Lecture 23 Part C
Anti-Aliasing in Shaders

Two Strategies

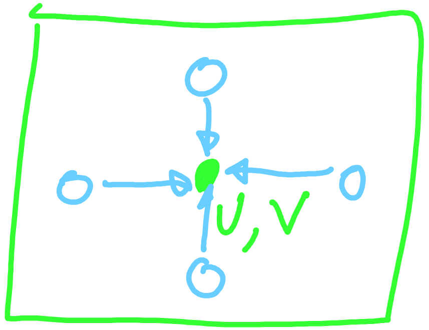
1. Multiple samples
2. Edge Smoothing

Use #1 mainly to introduce a small point

Multi-Sampling (and fwidth intuitions)

One sample per pixel misses things

(we look at one u,v value)



What if we had many samples per pixel?

(and averaged them)

Approximates integrating over the whole region

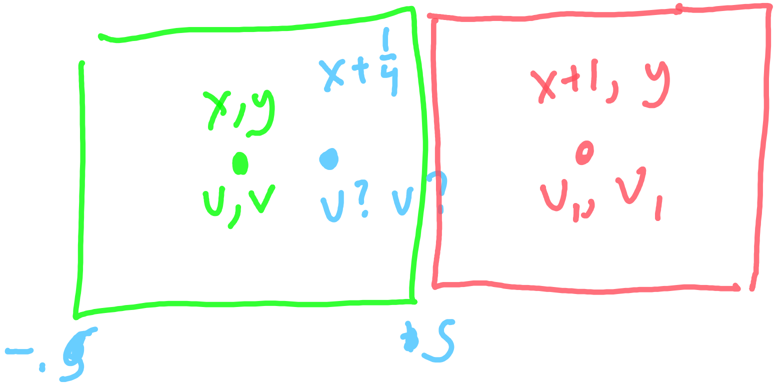
(still point samples - so still aliasing)

Multiple samples per fragment

We know x,y and u,v at the center

We know x,y at the other samples

How do we know what u,v is?



Look at the next pixel?

We can't look at the next pixel...

But GLSL knows what it is going to be

x, y ● u, v $x+1, y$ ● u_1 v_1

du/dx (change in u given change in x)

can do this for dy

can do this for anything computed for each pixel

dFdx dFdy

fwidth

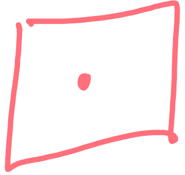
sum of the absolute values of df/dx and df/dy

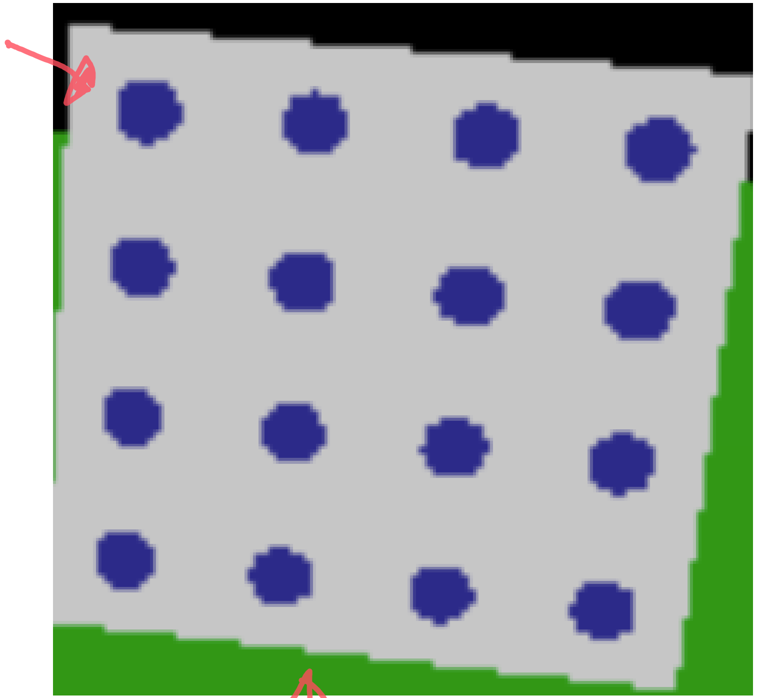
Good approximation of how things change in both directions

tells us how much things change over 1 "diagonal" pixel

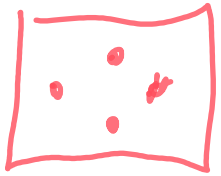
Is this better?

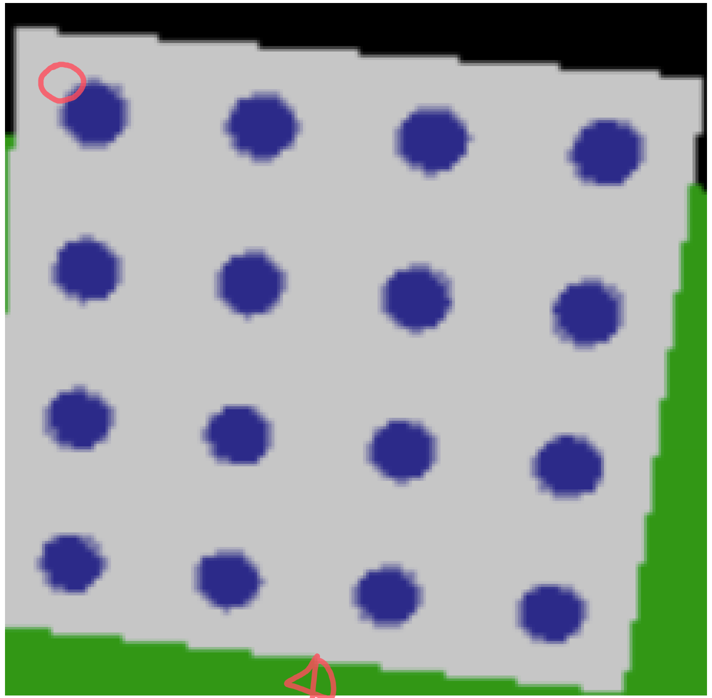
No Blur



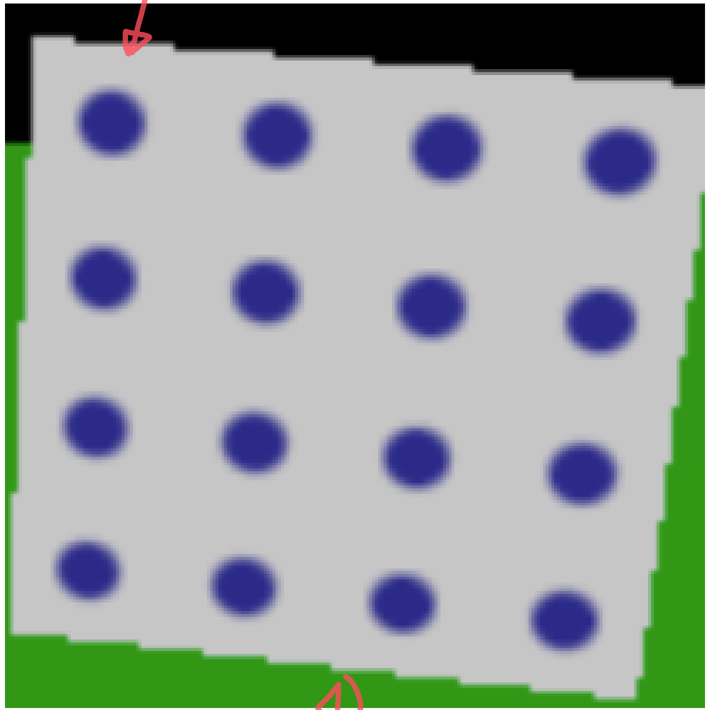


4 samples





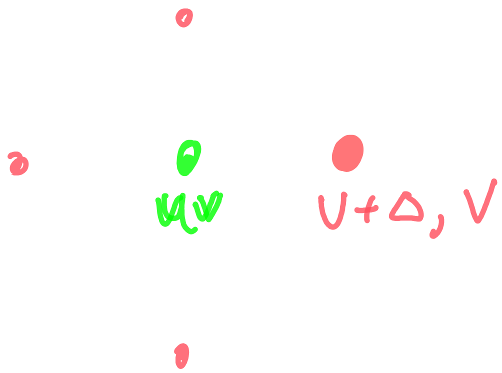
"Correct" (later)



1 Evaluation

```
float u = v_uv.x;                                   // uv at center of pixel
float v = v_uv.y;
float dc = fdot(vec2(u,v),0);
```

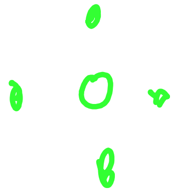
`dc` is used to mix colors



4 Evaluations

```
float u = v_uv.x;                            // uv at center of pixel
float v = v_uv.y;
float dudx = dFdx(u) / 4.0;                  // u step towards x and y
float dudy = dFdx(u) / 4.0;                  // 1/4 of a pixel from center
float dvdx = dFdy(v) / 4.0;                  // v step towards x and y
float dvdy = dFdy(v) / 4.0;                  // 1/2 pixel between samples
float dc1 = fdot(vec2(u+dudx,v+dvdx),0.0);   // sample to the "right"
float dc2 = fdot(vec2(u-dudx,v-dvdx),0.0);   // sample to the "left"
float dc3 = fdot(vec2(u+dudy,v+dvdy),0.0);   // sample "above" (+y)
float dc4 = fdot(vec2(u-dudy,v-dvdy),0.0);   // sample below (-y)
float dc = (dc1+dc2+dc3+dc4)/4.0;            // average
```

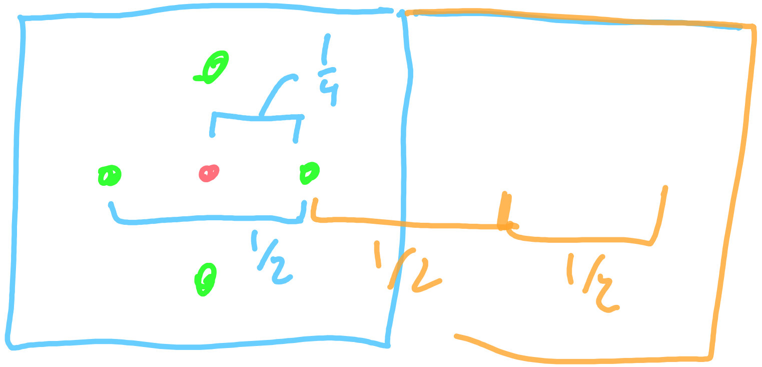
$\vec{dx}$



Some notes

1. I blended the "amount of dot" - not the color
2. 4 points in the "diamond pattern" (arbitrary)
   - 5th point (center)
   - other patterns
3. location of dots (+/- 1/4) is arbitrary

4. we'll see the dot function in a minute



Multi-Sampling

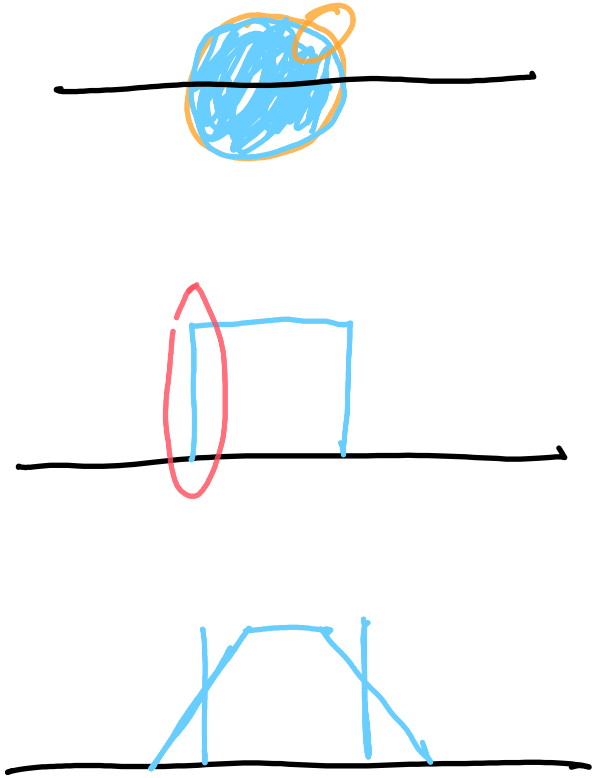
Good:

1. Can be done by the hardware
2. Easy to implement (multiple evals, average)

Bad:

1. Limited (still small number of points)
2. Inefficient (needs multiple points)

Stategy 2: Filter those edges!

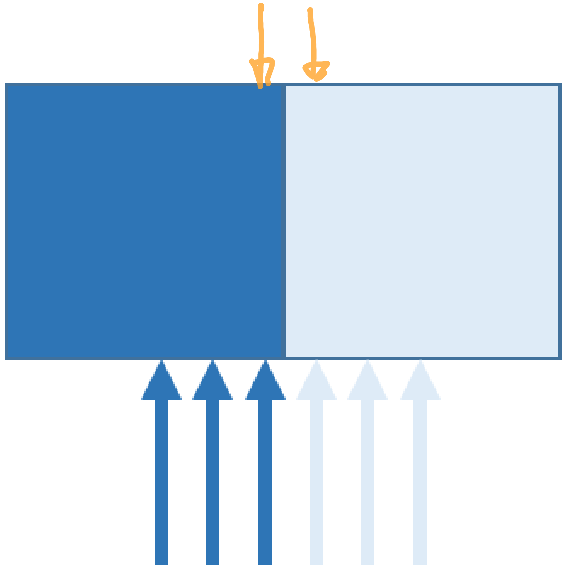


Intuition: Sharp Edges are bad

Sharp Edge:

Small change in position

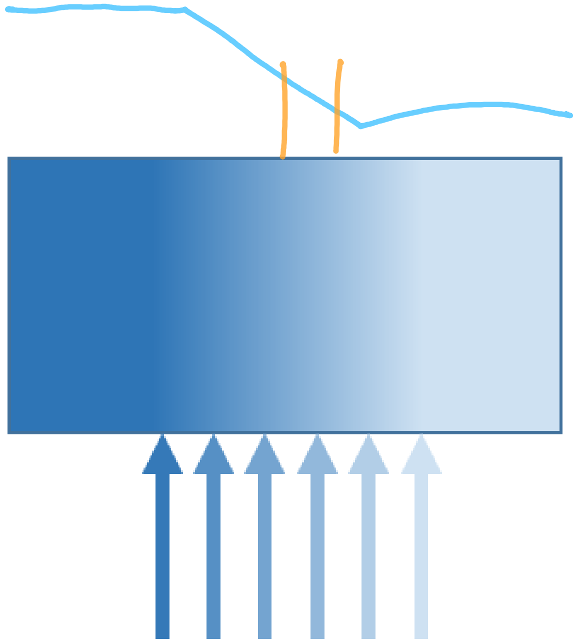
Big change in value



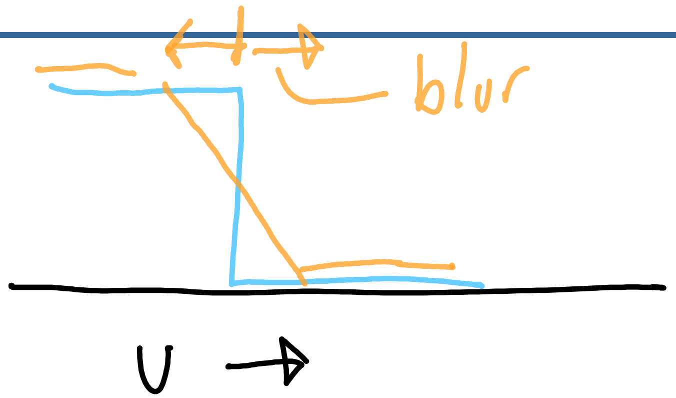
Smoother "Edge:"

Small change in position

Doesn't matter (that much)



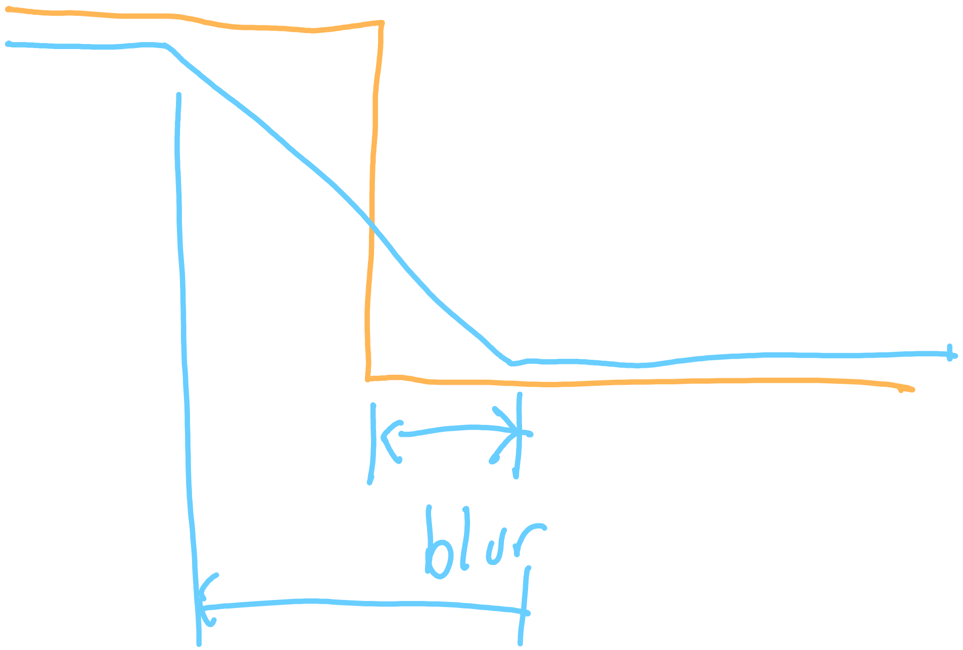
Where do sharp edges come from?



step

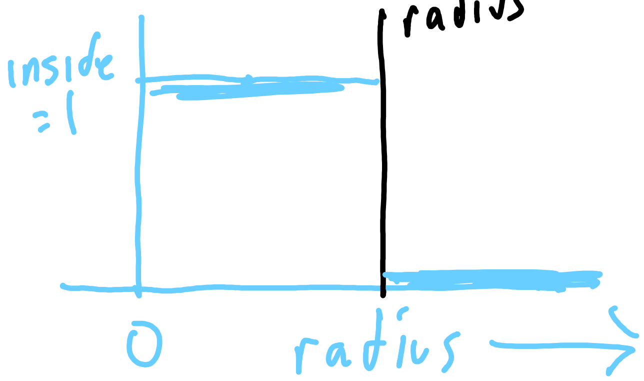
smoothstep

A Blurry Edge



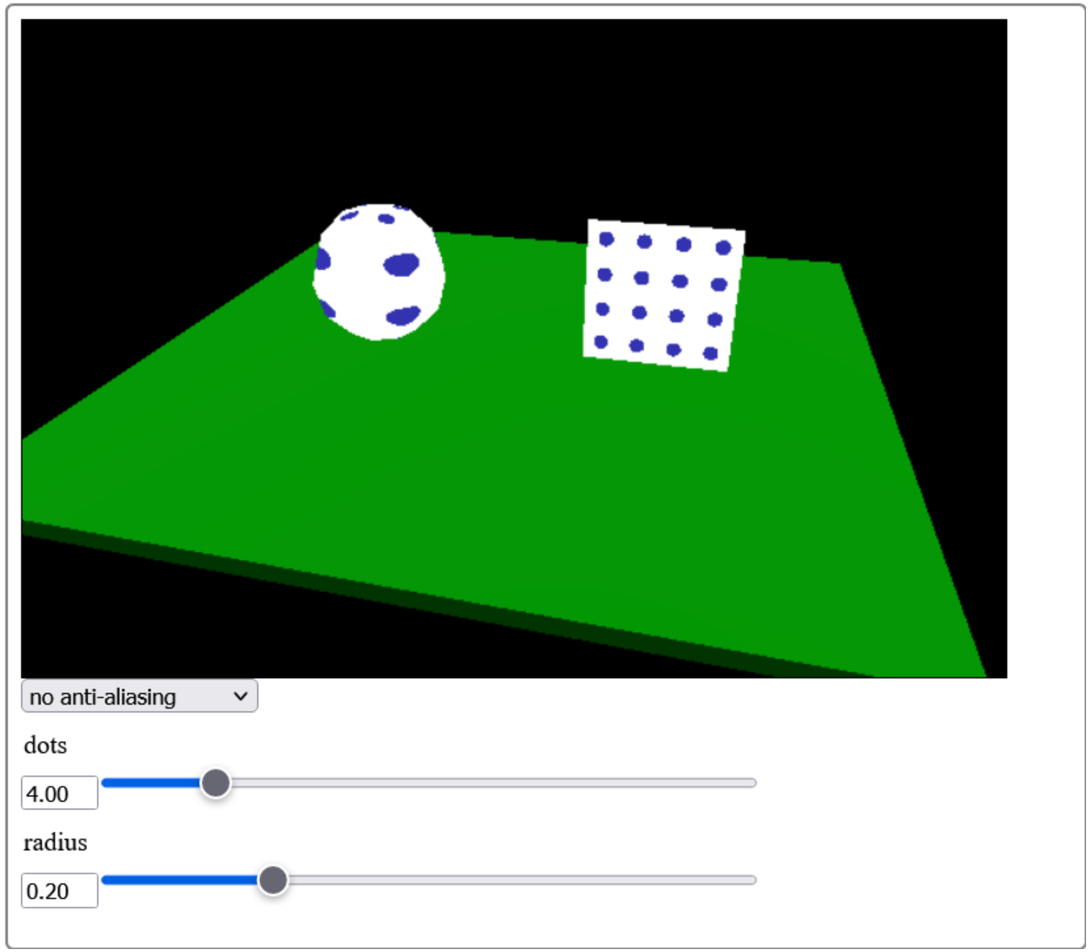
Dots

Note we `step` from 0 to 1 at the edge of a dot



```
float fdot(vec2 uv) {
    float x = uv.x * dots;
    float y = uv.y * dots;

    float xc = floor(x);
    float yc = floor(y);

    float dx = x-xc-.5;
    float dy = y-yc-.5;

    float d = sqrt(dx*dx + dy*dy);

    dc = 1.0-step(radius,d);

    return dc;
}
```

Dots



no anti-aliasing

dots

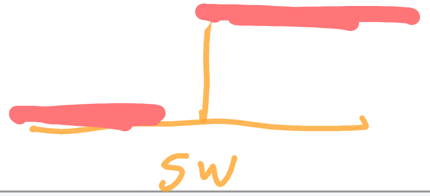
4.00

radius

0.20

```
float fdot(vec2 uv) {
    float x = uv.x * dots;
    float y = uv.y * dots;

    float xc = floor(x);
    float yc = floor(y);

    float dx = x-xc-.5;
    float dy = y-yc-.5;

    float d = sqrt(dx*dx + dy*dy);

    dc = 1.0-step(radius,d);

    return dc;
}
```

Step vs. SmoothStep

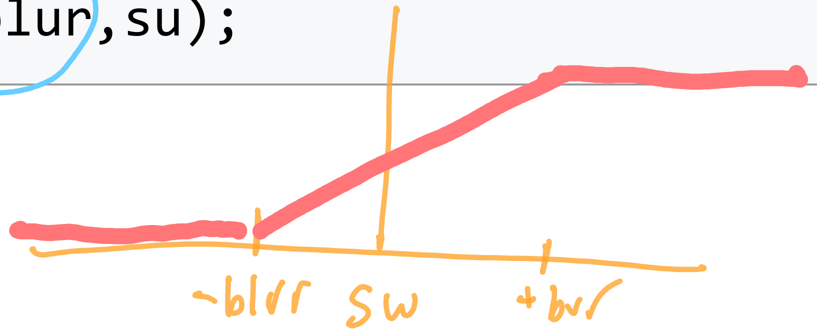


step

```
float st = step(sw,su);
```

smoothstep

```
uniform float blur;
float st = smoothstep(sw-blur,sw+blur,su);
```



Dots

`smoothstep` - but we needed blur!

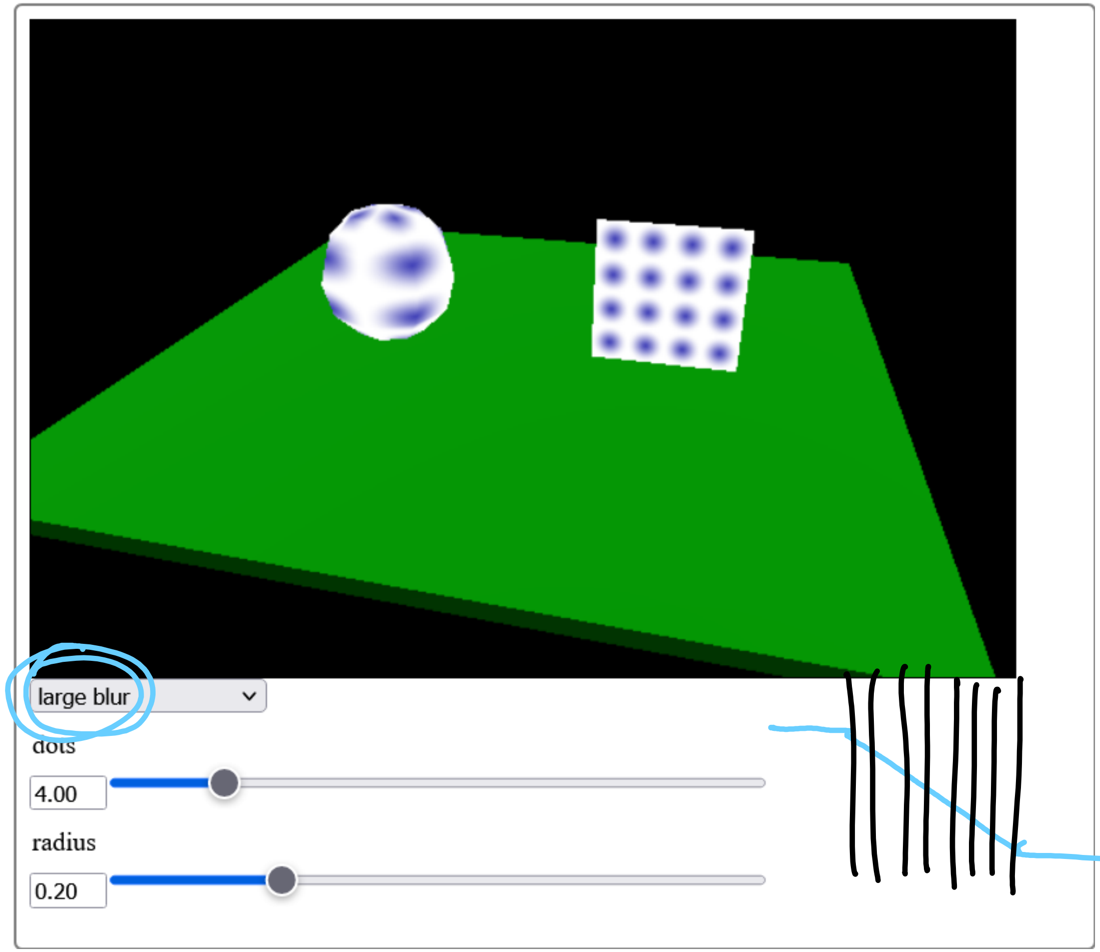
radius `r`

blur `b`

"range" of smooth step `r-b` to `r+b`

```
float fdot(vec2 uv) {
    float x = uv.x * dots;
    float y = uv.y * dots;

    float xc = floor(x);
    float yc = floor(y);

    float dx = x-xc-.5;
    float dy = y-yc-.5;

    float d = sqrt(dx*dx + dy*dy);

    dc = 1.0-smoothstep(r-b,r+b,d);

    return dc;
}
```

Blurry Dots



large blur

dots

4.00

radius

0.20

```
float fdot(vec2 uv) {
     float x = uv.x * dots;
     float y = uv.y * dots;

     float xc = floor(x);
     float yc = floor(y);

     float dx = x-xc-.5;
     float dy = y-yc-.5;

     float d = sqrt(dx*dx + dy*dy);

     dc = 1.0-smoothstep(r-b,r+b,d);

     return dc;
}
```

Too blurry?

Good news: no aliasing!

Need to balance:

- blurry enough (to avoid aliasing)
- not too blurry (to avoid looking bad)

How to pick the amount of blur?

The blurring challenge...

The units of the dot size is "u,v" coordinates

The amount of blurring we need is "about 1 pixel"

How much UV is 1 pixel?

What value for the width?

Want the blur to be "one pixel wide"

But what is that in u values?

GLSL will figure it out for us!

- `dFdx` - derivative of function with respect to x
- `fwidth` - derivative of function with respect to x and y

These are **extensions** to GLSL-ES (but three loads them for us)

- we need to enable them in the shader

Dots - with fwidth

`smoothstep` - but we needed blur!

radius `r`

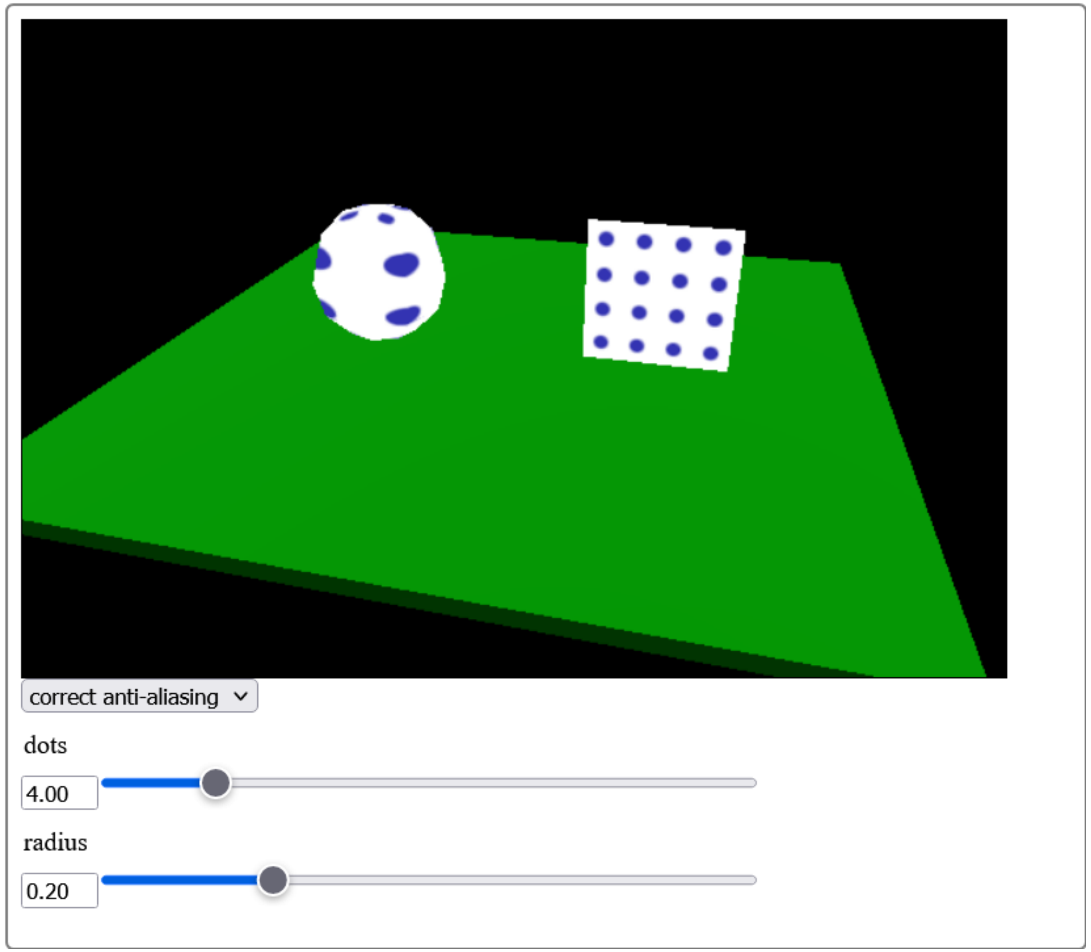
blur `b` = `fwidth(d)`

how much d changes over 1 pixel

"range" of smooth step `r-b` to `r+b`

```
float fdot(vec2 uv) {
    float x = uv.x * dots;
    float y = uv.y * dots;

    float xc = floor(x);
    float yc = floor(y);

    float dx = x-xc-.5;
    float dy = y-yc-.5;

    float d = sqrt(dx*dx + dy*dy);
    float b = fwidth(d);

    dc = 1.0-smoothstep(r-b,r+b,d);

    return dc;
}
```

Correctly Anti-Aliased Dots



correct anti-aliasing

dots

4.00

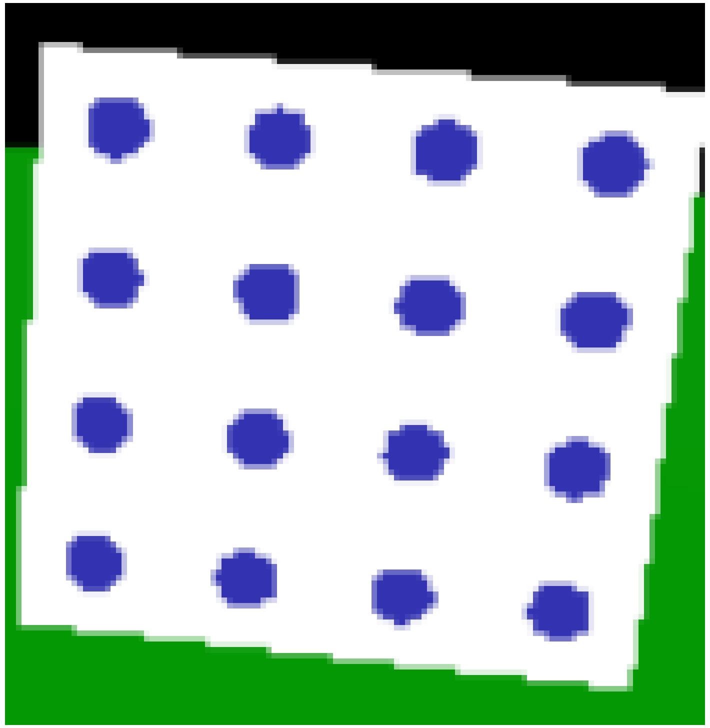
radius

0.20

```
float fdot(vec2 uv) {
    float x = uv.x * dots;
    float y = uv.y * dots;

    float xc = floor(x);
    float yc = floor(y);

    float dx = x-xc-.5;
    float dy = y-yc-.5;

    float d = sqrt(dx*dx + dy*dy);

    dc = 1.0-smoothstep(r-b,r+b,d);

    return dc;
}
```

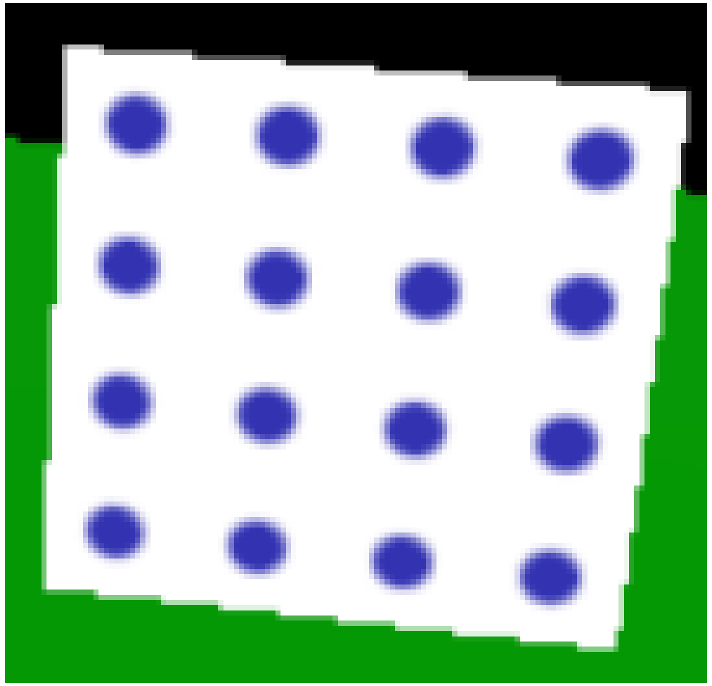
Is this better?

No Blur

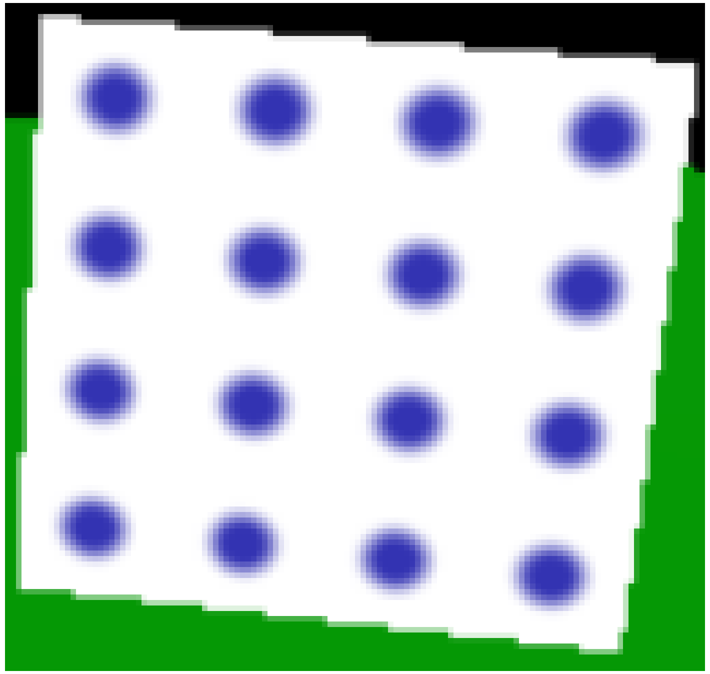


no anti-alias'g

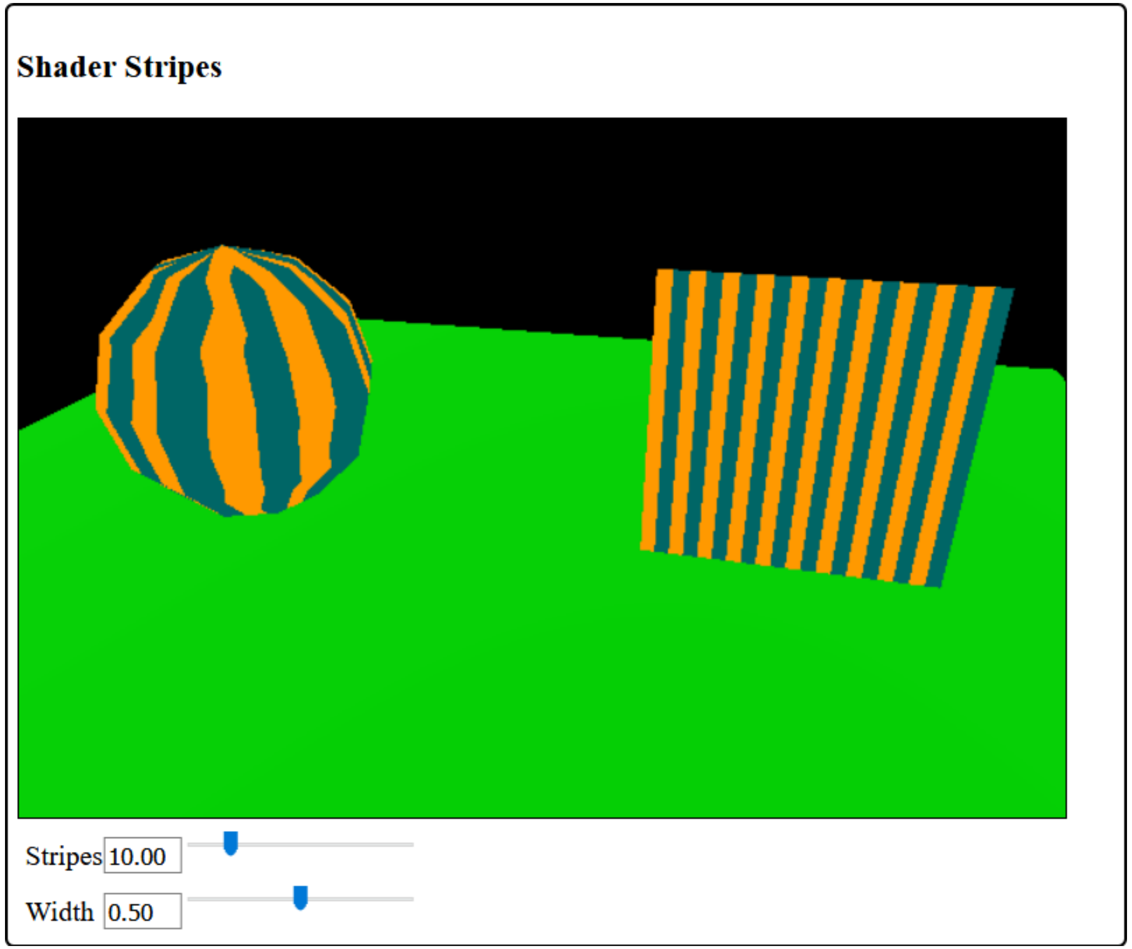
"Correct" Blur



Too Much Blur



Stripes

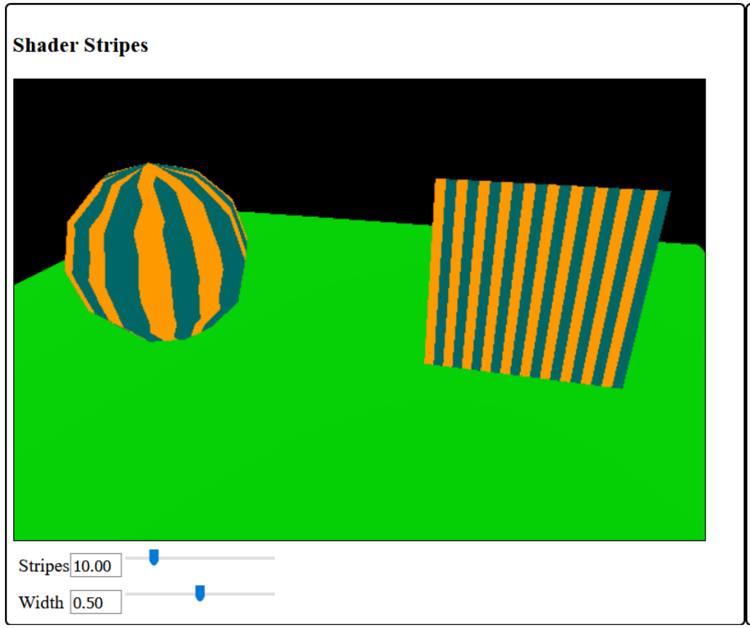


Shader Stripes

Stripes 10.00

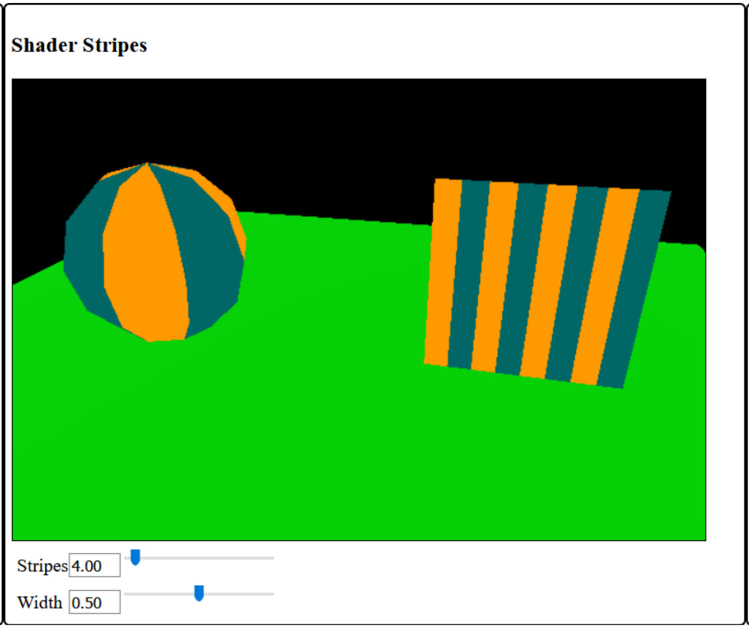
Width 0.50

Parameters - change as needed


Shader Stripes

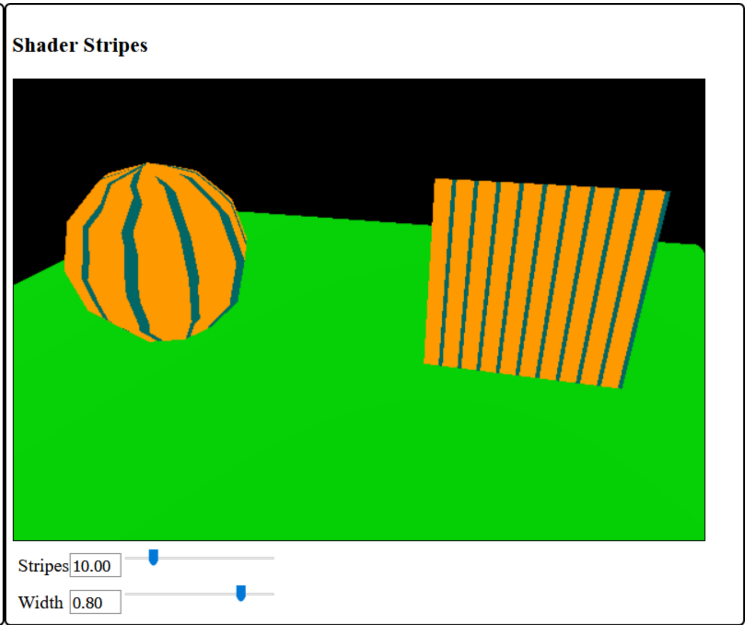
Stripes 10.00

Width 0.50


Shader Stripes

Stripes 4.00

Width 0.50


Shader Stripes

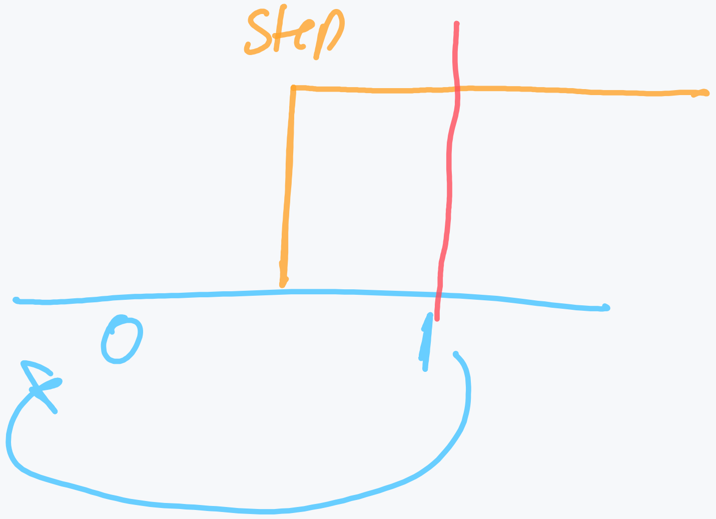
Stripes 10.00

Width 0.80

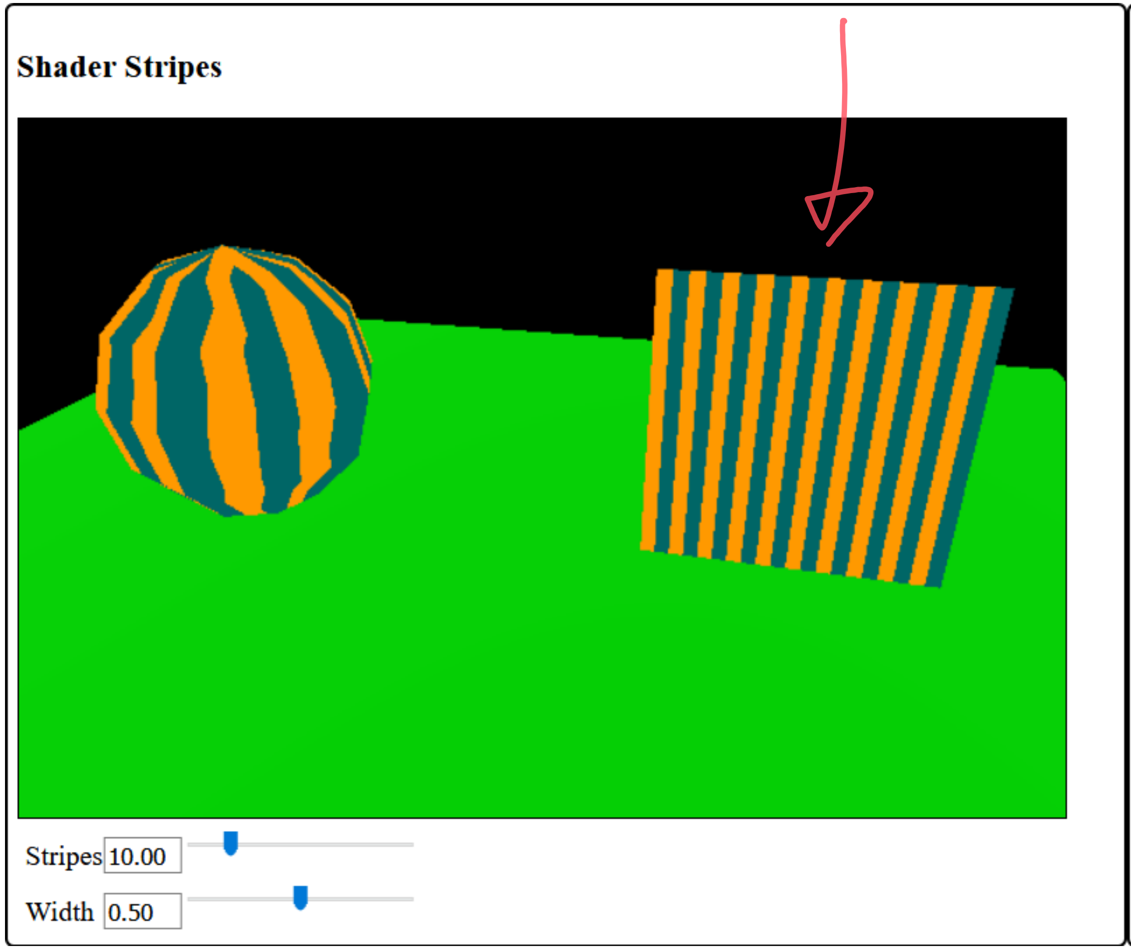
Parameters for the number of stripes and the width

Or change other things (colors)...

```
varying vec2 v_uv;

uniform vec3 color1;
uniform vec3 color2;
uniform float sw;
uniform float stripes;

void main()
{
    // broken into multiple lines to be easier to read
    float su = fract(v_uv.x * stripes);
    float st = step(sw,su);
    vec3 color = mix(color1, color2, st);
    gl_FragColor = vec4(color,1);
}
```



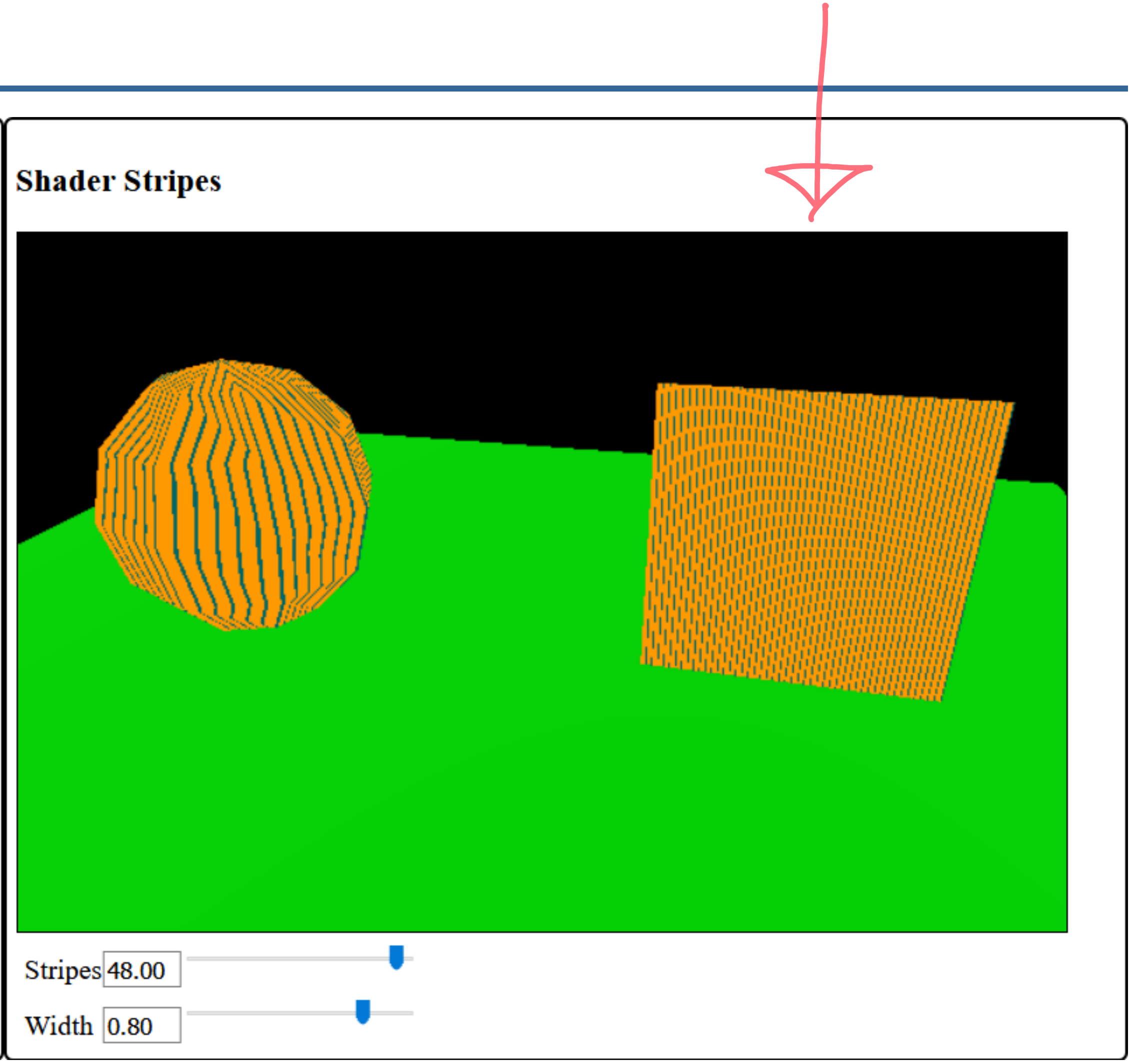
Note the jaggies



Shader Stripes

Stripes 10.00

Width 0.50



Shader Stripes

Stripes 48.00

Width 0.80

Warning - this is one sided!

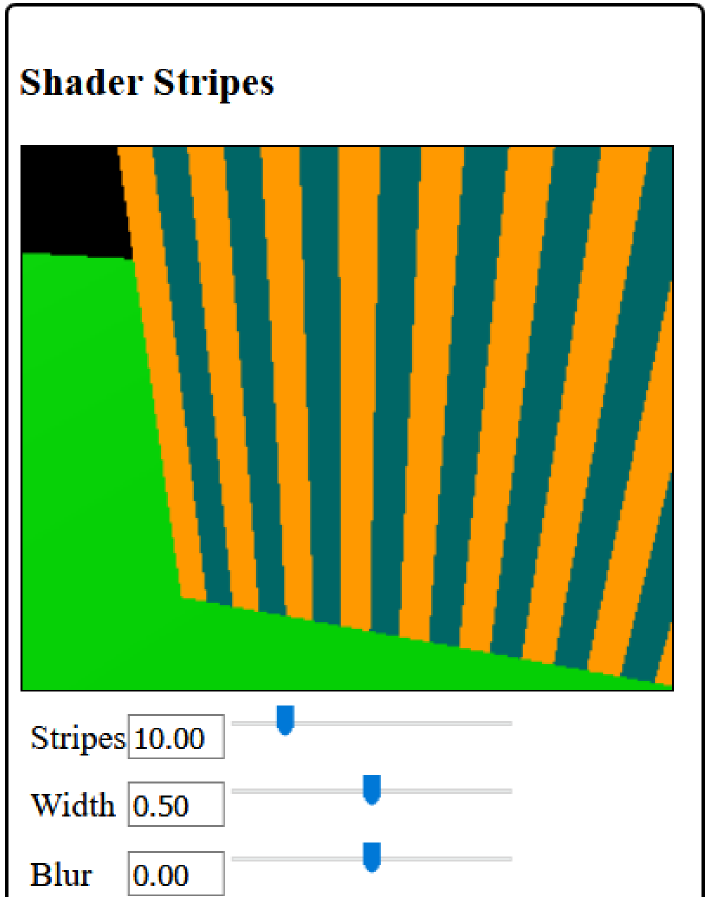
Step is only for the 0-1 transition

The 1-0 transition happens at the repeat

Need to deal with it separately

Does this help?

Width 0 (no blur)



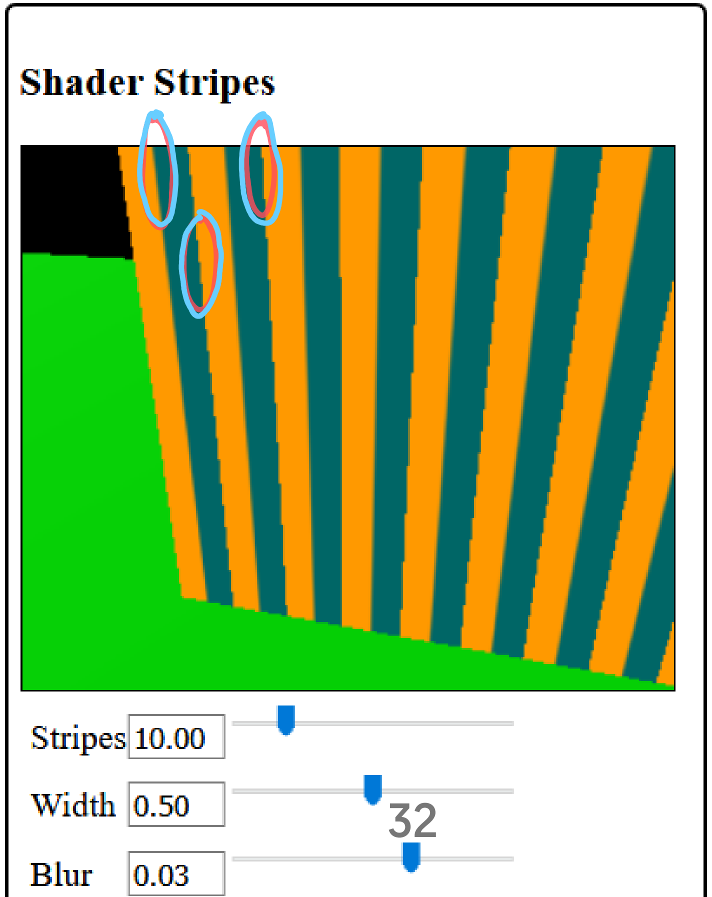
Shader Stripes

Stripes 10.00

Width 0.50

Blur 0.00

Width 3 (blur)



Shader Stripes

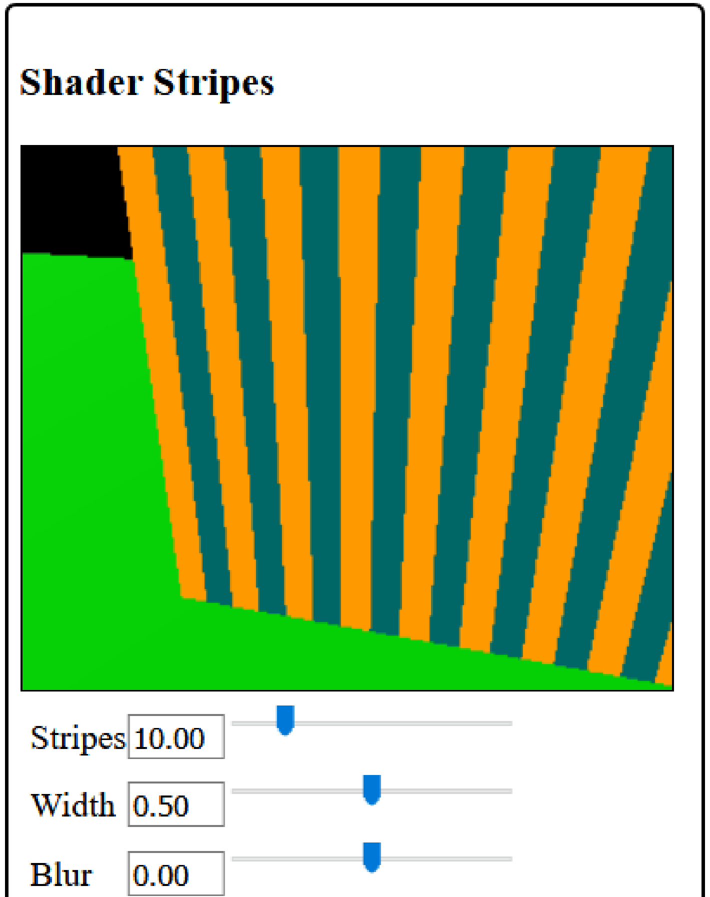
Stripes 10.00

Width 0.50

Blur 0.03

Too much of a good thing?

Width 0 (no blur)



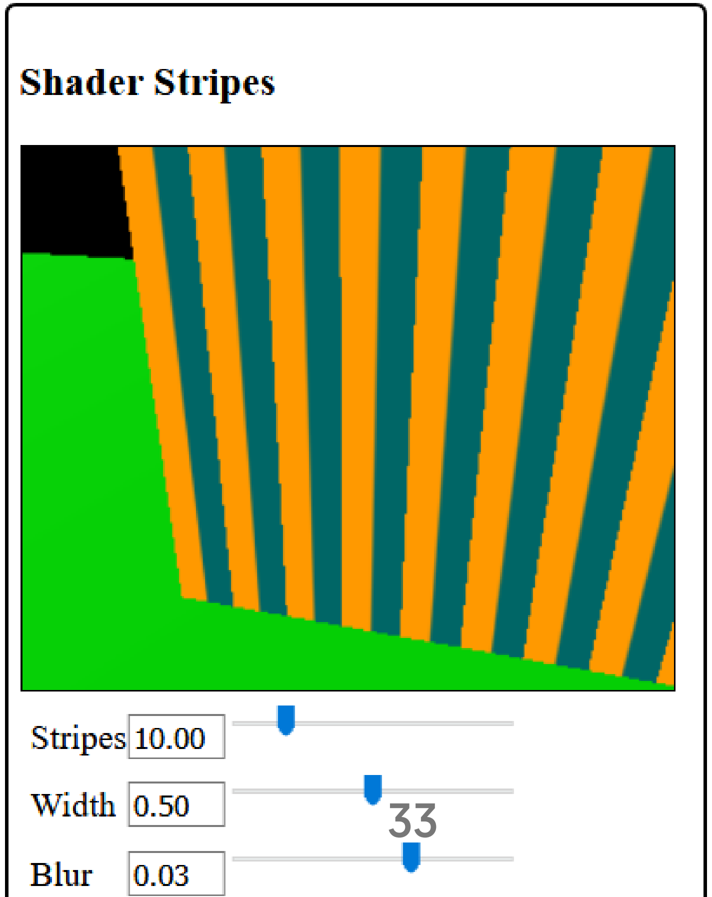
Shader Stripes

Stripes 10.00

Width 0.50

Blur 0.00

Width 3 (blur)



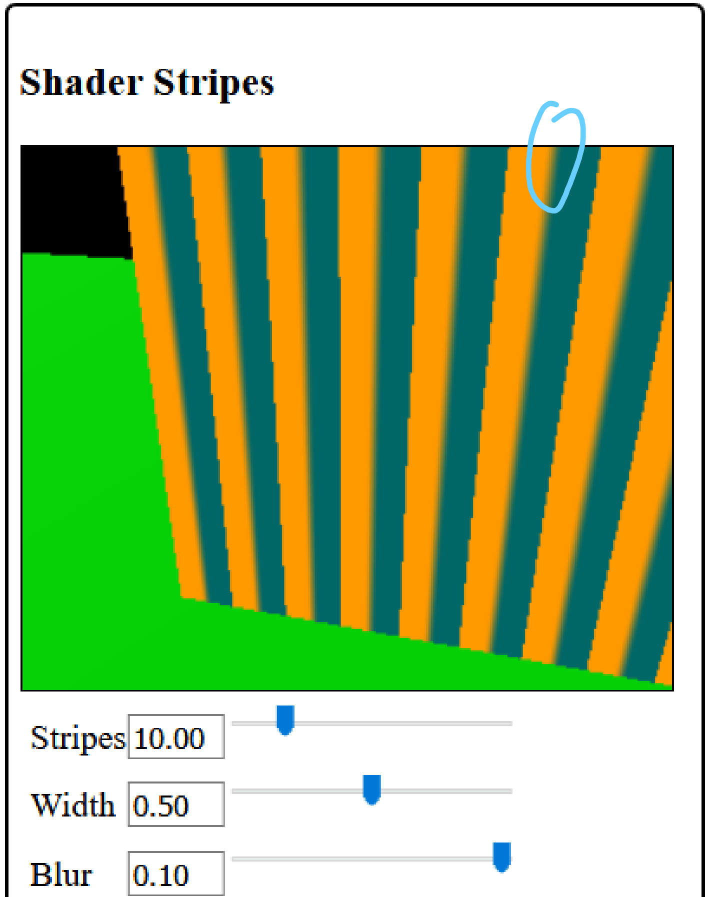
Shader Stripes

Stripes 10.00

Width 0.50

Blur 0.03

Width 10 (blurry)



Shader Stripes

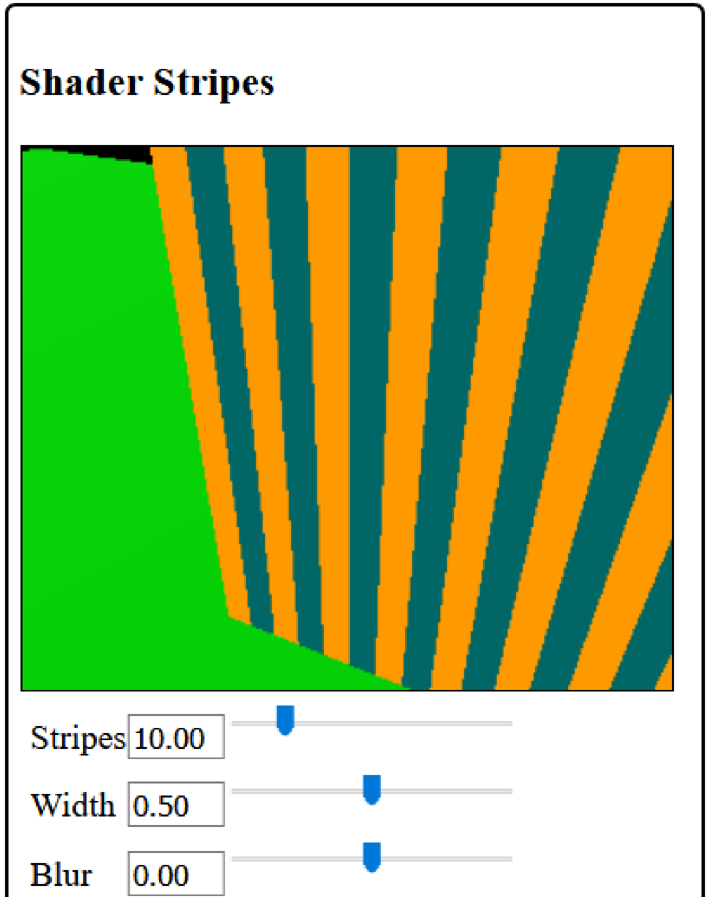
Stripes 10.00

Width 0.50

Blur 0.10

fwidth: just right!

No Blur



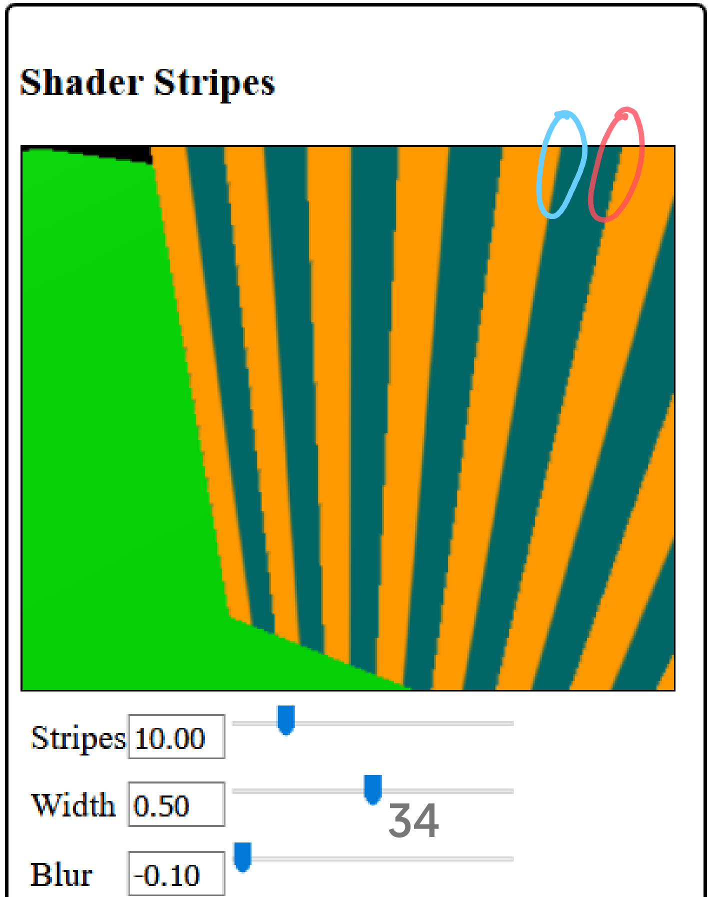
Shader Stripes

Stripes 10.00

Width 0.50

Blur 0.00

FWidth Blur



Shader Stripes

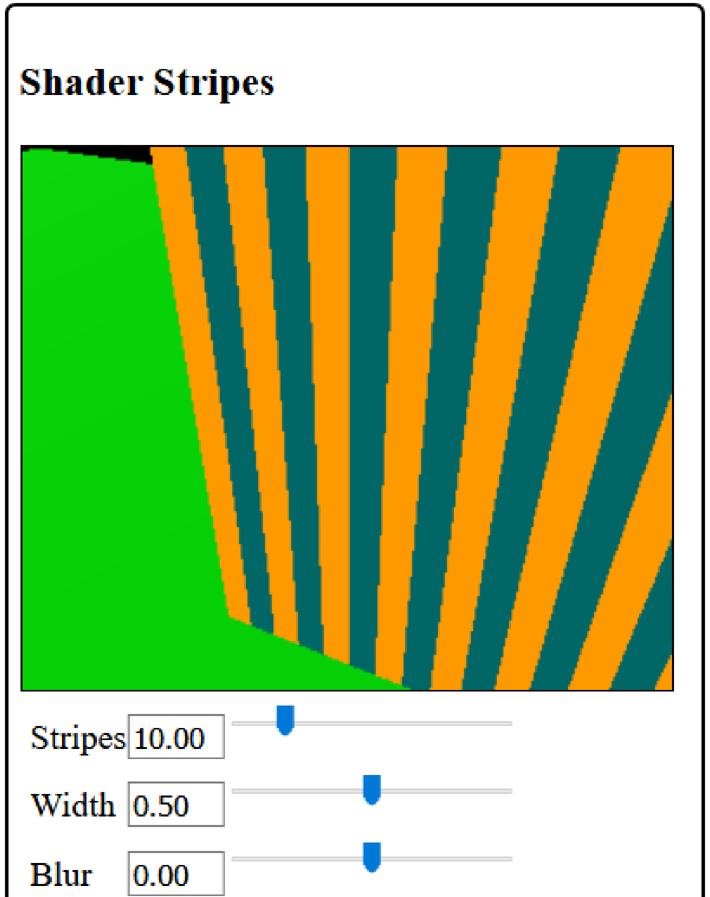
Stripes 10.00

Width 0.50

Blur -0.10

fwidth: just right!

No Blur



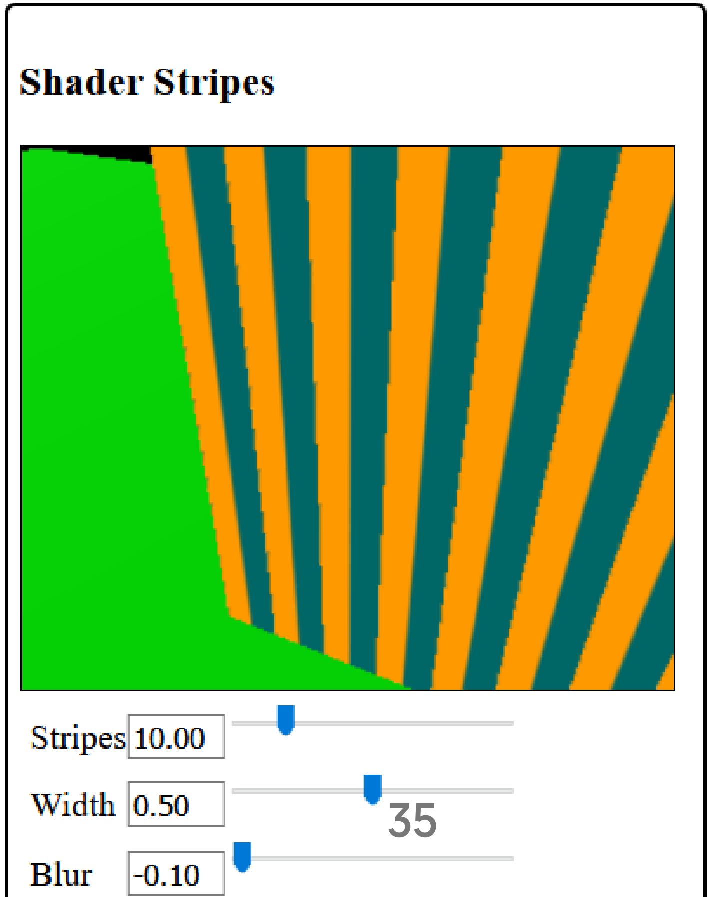
Shader Stripes

Stripes 10.00

Width 0.50

Blur 0.00

FWidth Blur



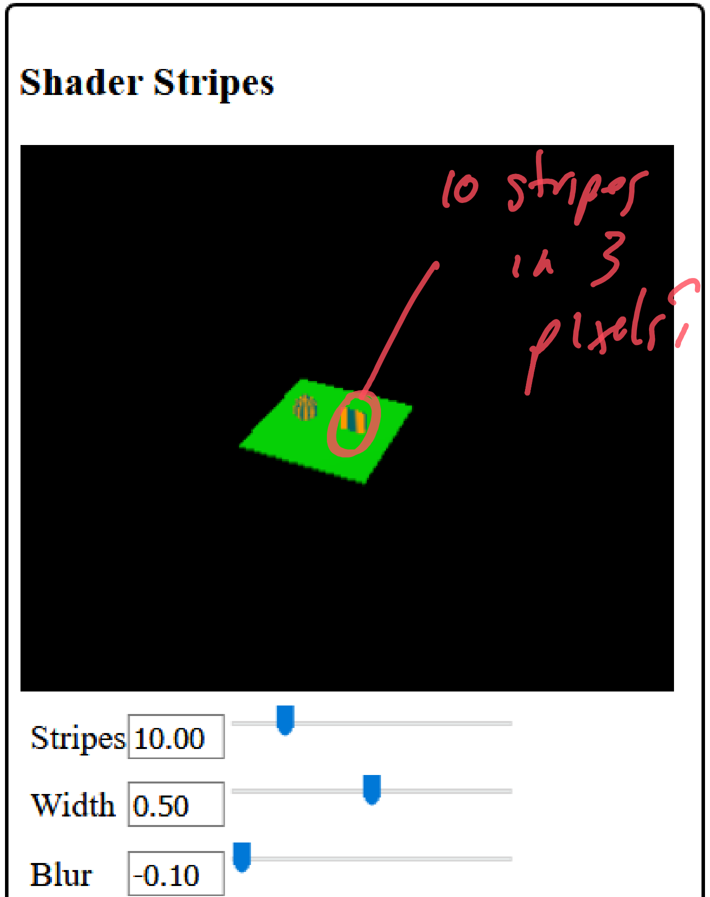
Shader Stripes

Stripes 10.00

Width 0.50

Blur -0.10

Zoom out?



Shader Stripes

10 stripes in 3 pixels?

Stripes 10.00

Width 0.50

Blur -0.10

Smoothstep handles texture **Magnification**

Not texture **Minification**

We still need some way to do filtering over a large range

- need the whole texture to a small number of pixels

Hard to do in a general way for procedures

Much easier to do for images

- MipMaps!

Shader Anti-Alising

What we can do:

1. mutliple samples (basic thing)
2. screen-space derivatives
3. edge smoothing

What we can't do:

1. Minification (large ranges in a pixel)
2. Triangle edges

(use big triangles and Image Textures?)