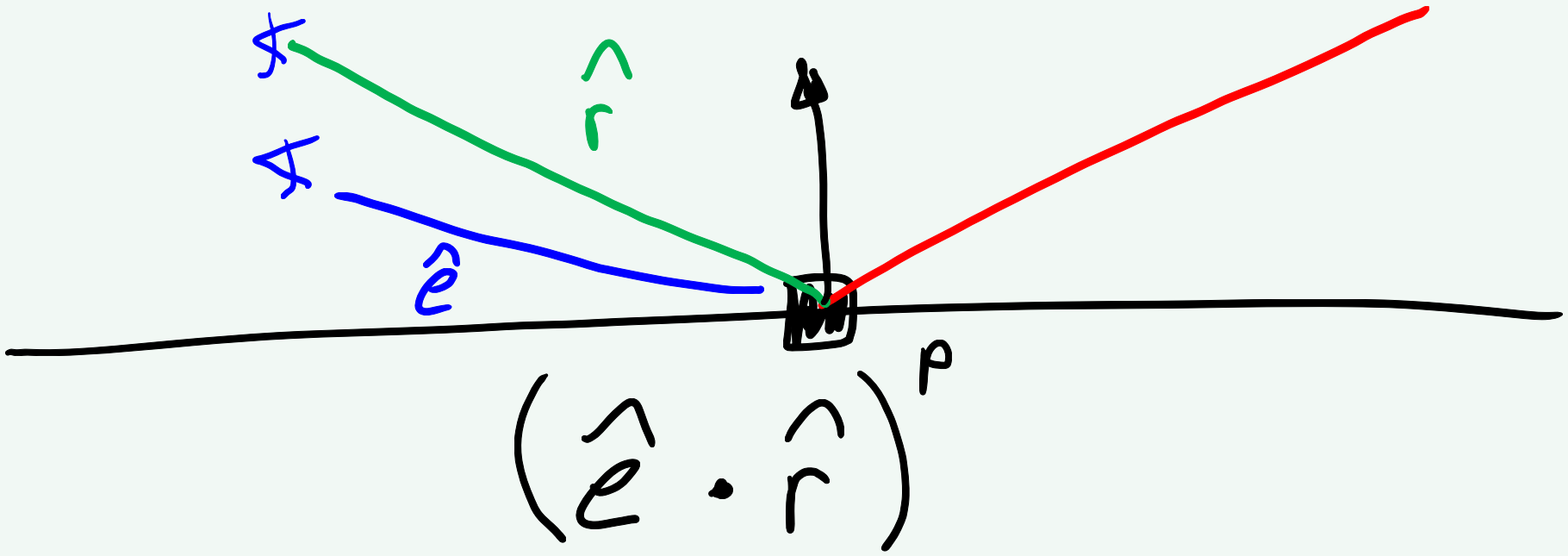
Light Geometry: Point, Directional, Spotlight



Lighting Review: Diffuse



Lighting Review: Specular — shiny



Lighting Review: Material Properties

C_S C_D C_A C_E

What Color are we specifying?

Model has 4 colors:

- emissive
- ambient
- diffuse
- specular

Material model allows us to specify all of them.

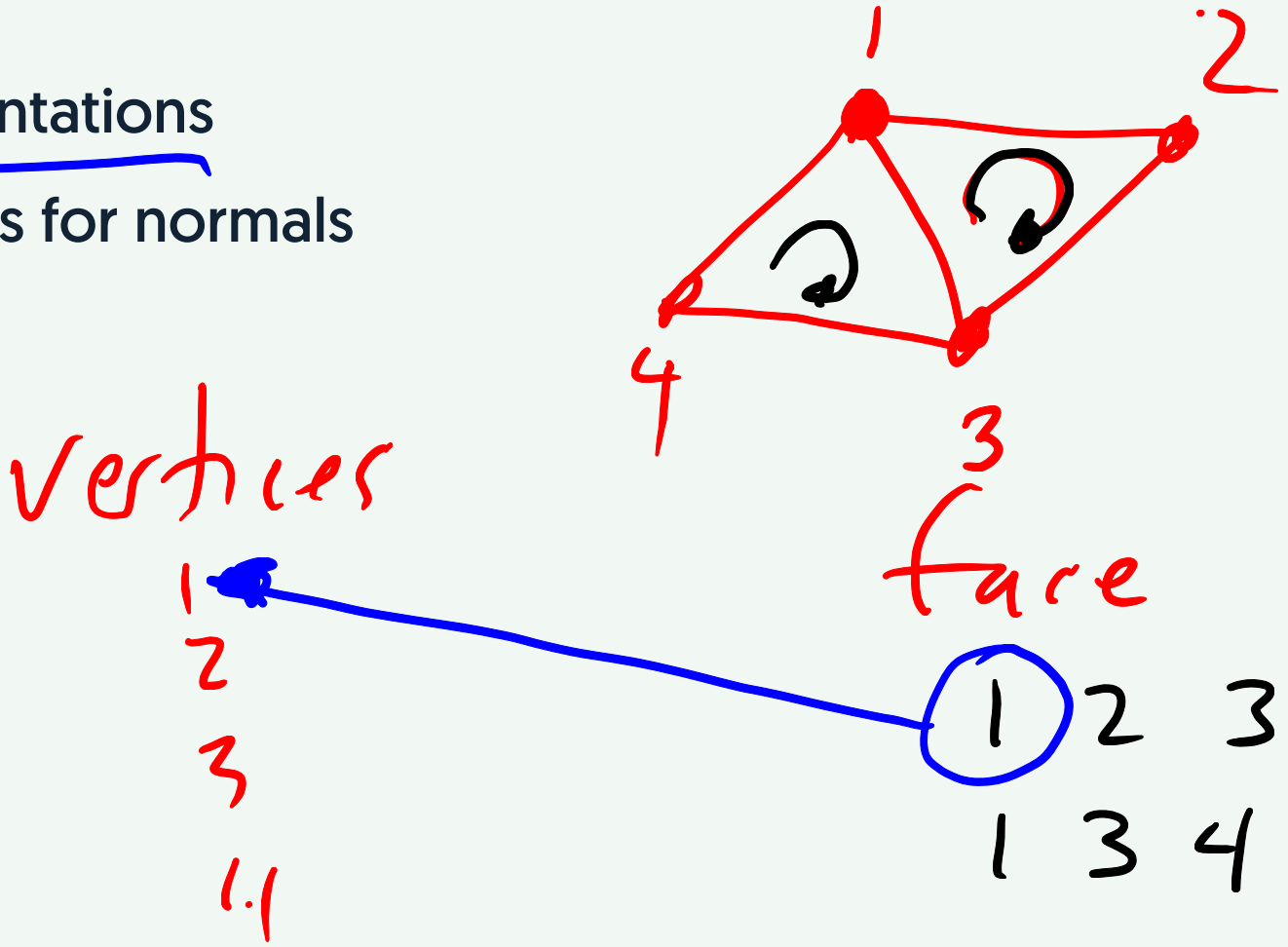
Typically 1 "main" color - ambient and diffuse

- other colors specified specifically

What are we coloring?

Triangle Meshes!

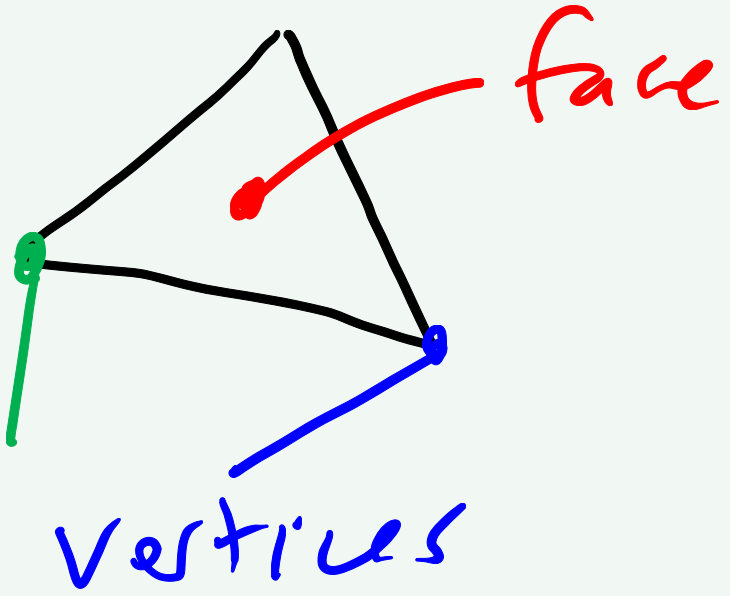
- index set representations
- consistent meshes for normals



Where do we put colors?

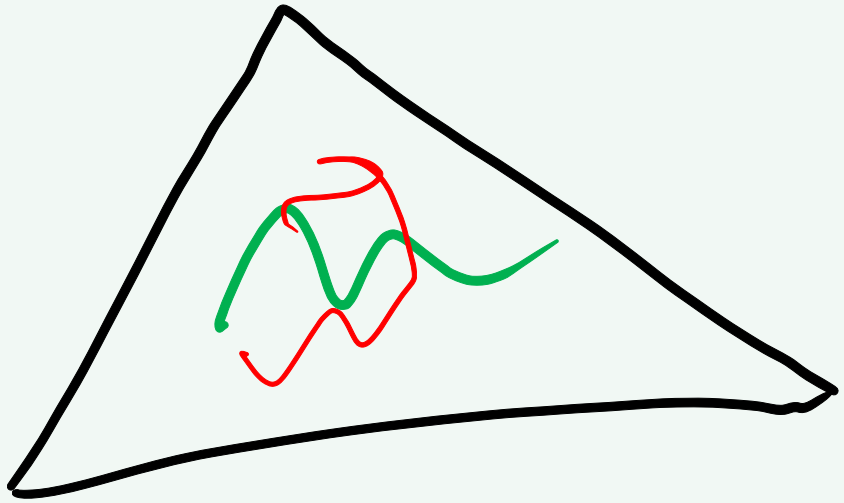
- A Color per **object** (material's color)
- A Color per **face**
- A Color per **vertex** (interpolate across faces)

How do we get more colors per triangle?



Lecture 18 Part B: Texture Basics

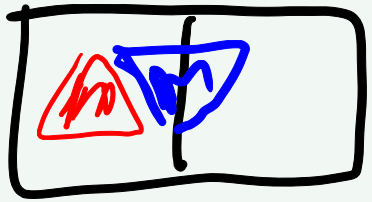
How to get more than 3 colors on a triangle?

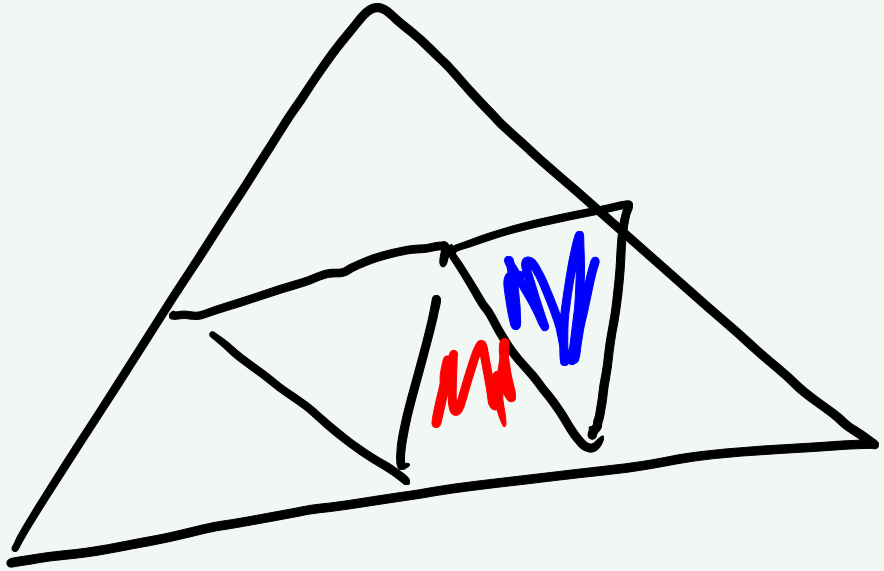


Bad Idea #1

Break the triangle into smaller pieces?

- need to keep track of little triangles
- need to transform/process little triangles
- a lot to draw
- sampling issues (triangles vs. pixels)

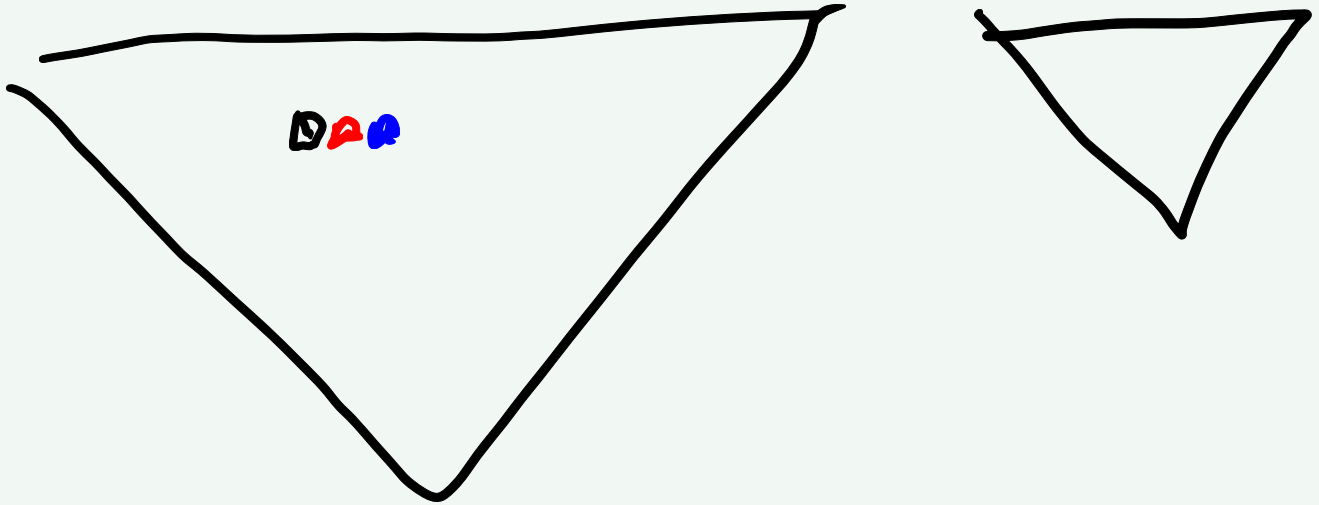




Bad Idea #2

Colors per Pixel?

- How do you know the pixels ahead of time?

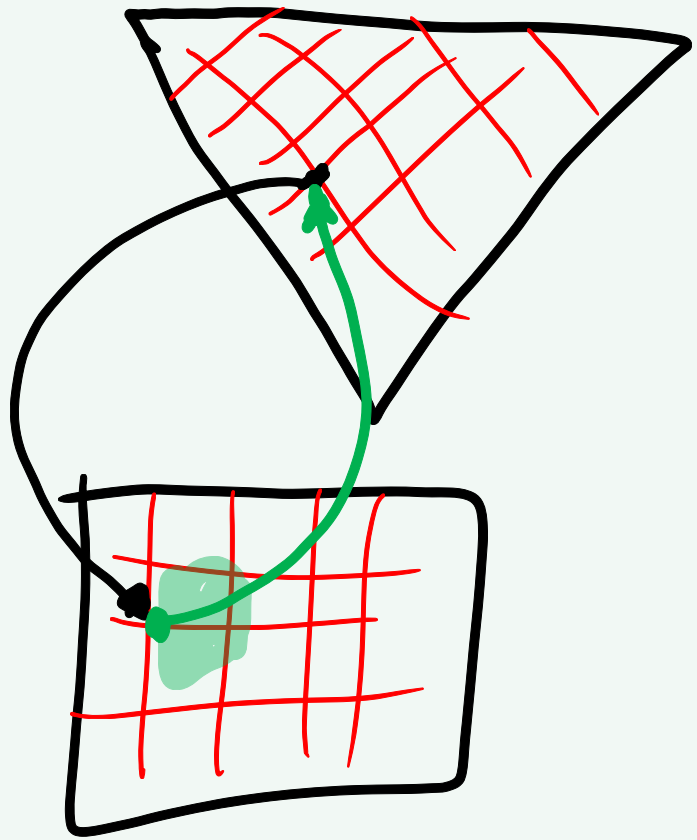


Texture Mapping: Basic Idea

1. Define a coordinate system on the triangle
2. Define a map from coordinate to color

When we draw, each pixel:

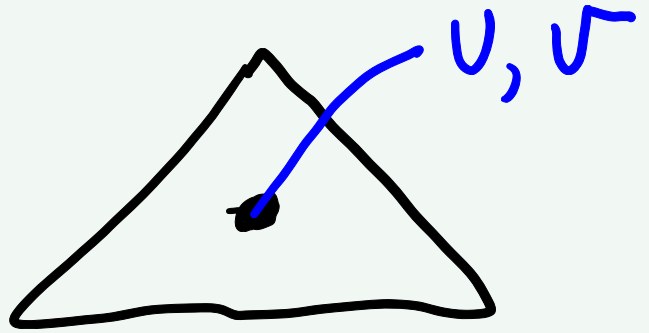
1. figures out its coordinate
2. looks up its color (*)



Texture Mapping

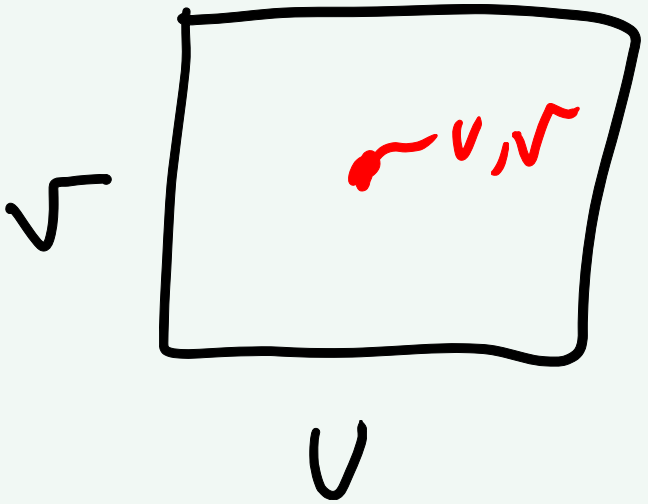
Defined by three parts:

1. Some way to assign coordinates
2. Some way to look up the value for that coordinate
3. Something to do with the value when we look it up



Most common/basic form:

1. UV coordinates on triangles (2D coord per point)
2. Images that define colors (lookup color in 2D)
3. Use looked up color as the color for lighting



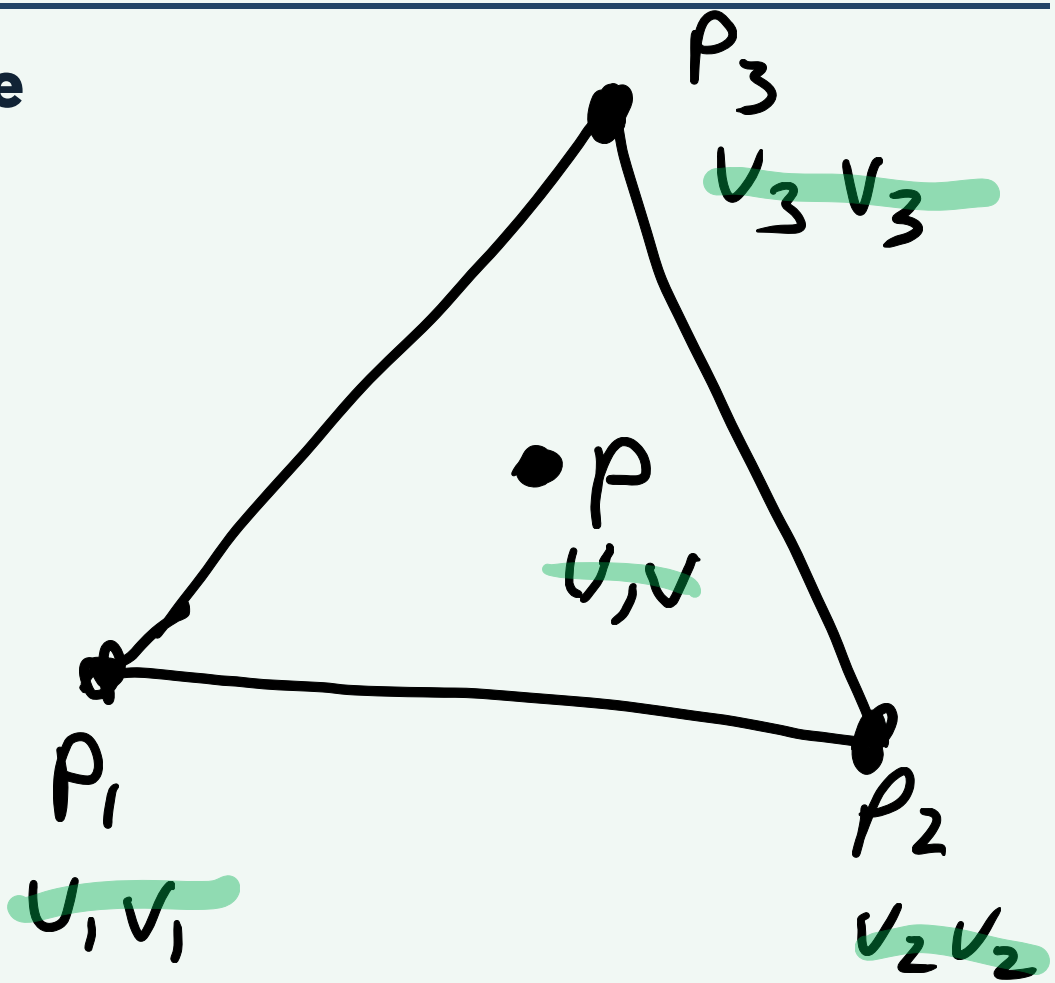
Basic Texture Mapping

1. **Define UV coordinates for the triangle**
2. Use an image to map UV to color
3. Use the color as the surface color

$$p = \alpha p_1 + \beta p_2 + \gamma p_3$$

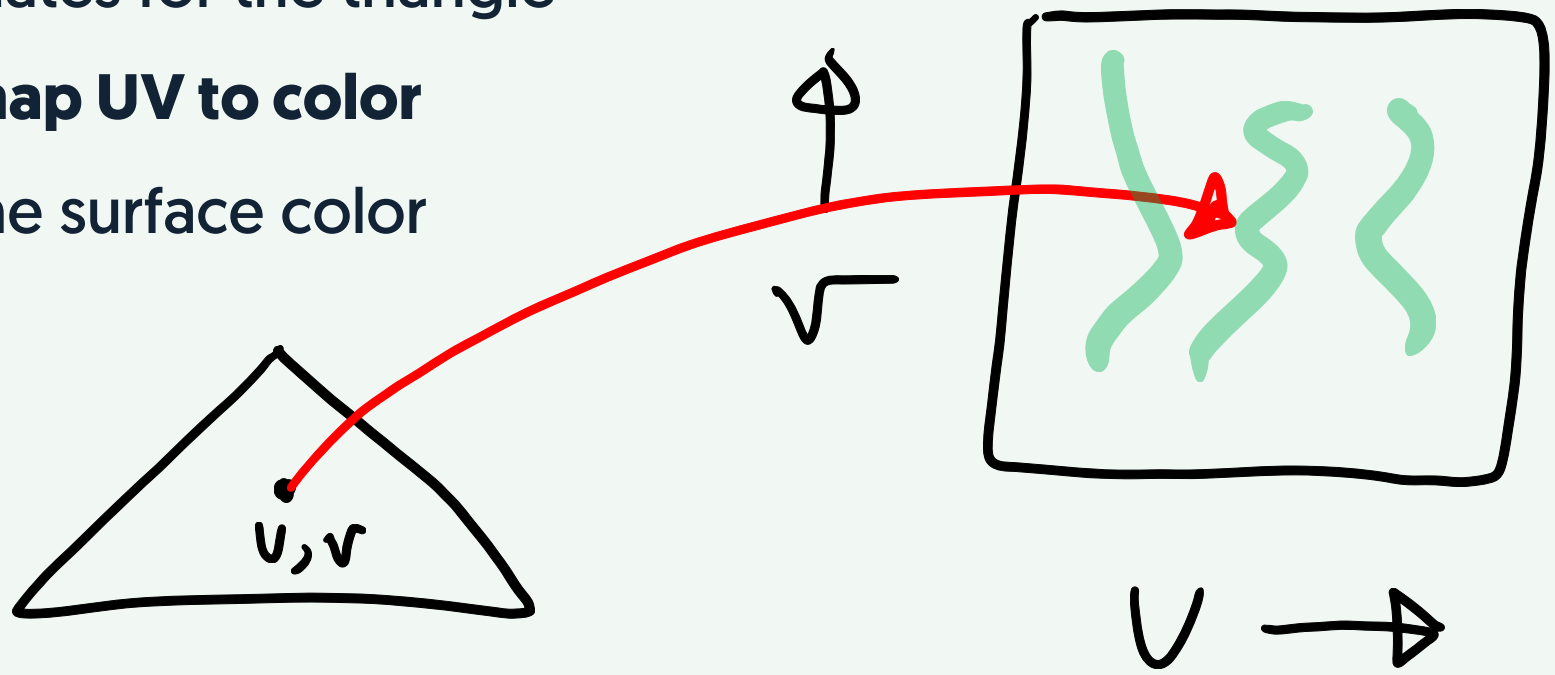
$$uv = \alpha u + \beta u + \gamma u$$

$$v \quad + \quad v \quad + \quad v$$



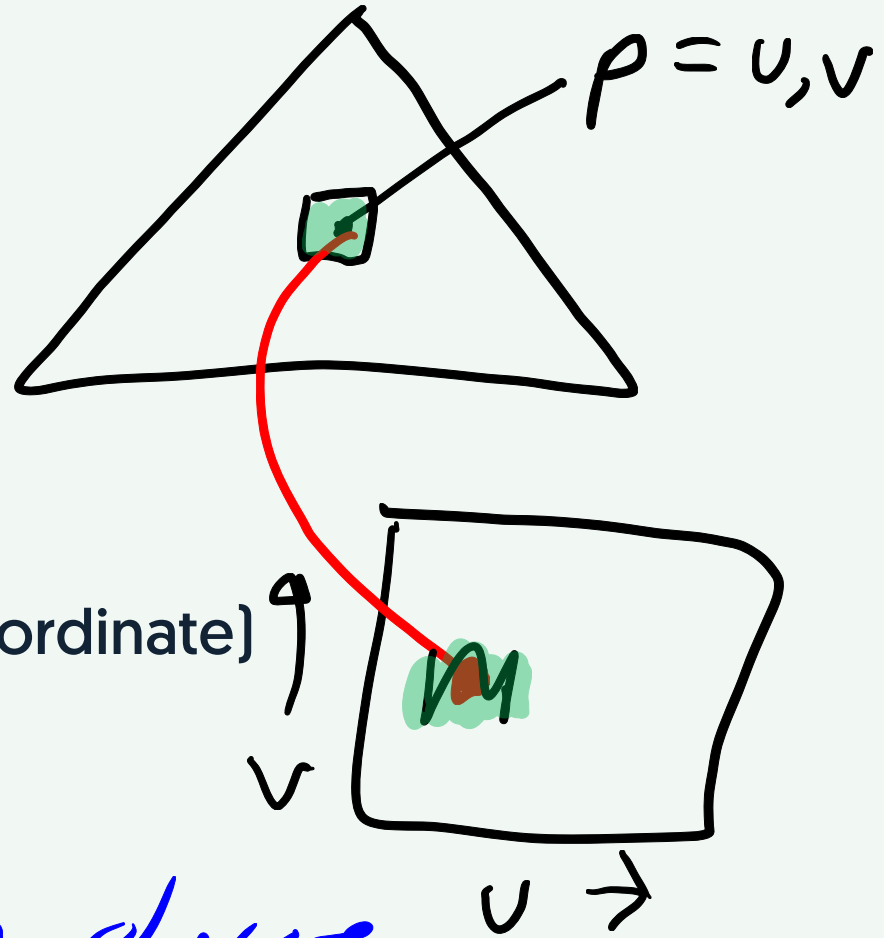
Basic Texture Mapping

1. Define UV coordinates for the triangle
2. **Use an image to map UV to color**
3. Use the color as the surface color



Basic Texture Mapping

1. Define UV coordinates for the triangle
2. Use an image to map UV to color
3. **Use the color as the surface color**



For each pixel inside triangle:

1. get the Barycentric coodinate
2. use this to interpolate the UV values (get UV coordinate)
3. use the UV coordinate to look up the color
4. use the color as the surface color

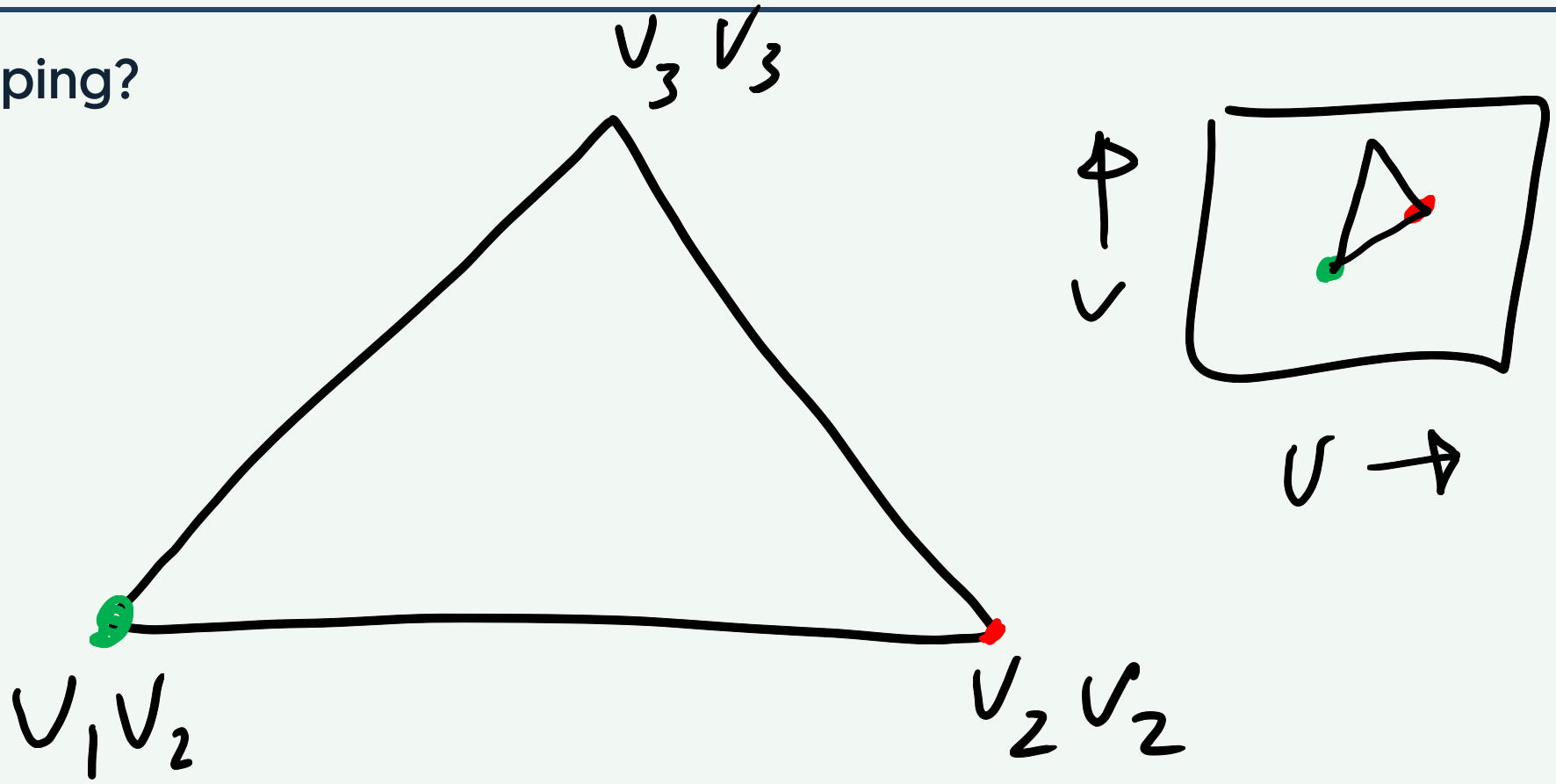
inside graphics hardware

Demo

(available online)

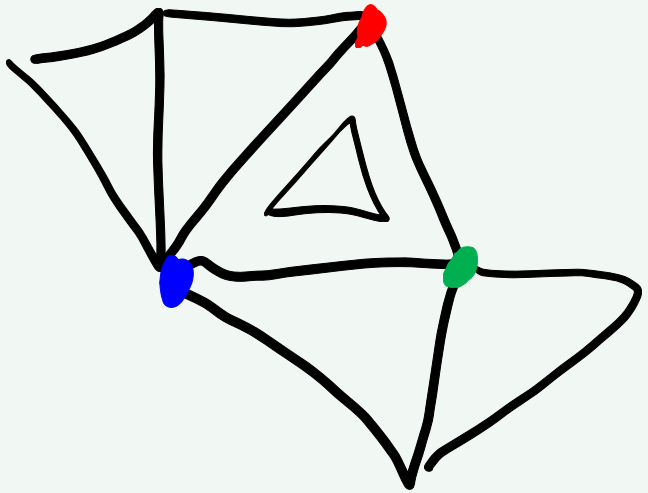
UV Mapping

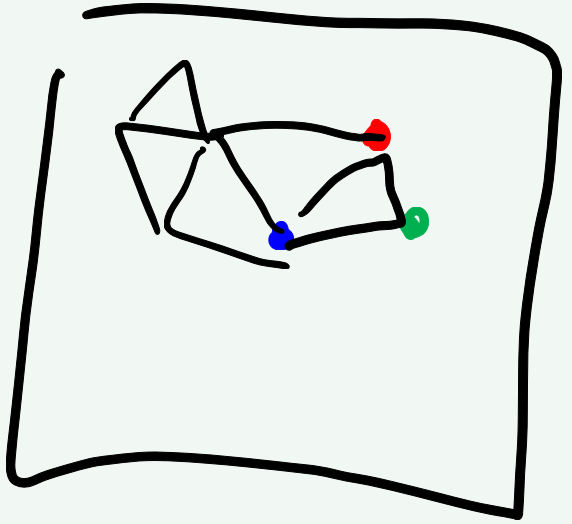
Or is it ST Mapping?



UV mapping and multiple triangles

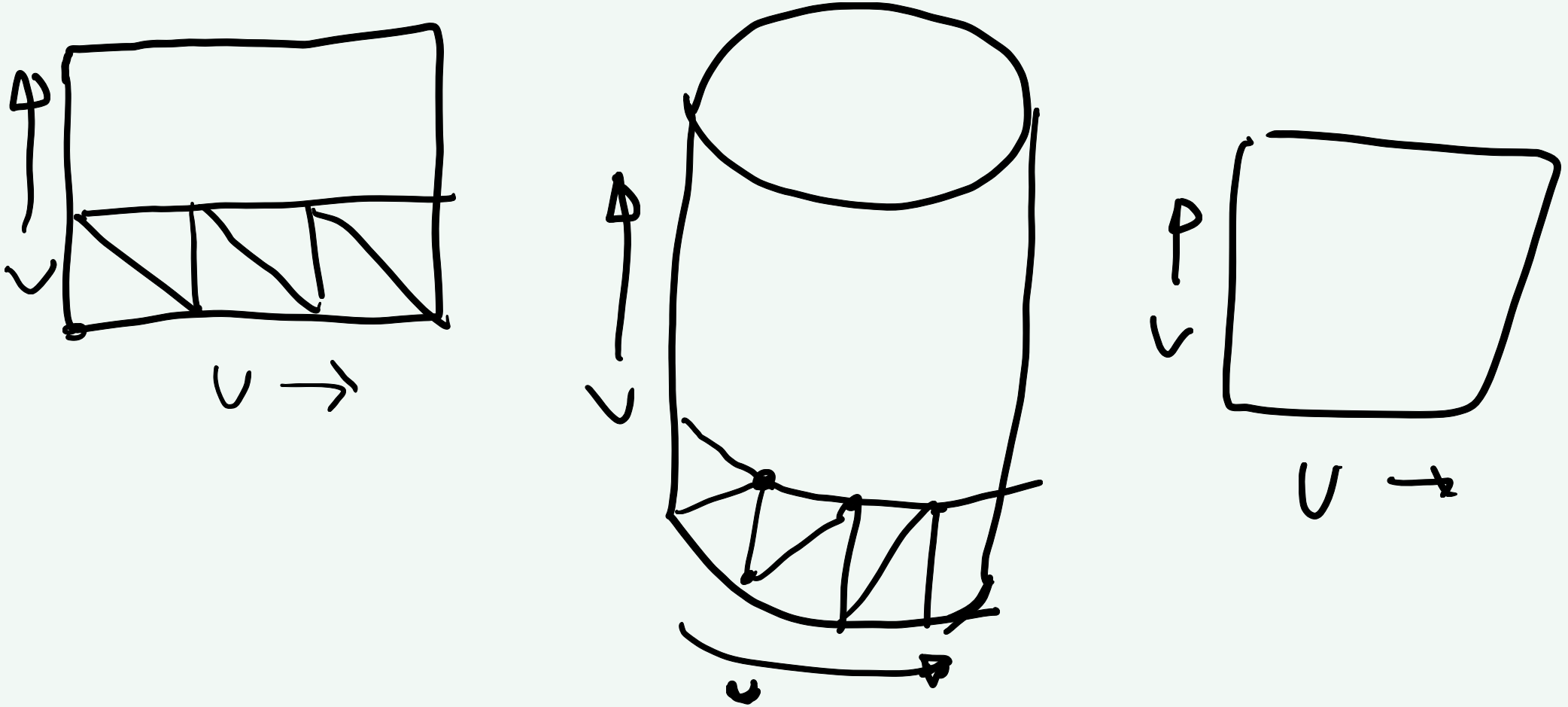
Pack the triangles into the UV plane





UV mapping for standard objects

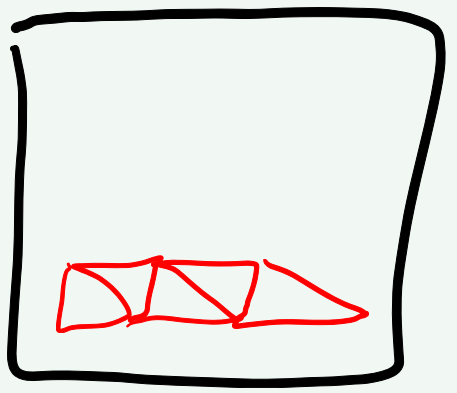
Many objects have a natural parameterization



Where do UVs come from?

We specify the UV values!

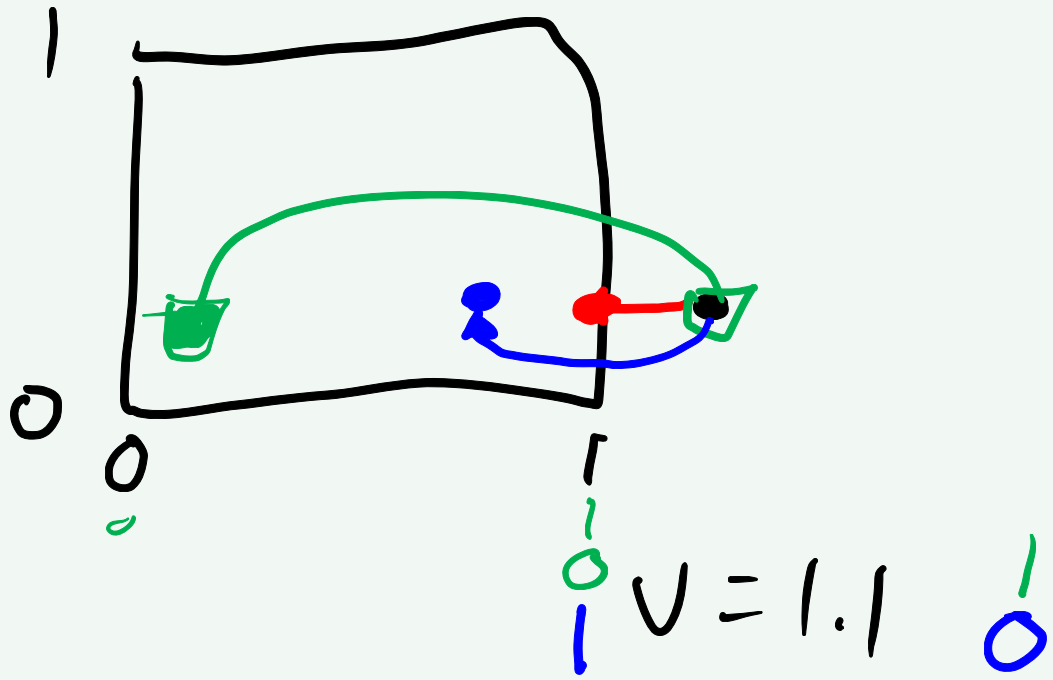
- If you have the image to start with
  - pick UV so the triangle gets the right piece
- If you don't have the image first
  - pick UV so each triangle is "nice" in UV space



Texture Wrapping

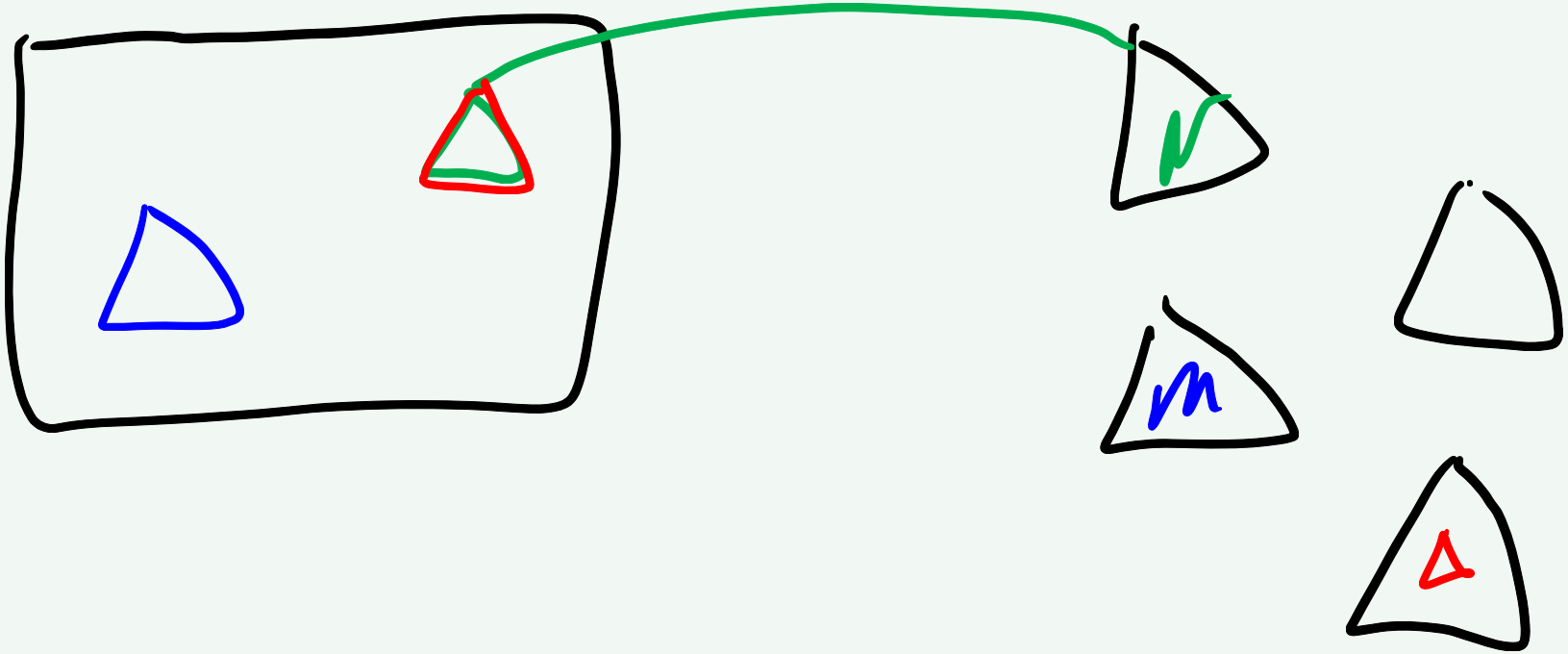
What if U or V outside of $[0, 1]$?

- clamp wrapping
- repeat wrapping
- mirror repeat wrapping



Texture use and Re-Use

Loading, processing and storing textures is resource intensive

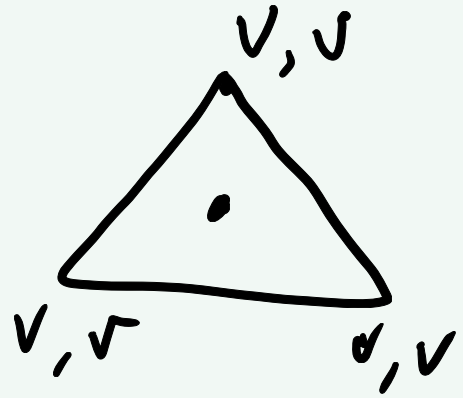


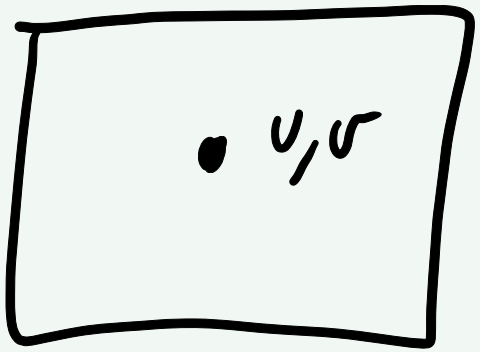
Steps for using Texture Mapping

1. Specify UVs for Triangles
2. Specify an Image to be used for lookup
3. Specify how to use the image in the material

and (a hidden first step)

4. Specify all the different parameters
   lookup type, wrapping mode, etc.





Texture Basics

1. Specify UVs per vertex
2. Look up colors in an image
3. Use looked up value in an image

Texture (Beyond) Basics

1. Specify UVs per vertex
   compute the UVs dynamically
2. Look up colors in an image
   use something other than simple lookup for map
3. Use looked up value as color
   use the values for something other than color

just wait, this we'll see examples of all of these

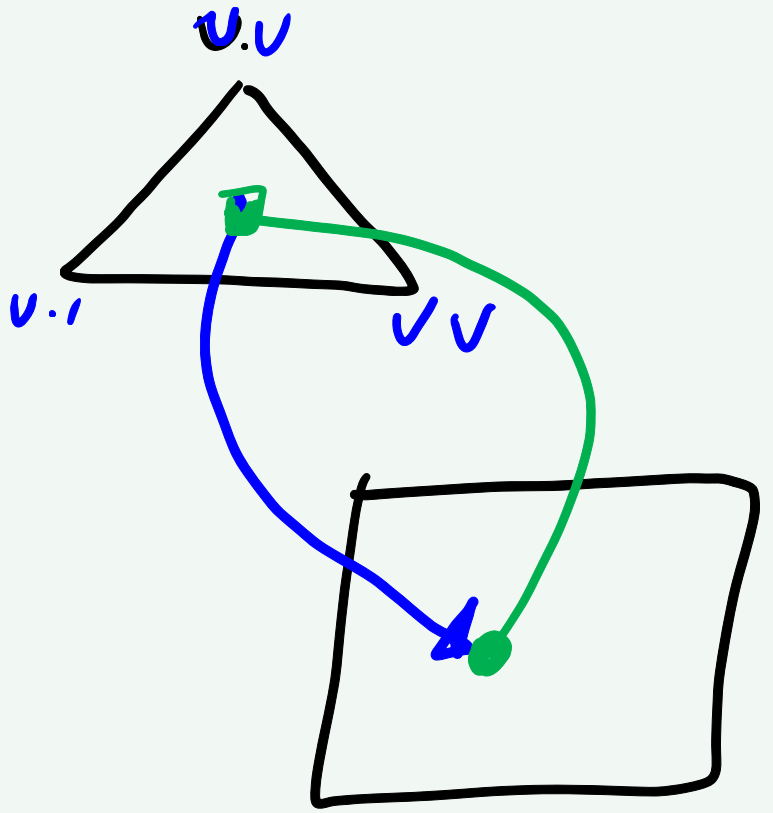
Lecture 18 Part C:
Texture Mapping in THREE

The steps to texture mapping

1. Define UVs for each triangle
2. Specify image to be used for lookup
3. Specify how to use the color information

and

4. make sure all the parameters are set right

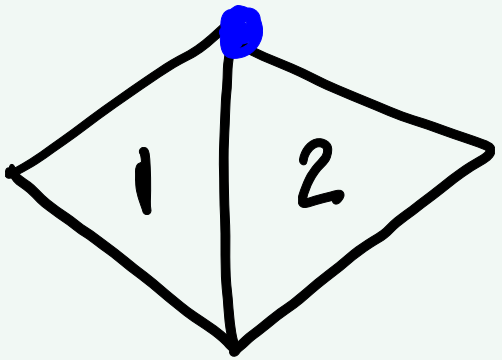


(Basic) Texture Mapping in THREE

- Create objects with UVs
- Load Textures
- Attach as colors to objects

Textures in THREE

- **Create objects with UVs**
- Load Textures
- Attach as colors to objects

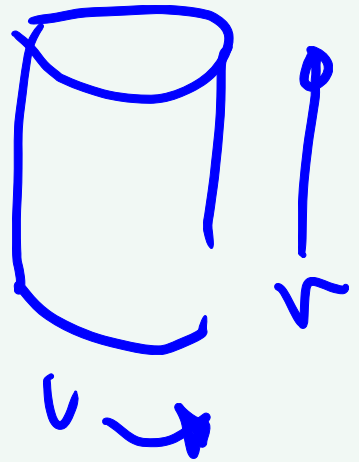


1 vert 1

2 vert 1

Primitives

- have predefined UVs

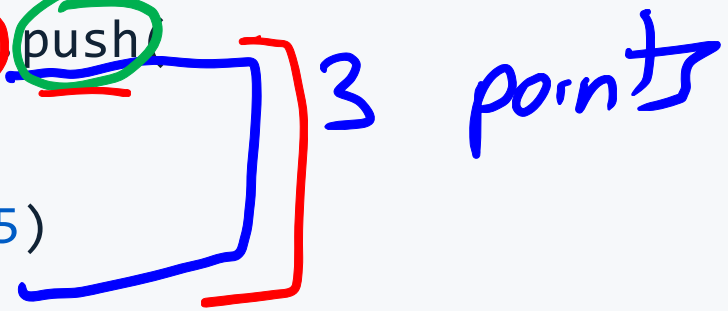


Geometries

- Define UV for each vertex
- Need to define for each face
- Possibility for "layers"
- Not part of faces

UVs for Geometry

```
geometry.faceVertexUvs = [ [] ];
let f1 = new T.Face3(0,1,2);
geometry.faces.push(f1);
geometry.faceVertexUvs[0].push(
  [ new T.Vector2(0,0),
    new T.Vector2(.5,0),
        new T.Vector2(0,.5)
  ]);
```



`faceVertexUvs` is a property of `geometry`

Array of Layers

Each Layer is an Array of Faces

Each Face is an Array of UVs (for each vertex)

UVs for Geometry

```
let vertex1 = new T.Vector2(0,0);          // vertex 2 and 3 defined as well
let face1 = [vertex1, vertex2, vertex3]; // a face is 3 vertices
let layer = [face1];                       // and more faces...
let geometry.faceVertexUvs = [layer];      // we only have one layer
```

1. Each Vertex has a coordinate (2D)
2. Each Face (Triangle) has 3 vertices
3. Each Layer has **N** faces
4. Each Object has 1 layer (or more?)

What is the "Layers" thing?

- Multi-texturing!
- Caveat: it is unclear that THREE materials use this

THREE uses layers for other purposes: Lighting (combines with color)

- Lightmaps
- Ambient Occlusion Maps

We'll talk about these things later

Textures in THREE

- ~~Create objects with UVs~~
- **Load Textures**
- Attach as colors to objects

Loading images may take time

Asynchronous loading

THREE takes care of it!

- texture is blank and fills in later

immediately

```
let tl=new T.TextureLoader().load("./THREE/UV_Grid_Sm.jpg");
```

- re-use these if possible (share between objects)
- different kinds of pre-processing happens (it's not just an image)

Textures in THREE

- ~~Create objects with UVs~~
- ~~Load Textures~~
- **Attach as colors to objects**

Most Materials take **maps**

Color map is called `map`

Material

```
let tl=new T.TextureLoader().load("./THREE/UV_Grid_Sm.jpg");
let material = new T.MeshStandardMaterial({map:tl,roughness:0.75});
```

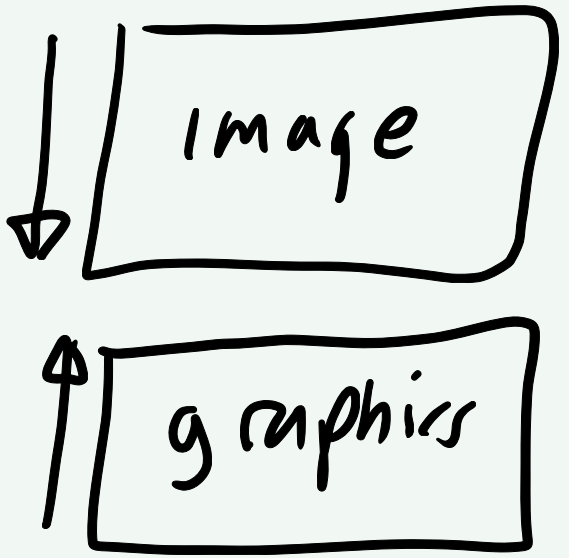
color map

texture map

A **lot** is happening ~~behind the scenes~~.

Some caveats for Textures in THREE

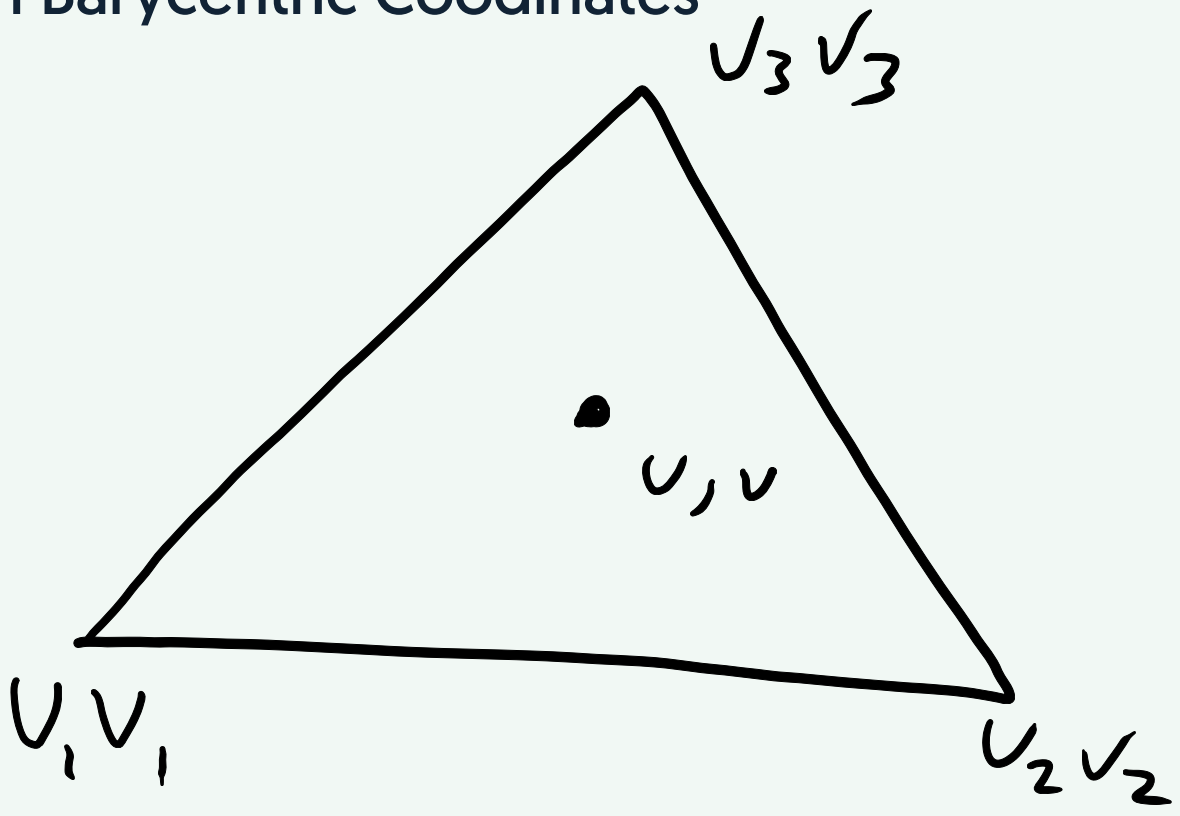
1. Lighting and material affects color too!
    - `MeshBasicMaterial` if you don't want lighting
2. Y is flipped by default



Lecture 18 Part D:
How Texture Mapping Works

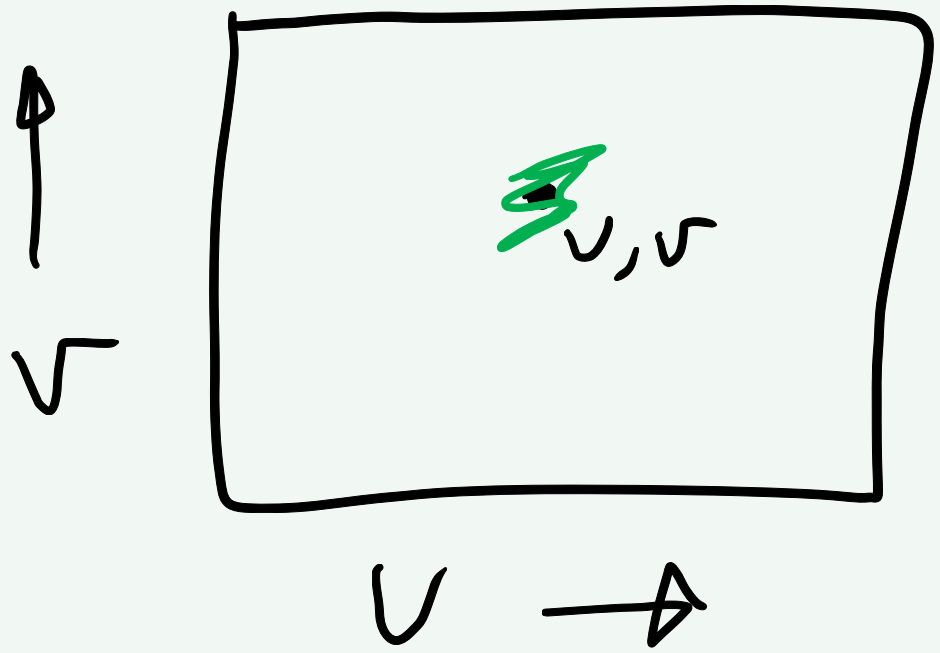
The Texture Coordinate

UV Mapping with Barycentric Coodinates



The Texture Lookup

Given UV what is the color?

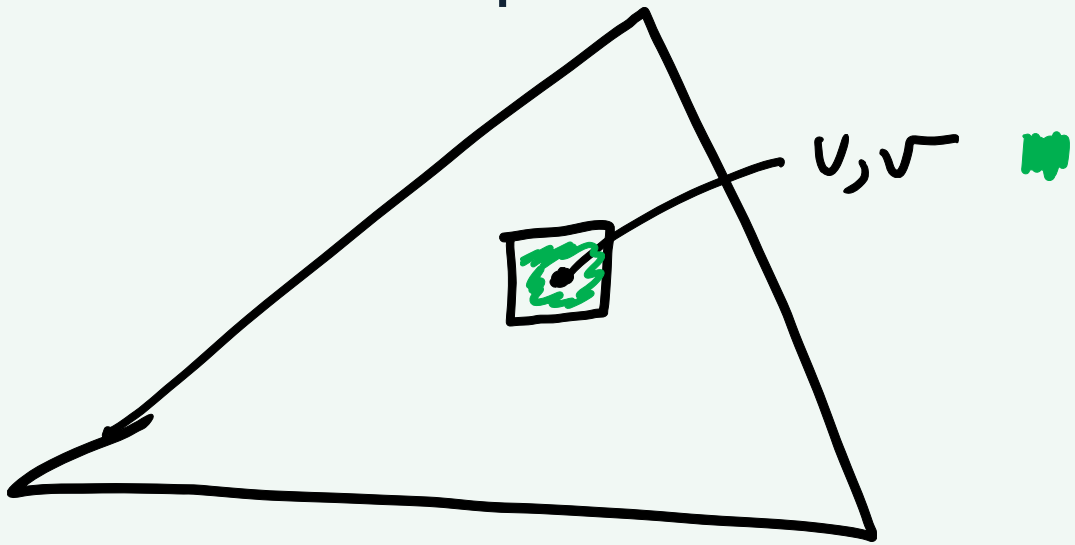


Where does the color go?

Pixel is a little square?

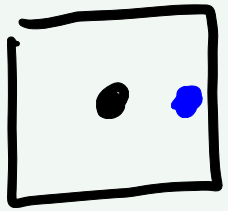
(no, but useful for thinking about it)

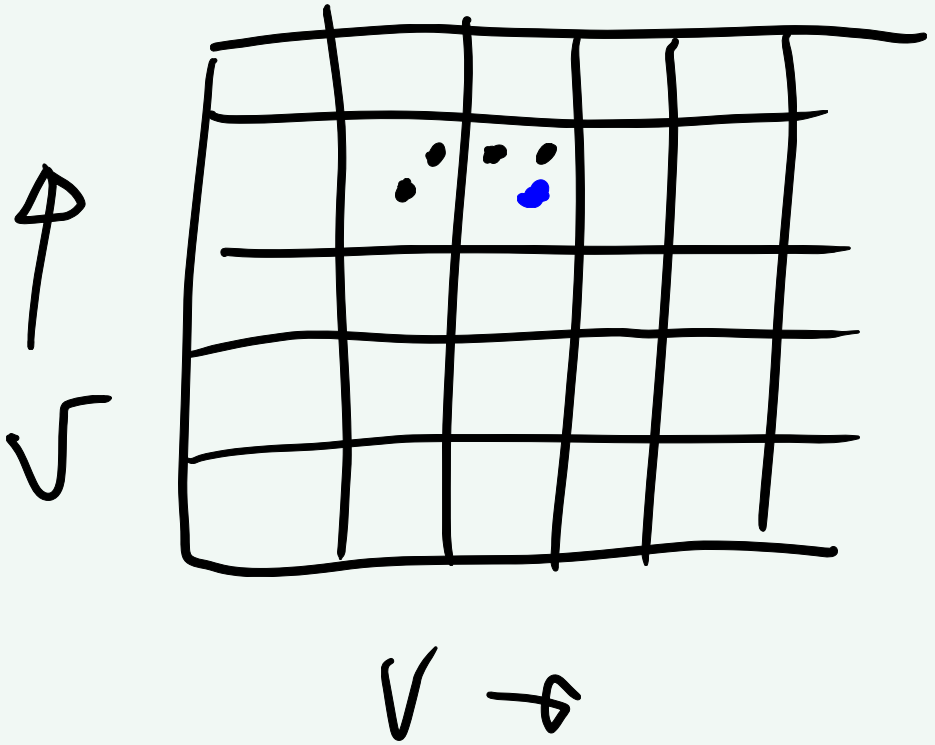
Assume UV is the center of the pixel



The problems

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





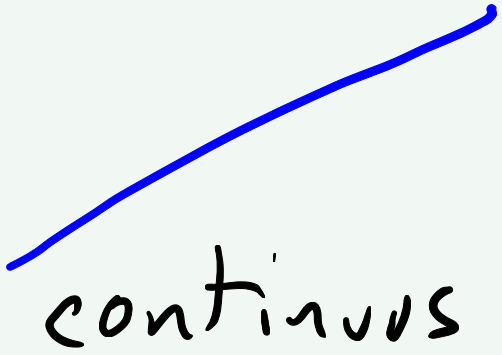
The general problem...

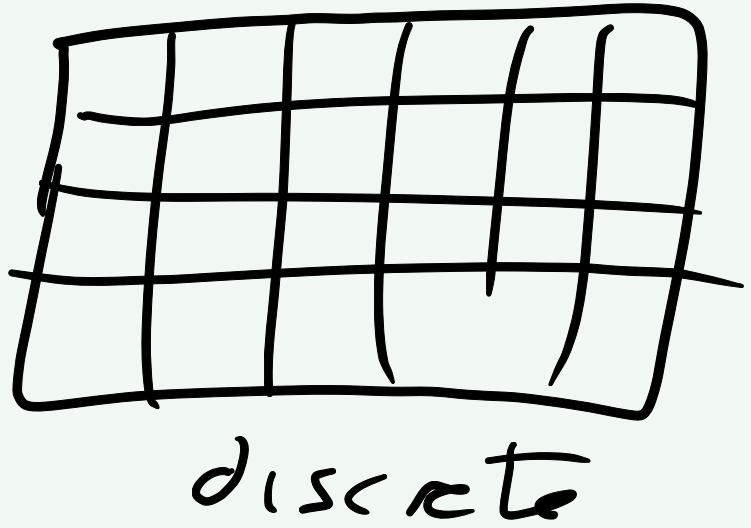
The world/image in continuous

The pixel grid is discrete

This is a fundamental problem in graphics

We'll come back to it

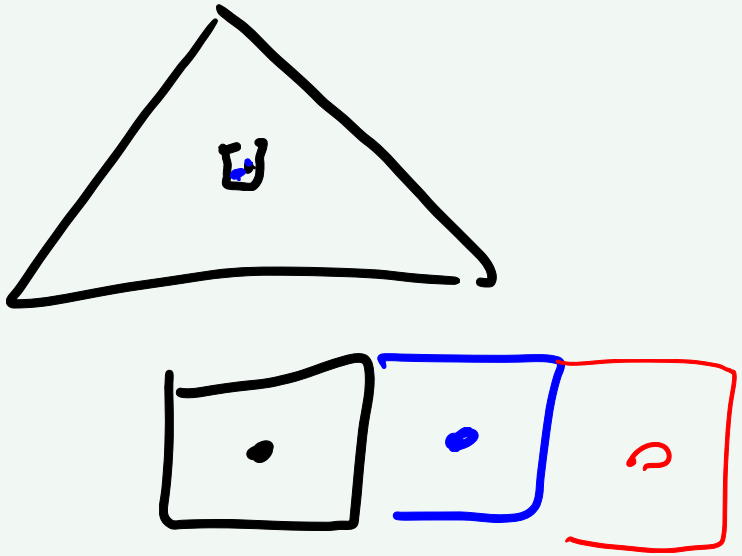


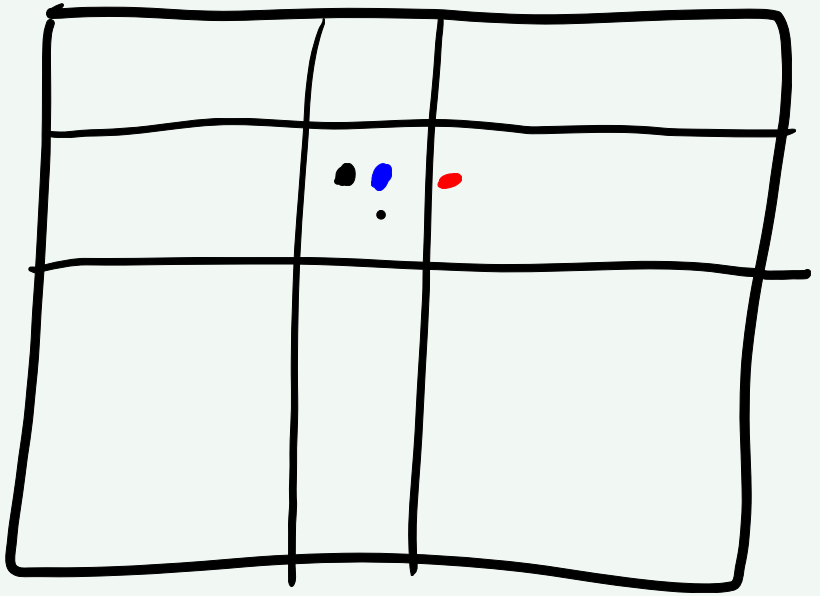


Texture Lookups: Magnification

A pixel covers less than a pixel in the image

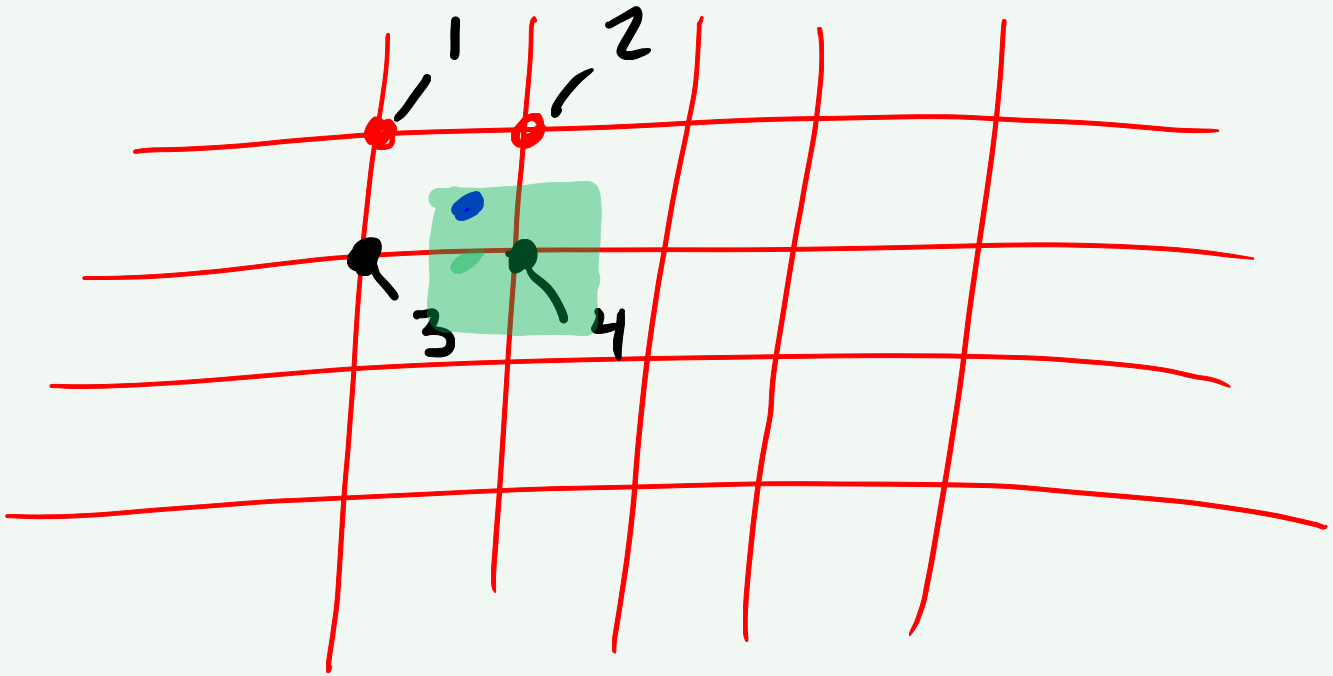
Note: this same reasoning works for "point sampling" (the center of an area)





Basic Solution: Nearest-Neighbor

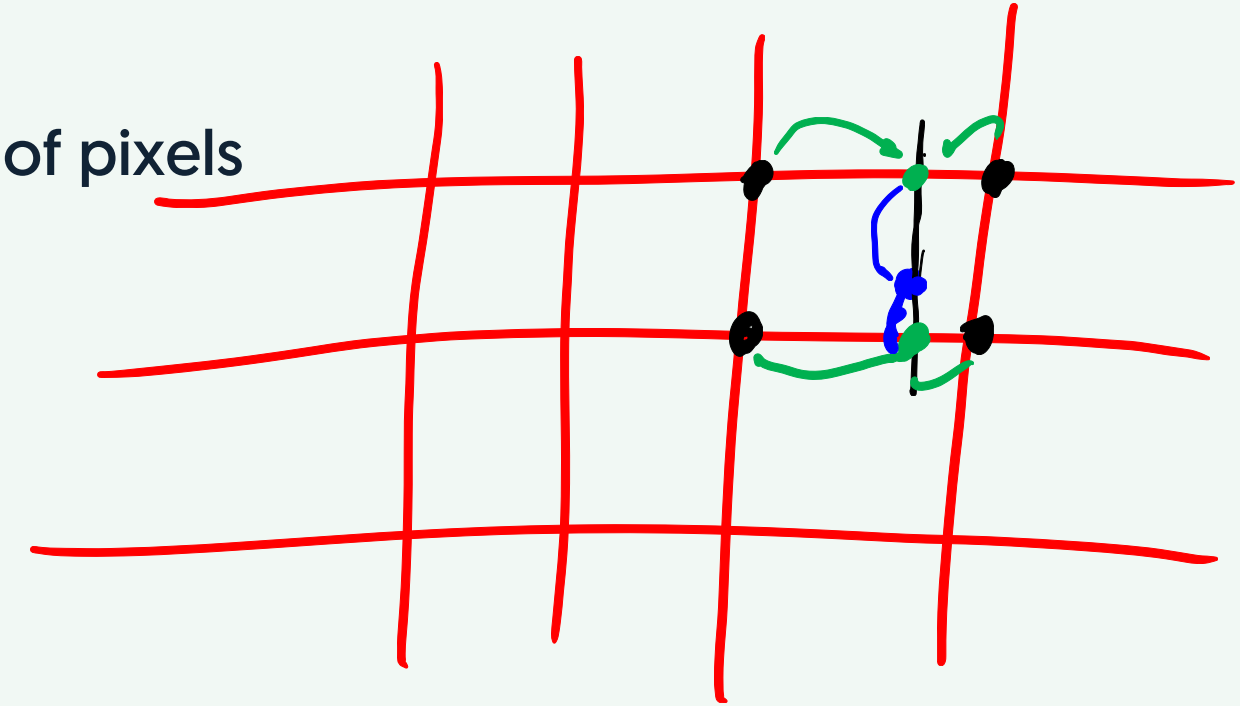
Think of colors at grid corners (points) not filling squares



Standard Solution: Bi-Linear Interpolation

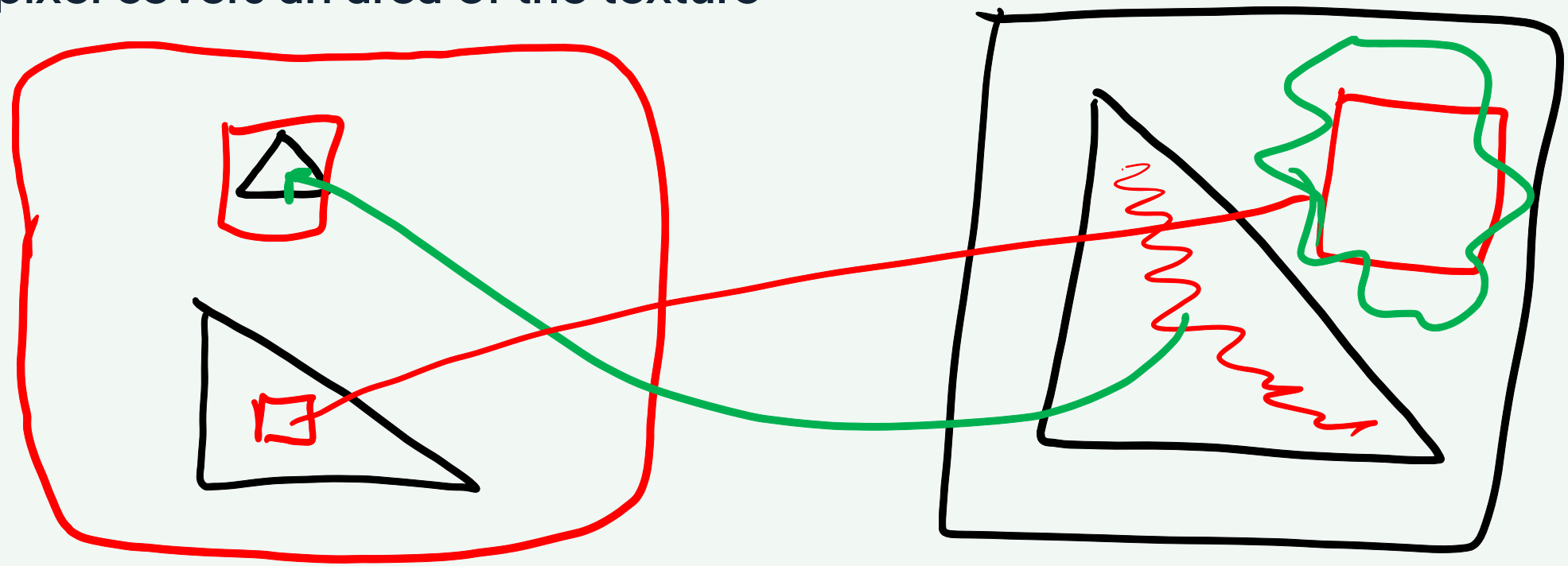
Treat the pixel as a point

Does not consider edges of pixels

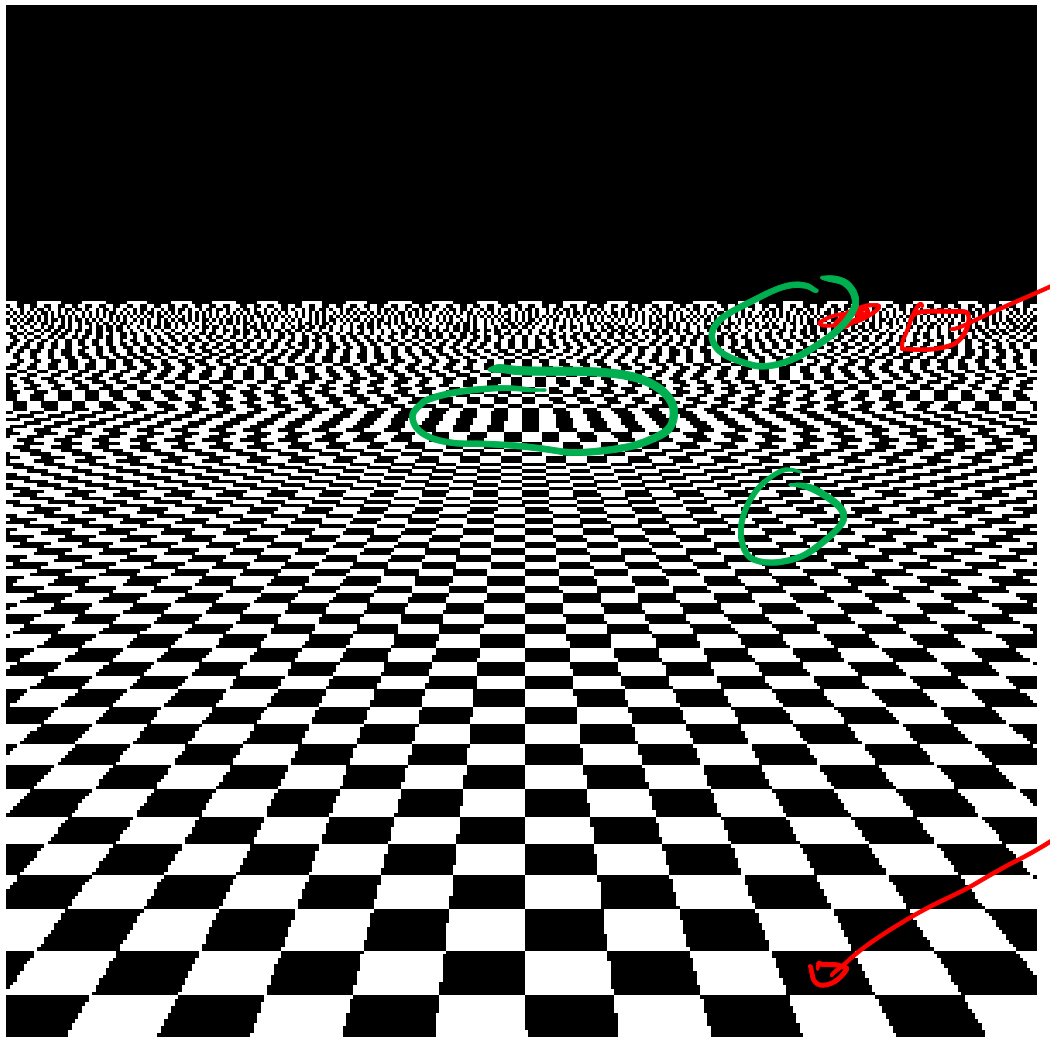


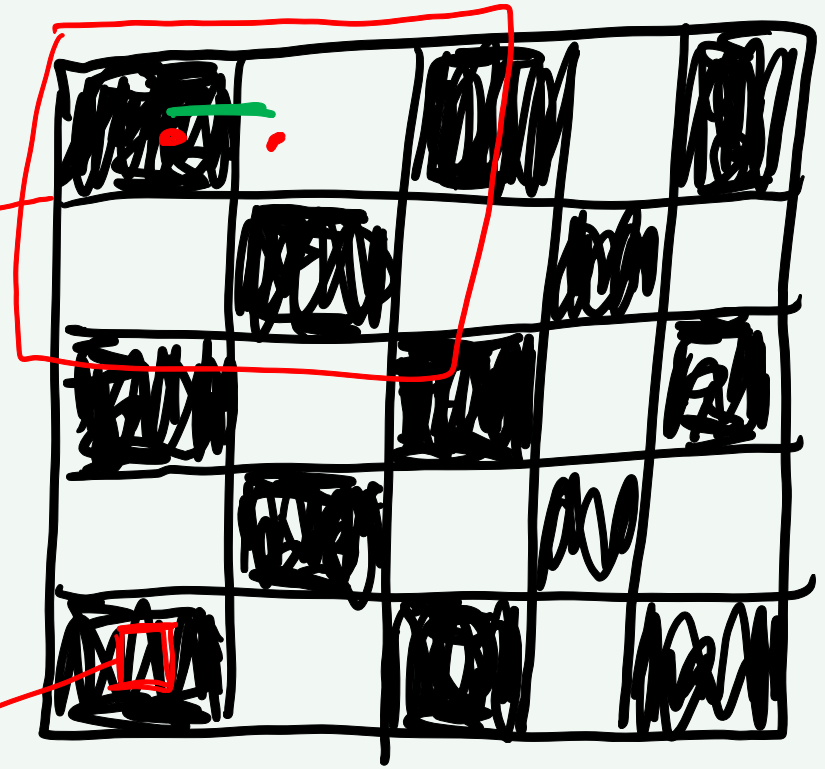
Texture Lookups: Minification

One pixel covers an area of the texture



What if we just look up the color?

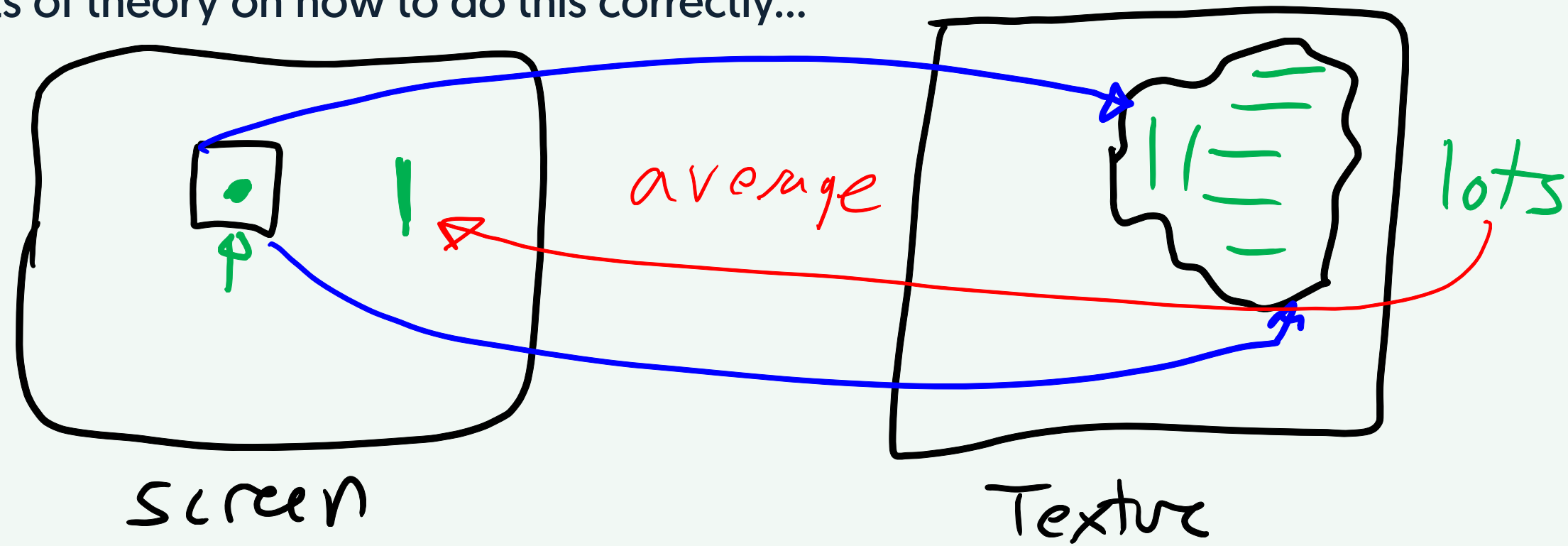




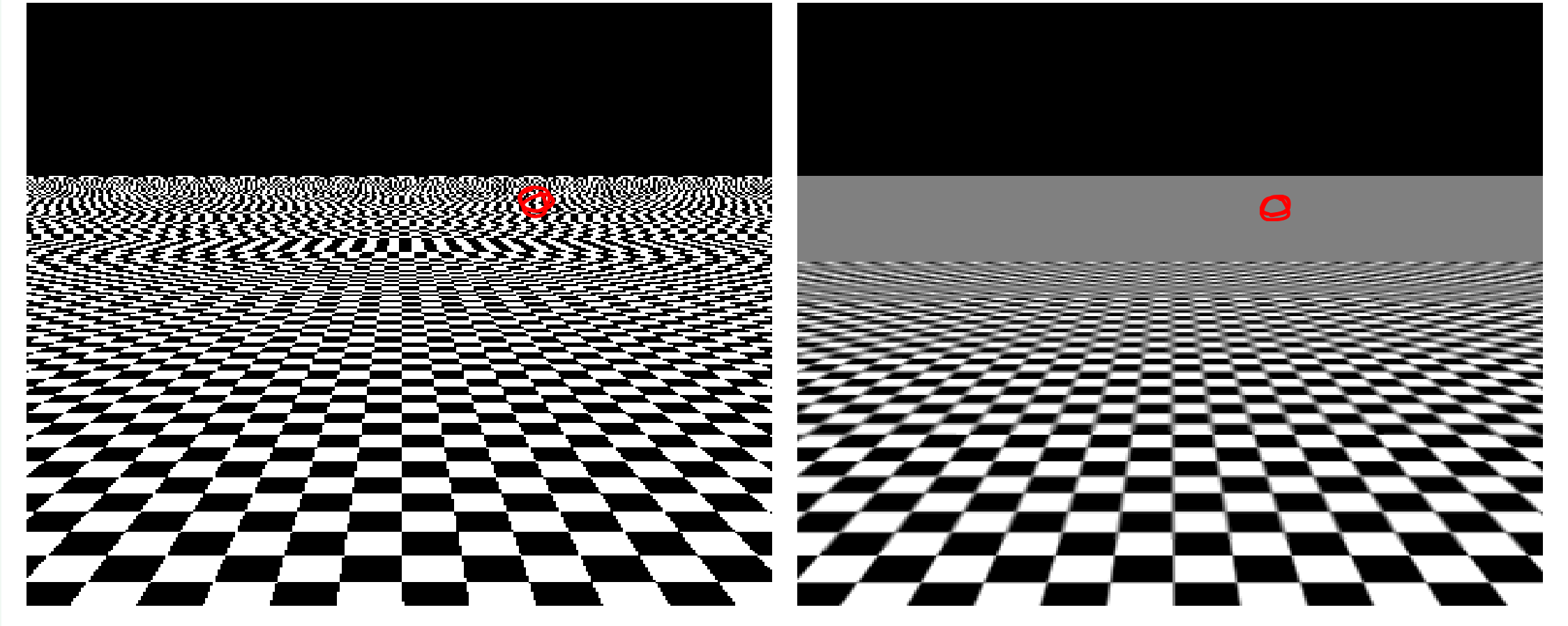
Filtering: The Basic Idea

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...

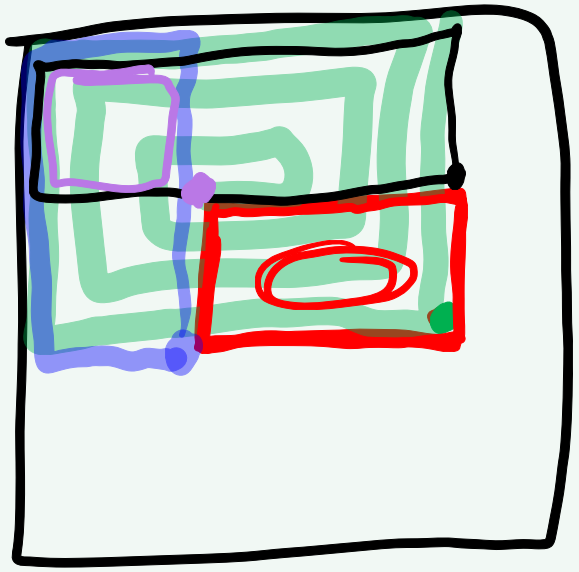
1. Pre-computation
2. Amortization
3. Approximation

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

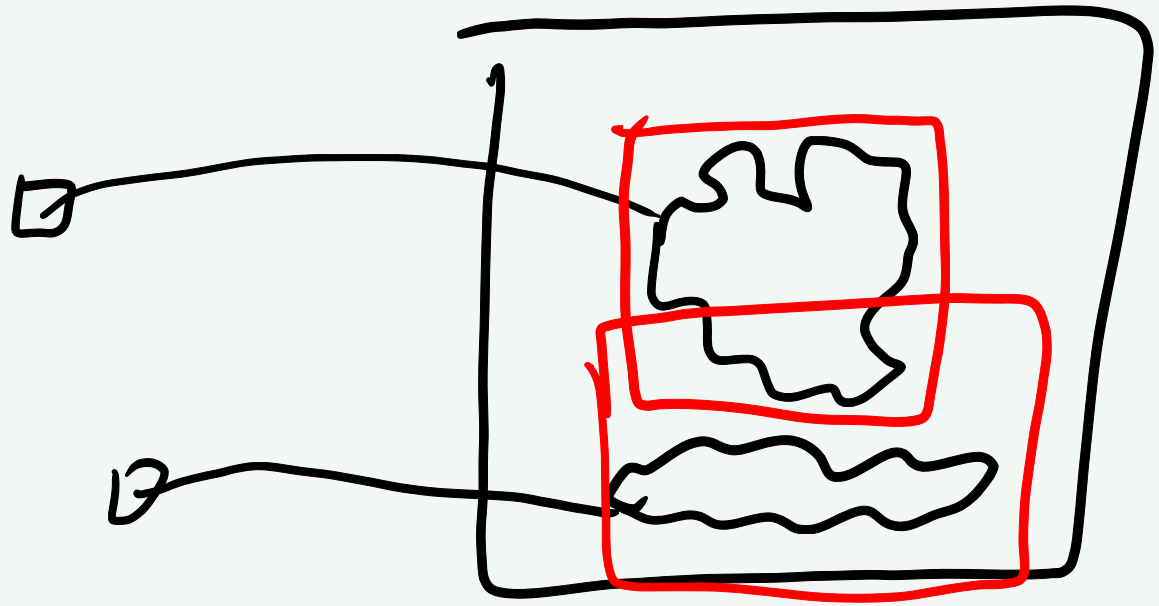
Solution 2: Mip-Maps

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

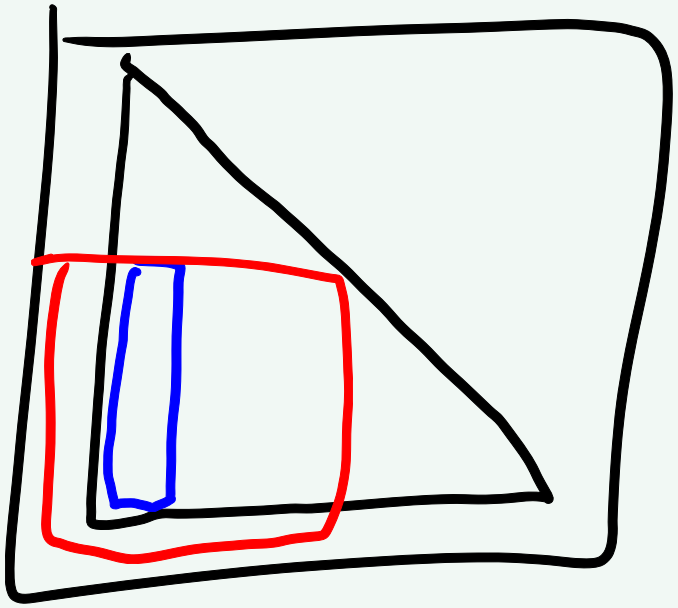
Mip Maps 1: Approximate as square

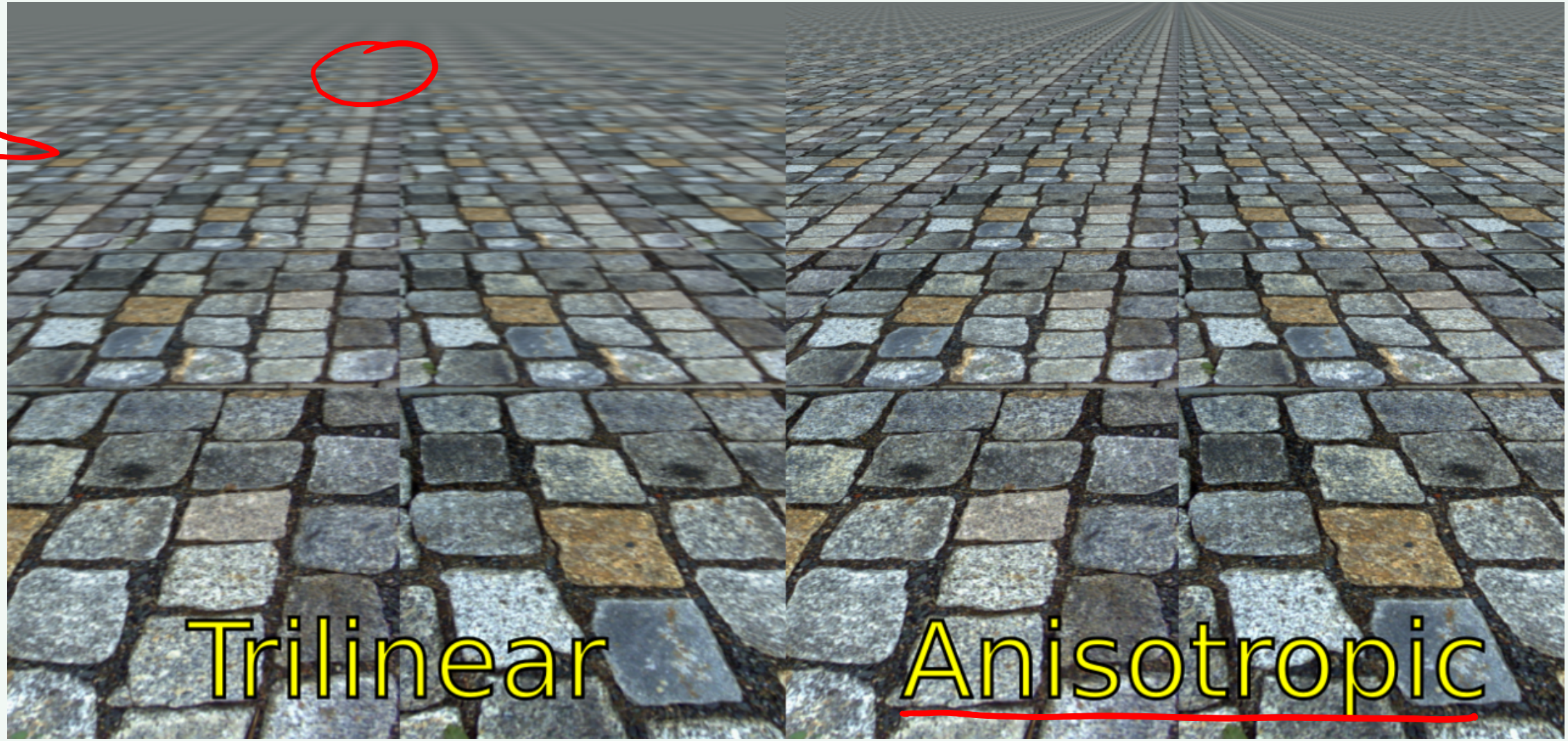


Mip Maps 1: Approximate as square

Problem: Anisotropy

The areas are not square!

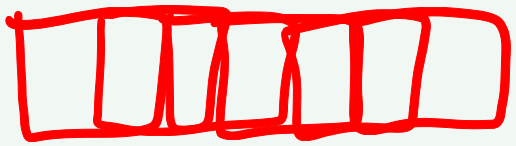


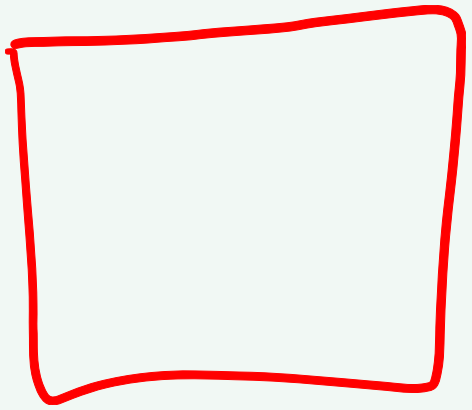

Trilinear
Anisotropic

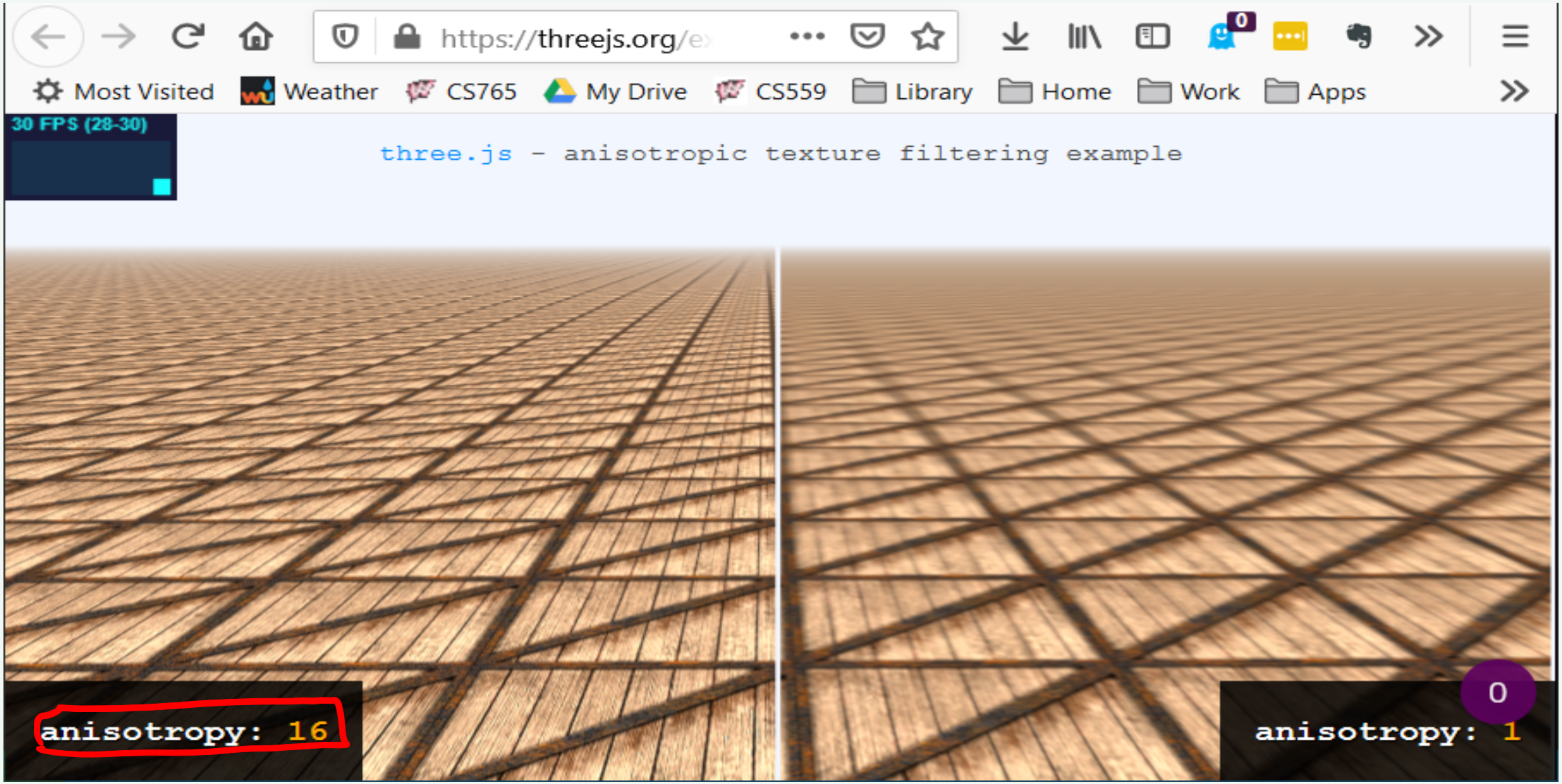
Anisotropic Filtering

Hack version:

Use multiple squares (three supports this)



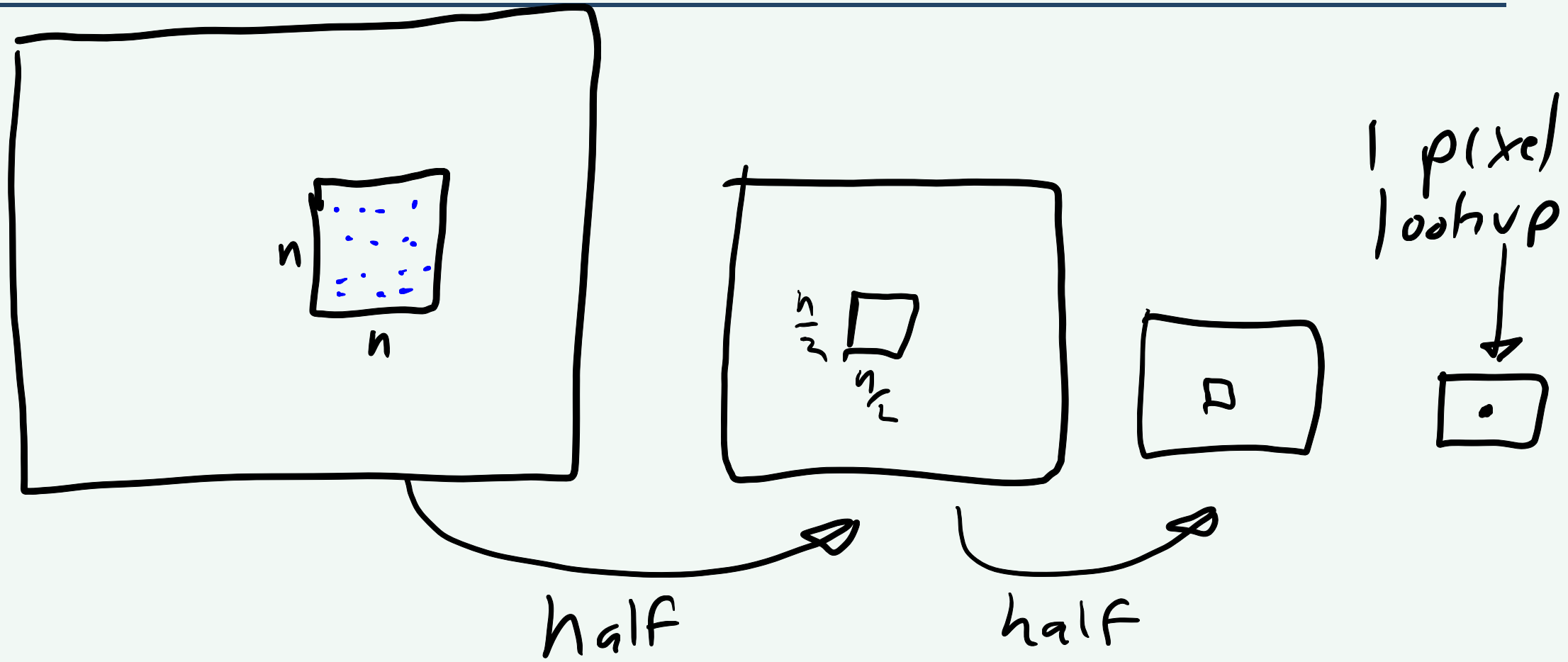




16 squares

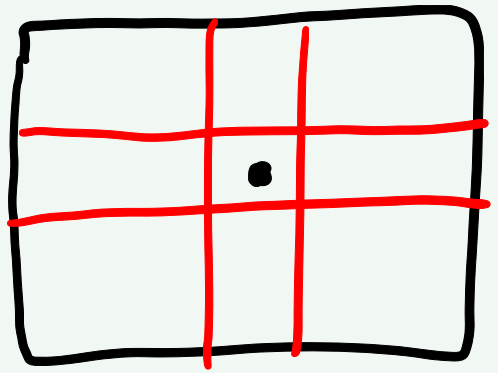
no - anisotropy

Mip Maps 2: Lookup in smaller image



Mip Maps 3: Lookup in smallest image

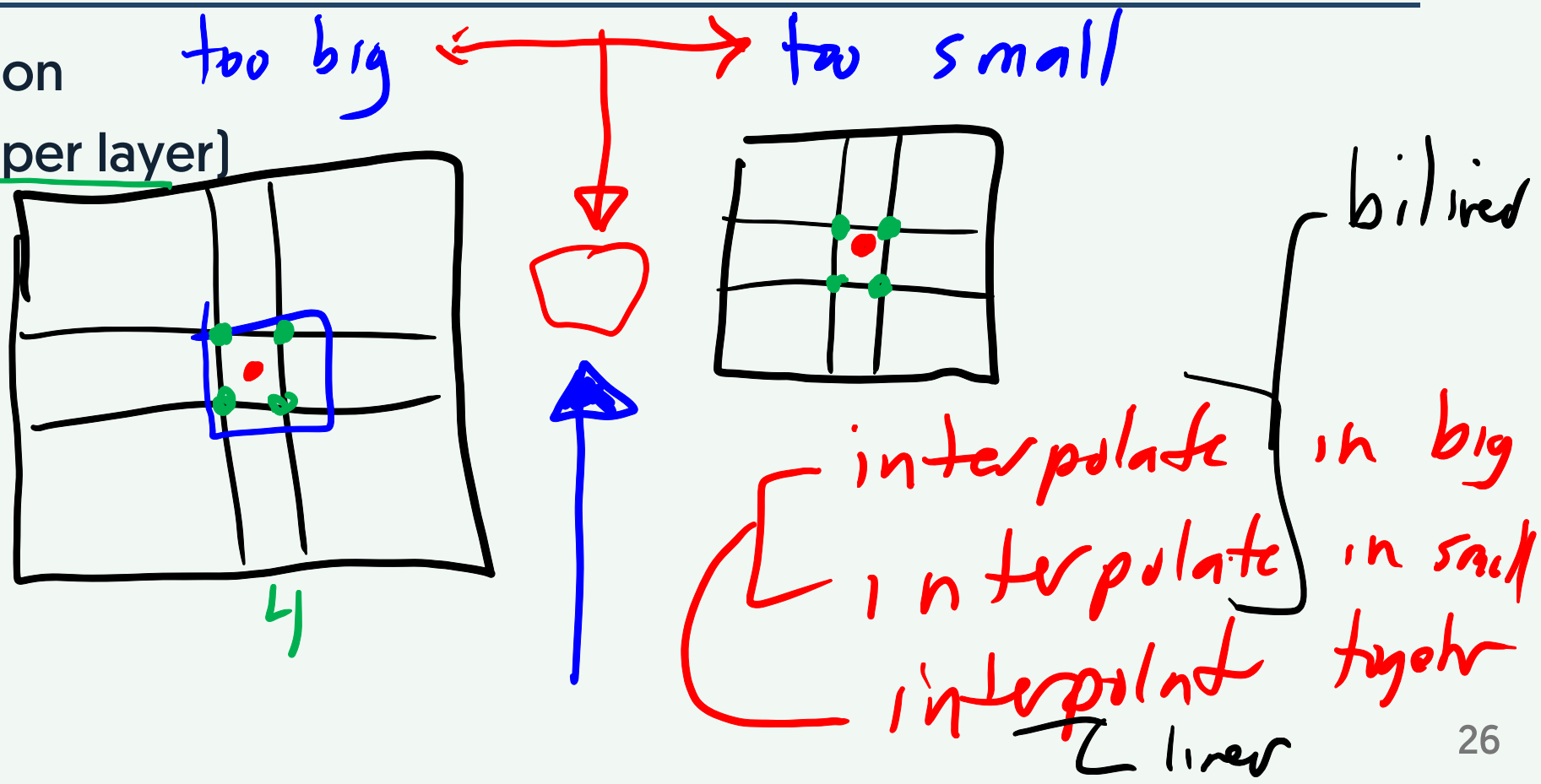
Bi-Linear Interpolation



Mip Maps 4: Lookup in-between images

Tri-Linear Interpolation

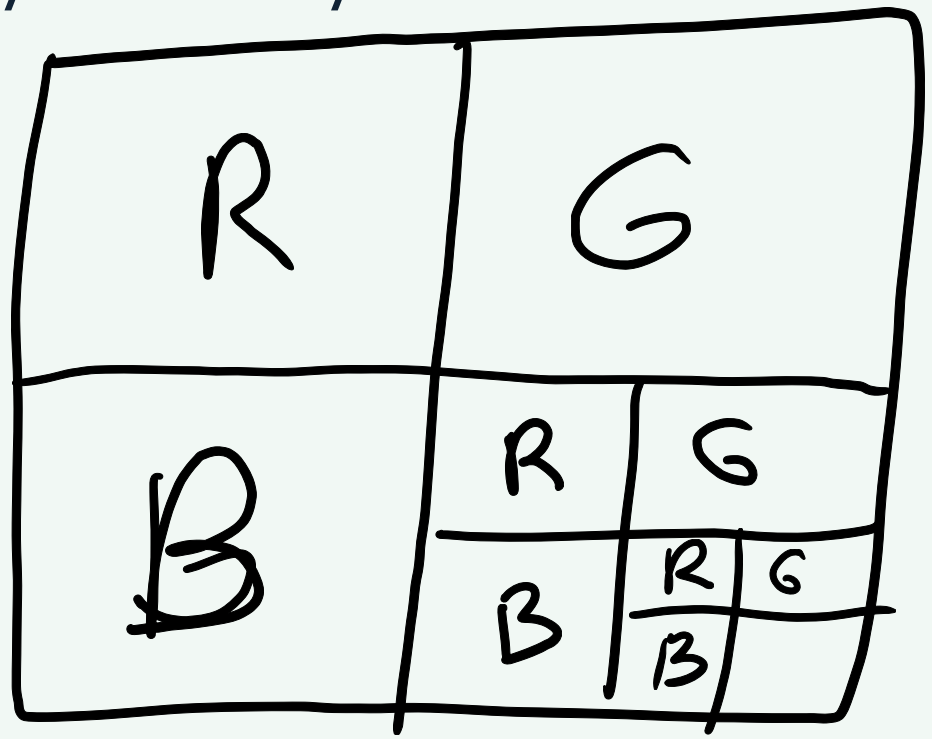
requires 8 values (4 per layer)

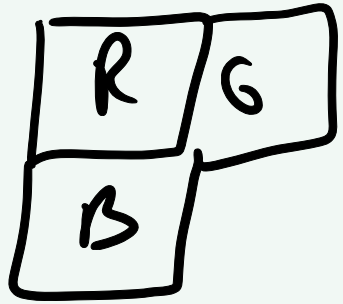


Mip Maps 5: A Historic Storage Trick

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!