Curl Noise

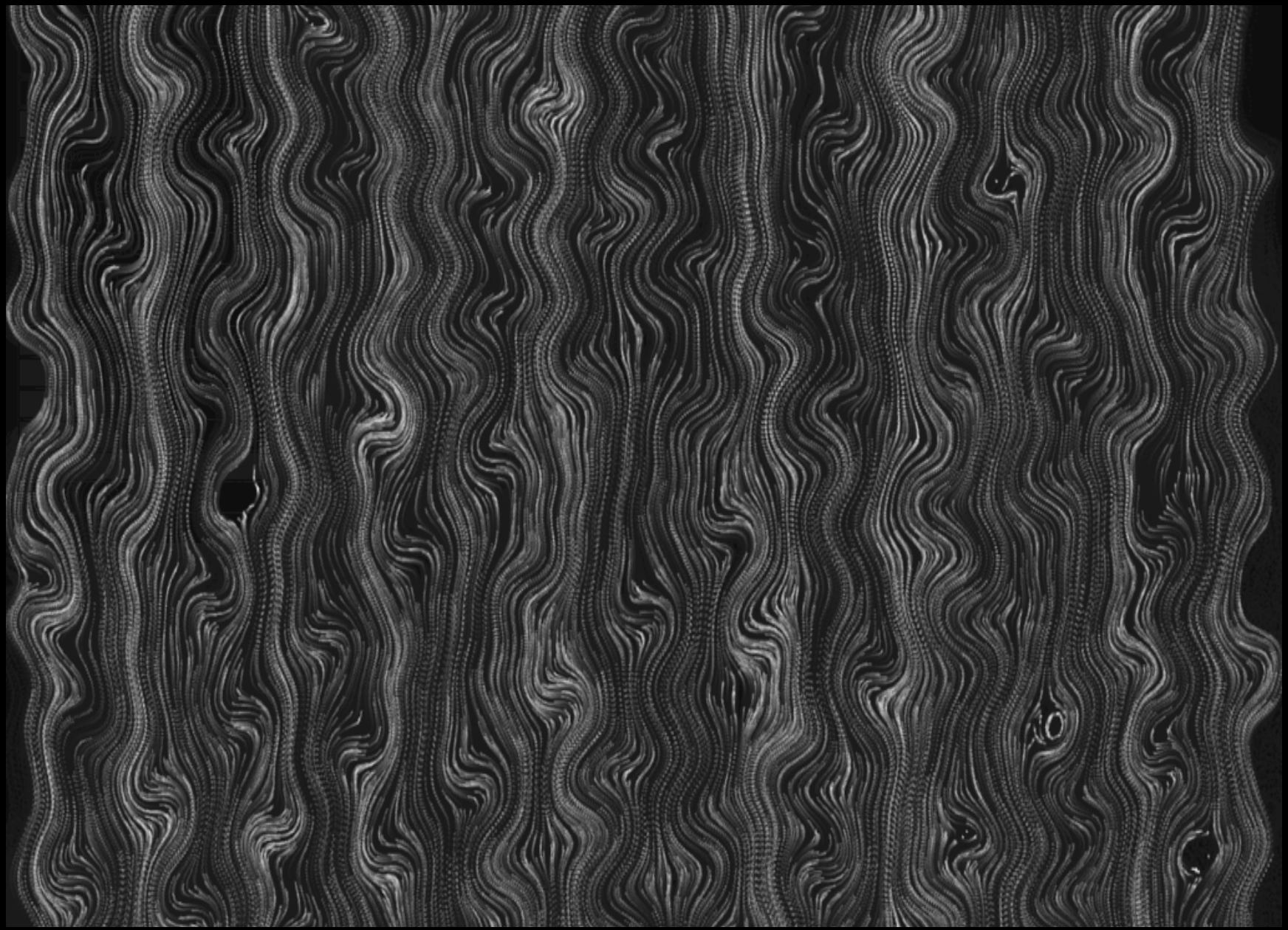
Peter Werner

Overview

- Take a look at end results
- Cover some background stuff
- Look at 2D case
- Some more background stuff
- Look at 3D case
- General computation considerations

Curl Noise

- Move a bunch of particles around
- Used for smoke, fire, fluid effects
- Relatively inexpensive to calculate
- Especially compared to other methods of fluid simulation
- Also just generally looks kinda cool



Objectives

- Aim is to understand what the curl operator does
- Can go off and read other peoples papers/ code
- Not a mathematically rigourous talk
- Interludes time for questions

Background

- Curl is a mathematical operator like +, -, etc
- Its input is a vector field
- Its output is a divergence free vector field
- Measure of rotational force
- In our case the input will be Perlin Noise
- We will use the output to move particles around
- A divergence free vector field will stop our particles smooshing together too much


Particles moving with straight perlin noise

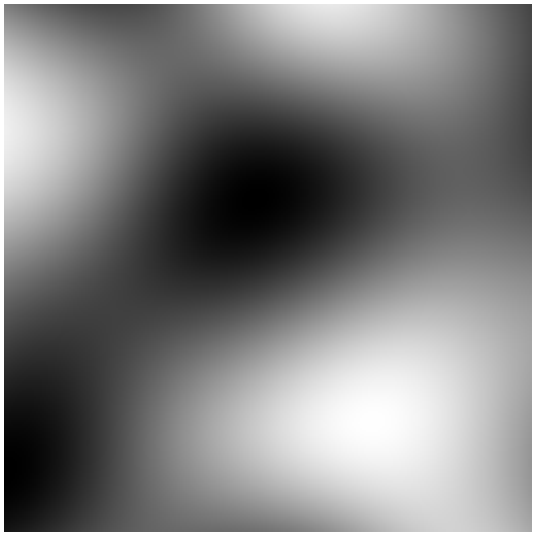

Particles moving with curl of perlin noise

Perlin Noise

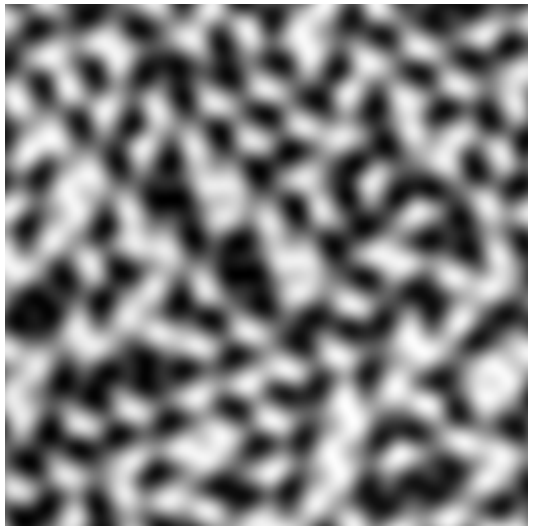
- Used whenever you want something that varies but not completely randomly
- Takes x/y/z locations
- Gives values from [0, 1] or [-1, 1]
- Low frequency noise varies from high values to low values slowly
- High frequency noise varies quickly
- Common to sum layers of noise at different frequencies

Perlin Noise

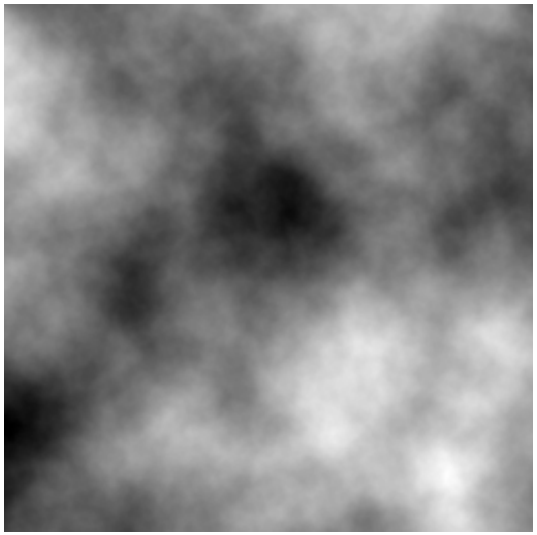
Low frequency



High frequency

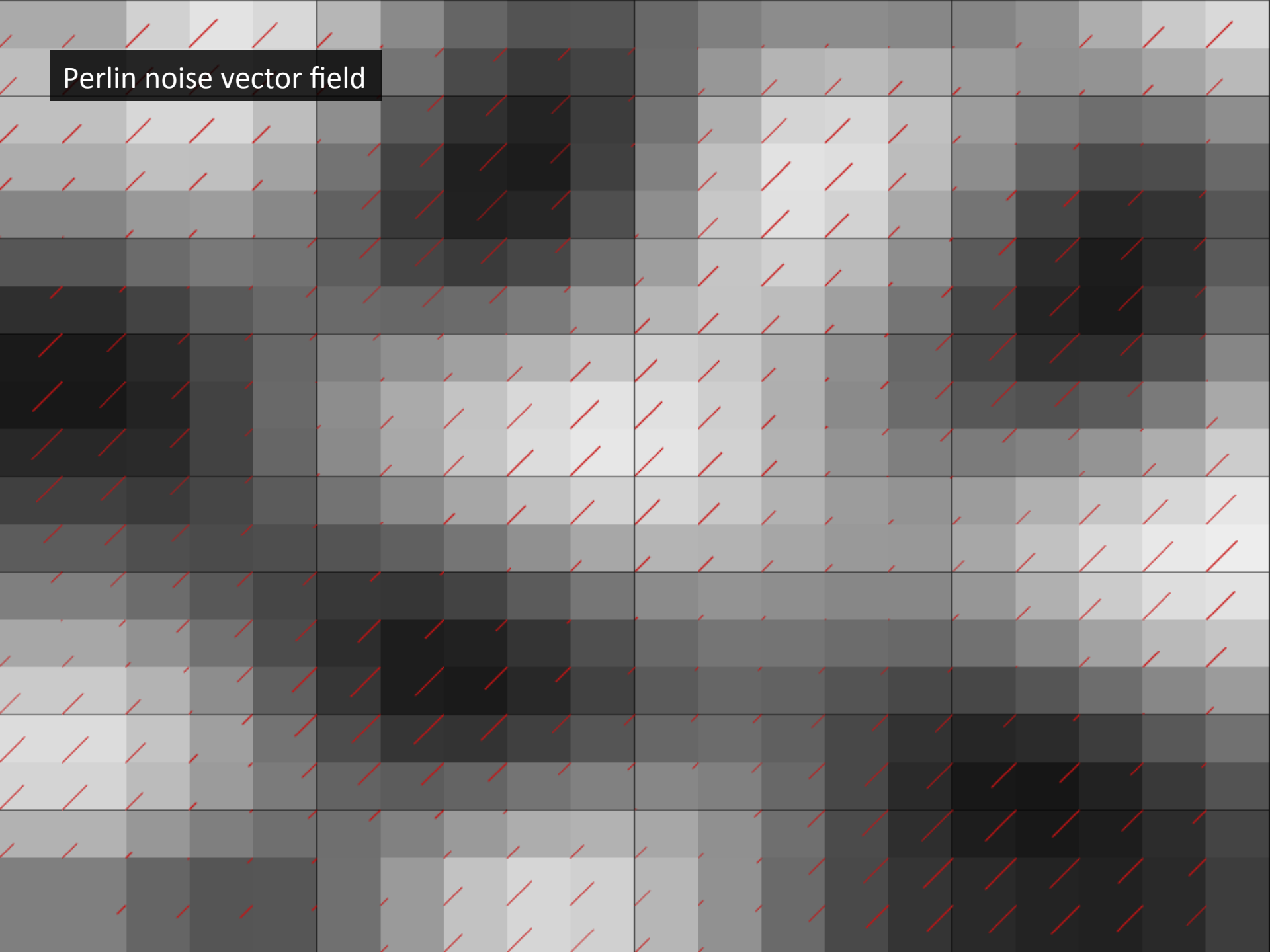


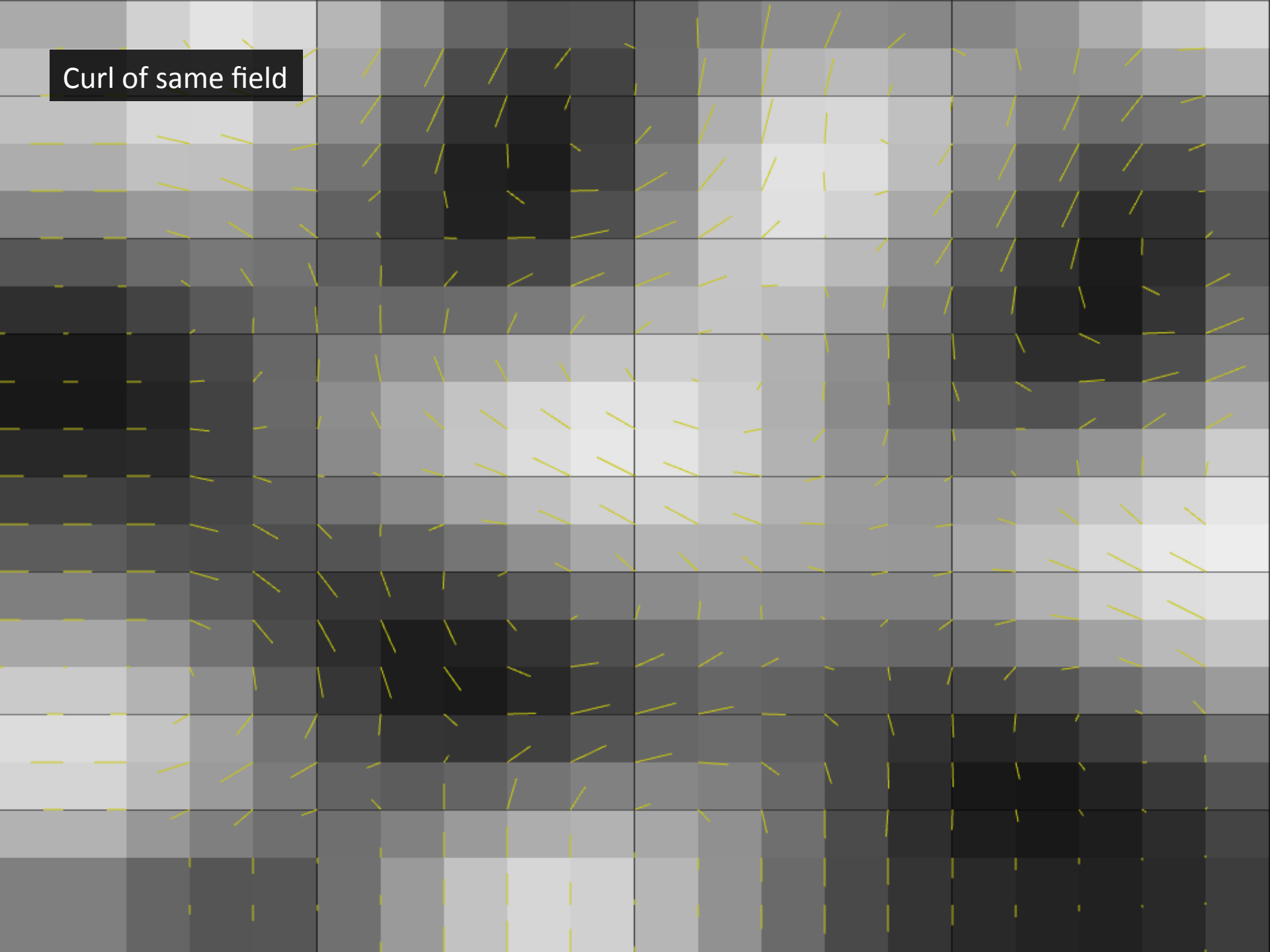
Sums of frequencies



Vector Field

- We use Perlin noise as our vector field
- For a particle at a certain (x, y) position
- Evaluate the noise function at (x, y) to get a value n
- Make our velocity vector v = (n, n)
- Next particle position = (x, y) + (n, n)


Perlin noise vector field


Curl of same field

Curl in 2D

- Potential Field (perlin noise) $\vec{\psi} = (\psi_1, \psi_2, \psi_3)$
- Instead: $F = (x, y, z)$
- In 2D: $F = (x, y)$
- Some point $P = (x_1, y_1)$
- Curl: $v(x, y) = \left(\frac{\partial x_1}{\partial y}, -\frac{\partial y_1}{\partial x} \right)$
- v(x, y) is the output of the curl operator.
- It has a vector value, so x,y in 2D, x,y,z in 3D
- The derivative terms are rates of change.

Curl in 2D

- Don’t worry, no calculus is required
- What does this term $\frac{\partial x_1}{\partial y}$ really mean?
- We want the rate of change in x
- At a given point x1
- On the y axis/relative to y
- i.e. how much the x1 value would change as the y values change around it
- Just a regular number as its value

Curl in 2D

- Similarly for $\frac{\partial y_1}{\partial x}$
- How much do y values change around some point y1 …
- As the values of x change?
- What's the rate of change in y relative to x?
- How much does y change as x changes?
- Gradient = derivative = rate of change

Computing the Curl

- Step 1: Calculate the gradients
- Step 2: Jigger the values round to fit curl definition
- e.g. Curl definition $v(x,y) = \left(\frac{\partial x_1}{\partial y}, -\frac{\partial y_1}{\partial x} \right)$
- Step 1: $a = \frac{\partial x_1}{\partial y}, \quad b = \frac{\partial y_1}{\partial x}$
- Step 2: $v(x,y) = (a, -b)$

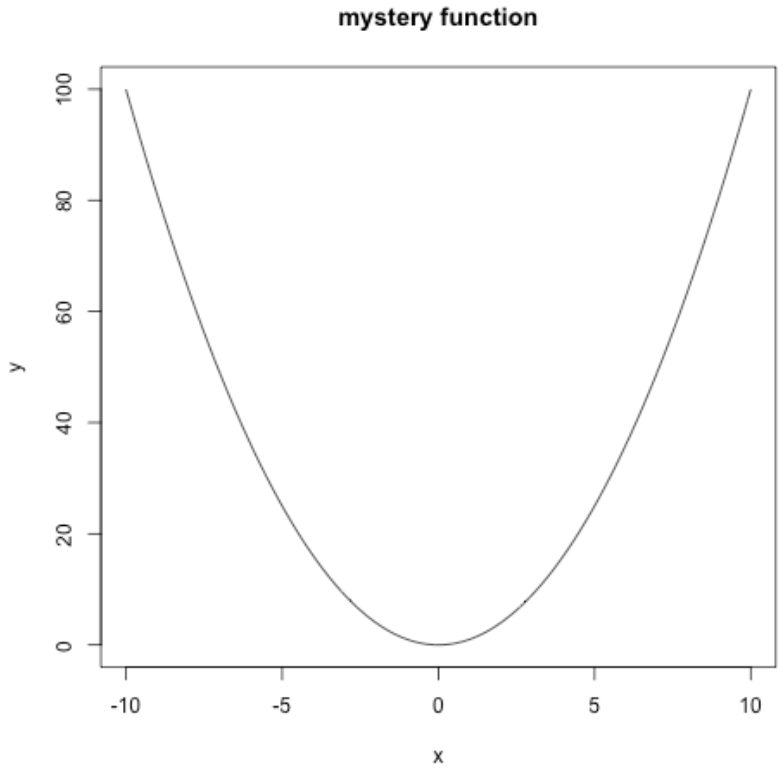
$$v(x,y) = \left(\frac{\partial x_1}{\partial y}, -\frac{\partial y_1}{\partial x} \right)$$

Finite Difference Gradient

- But how to compute the gradients?
- Can be done using calculus using simplex noise
- Much easier way is to approximate the rate of change using a method called finite differences.
- Super easy, uses primary school math

Finite Differences

- We have a mystery function we want to approximate a derivative for

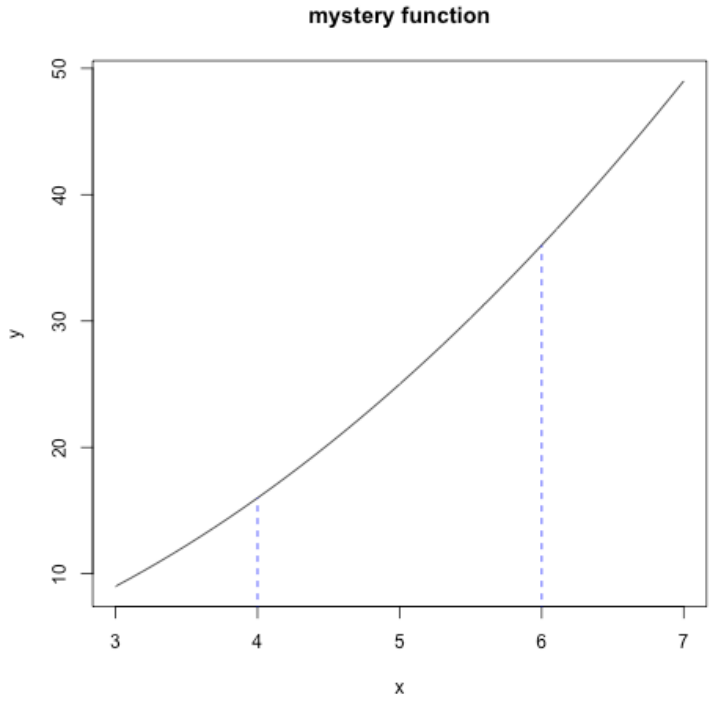


Finite Differences

- Have some 2D data with x and y values
- Want to approximate the rate of change of y as x changes (i.e. $\frac{dy}{dx}$) at say x = 5
- We know the x values and y values
- But not the function that generated them

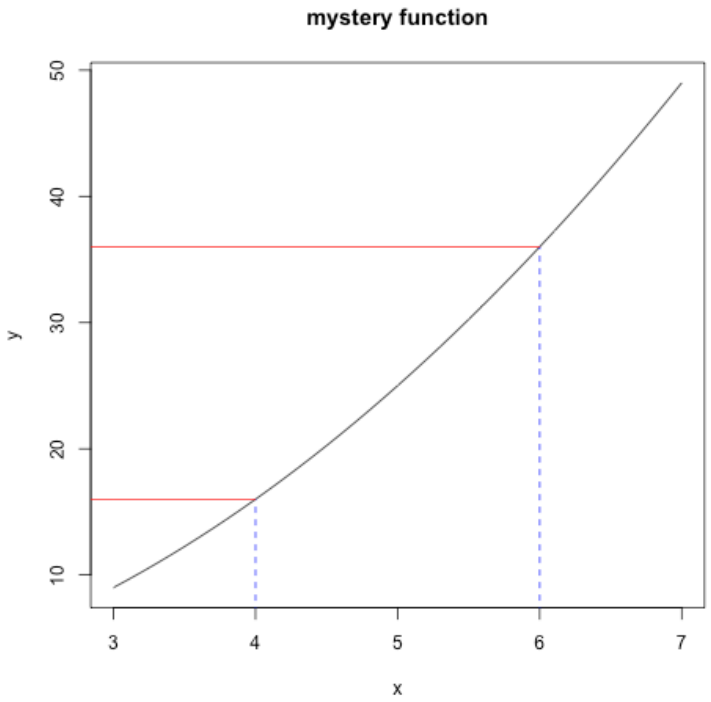
Finite Differences

- How about we take the average y values at points “close” to x = 5
- In this case, x = 4 and x = 6



Finite Differences

- Looking at our data, we get y = 16 and y = 36



Finite Differences

- Taking the average gives $\frac{36-16}{2} = \frac{20}{2} = 10$
- Our function is really $y = x^2$
- We know its derivative $\frac{dy}{dx} = 2x$
- Which at x = 5 gives 10, the same as our approximation

Finite Differences

- The general idea for some $\frac{\partial top}{\partial bot}$ or $\frac{d\,top}{d\,bot}$
- Move the values of bot a bit
- Get the top values at those points
- Subtract one of those values from the other
- Divide it by 2 to get the average
- This will approximate the gradient at that point

Finite Differences

- Common to talk of small differences in terms of the greek epsilon ε
- For our $\frac{dy}{dx}$ case with $y = f(x) = x^2$
- Set epsilon to some small value ($\varepsilon = 1$)
- Then look at $\frac{f(x+\varepsilon) - f(x-\varepsilon)}{2 \times \varepsilon} = \frac{f(5+1) - f(5-1)}{2 \times 1} = \frac{6^2 - 4^2}{2} = \ldots = 10$
- If you are using normalized texture coords epsilon might be 0.0001

Computing the Curl

- Step 1: Calculate the gradients
- Step 2: Jigger the values round to fit curl definition
- e.g. Curl definition 2D: $v(x,y) = \left(\frac{\partial x_1}{\partial y}, -\frac{\partial y_1}{\partial x} \right)$
- Step 1: $a = \frac{\partial x_1}{\partial y}, \quad b = \frac{\partial y_1}{\partial x}$
- Step 2: $v(x,y) = (a, -b)$

$$v(x,y) = \left(\frac{\partial x_1}{\partial y}, -\frac{\partial y_1}{\partial x} \right)$$

Computing the Curl

```
ofVec2f ComputeCurl(float x, float y)
{
   float eps = 1.0;
   float n1, n2, a, b;

   n1 = noise(x, y + eps);
   n2 = noise(x, y - eps);
   a = (n1 - n2)/(2 * eps);

   n1 = noise(x + eps, y);
   n2 = noise(x - eps, y);
   b = (n1 - n2)/(2 * eps);

   ofVec2f curl = ofVec2f(a, -b);
   return curl;
}
```

$$a = \frac{\partial x_1}{\partial y}$$

$$b = \frac{\partial y_1}{\partial x}$$

$$v(x,y) = \left(\frac{\partial x_1}{\partial y}, -\frac{\partial y_1}{\partial x} \right)$$

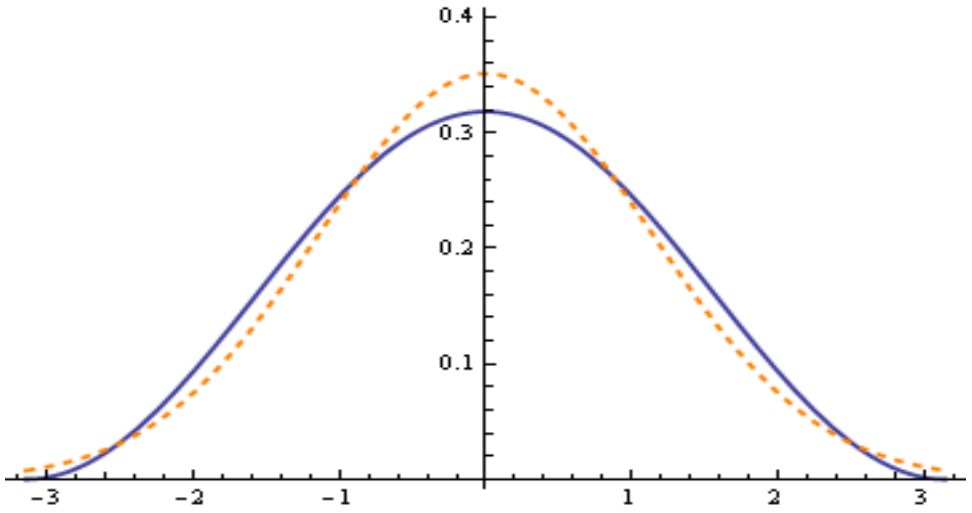
Recap

- Perlin Noise
- Vector Fields
- Curl in 2D
- Computing gradients
- Computing curl in 2D

Mathematical Fact

- cosine can be used to approximate the normal distribution

$$f(x) = \frac{1+\cos(x)}{2\pi}, \quad x \in (-\pi, \pi)$$



Curl in 3D

- Bridson: $\vec{v}(x,y,z)=\left(\frac{\delta\psi_3}{\delta y}-\frac{\delta\psi_2}{\delta z},\frac{\delta\psi_1}{\delta z}-\frac{\delta\psi_3}{\delta x},\frac{\delta\psi_2}{\delta x}-\frac{\delta\psi_1}{\delta y}\right)$
- Some point in 3D $P=(x_1,y_1,z_1)$
- We get $\vec{v}(x,y,z)=\left(\frac{\delta z_1}{\delta y}-\frac{\delta y_1}{\delta z},\frac{\delta x_1}{\delta z}-\frac{\delta z_1}{\delta x},\frac{\delta y_1}{\delta x}-\frac{\delta x_1}{\delta y}\right)$
- Three valued function (a vector in 3D)
- The last term has $\frac{\partial y_1}{\partial x}$ and $\frac{\partial x_1}{\partial y}$
- Just like we saw for the 2D case

Curl in 3D

- Curl $\vec{v}(x,y,z) = \left(\frac{\delta z_1}{\delta y} - \frac{\delta y_1}{\delta z}, \frac{\delta x_1}{\delta z} - \frac{\delta z_1}{\delta x}, \frac{\delta y_1}{\delta x} - \frac{\delta x_1}{\delta y} \right)$
- Gives a vector with three values:

$$\vec{v}(x,y,z) = (rate\ of\ change\ for\ x, rate\ of\ change\ for\ y, rate\ of\ change\ for\ z)$$

- Say we have some function $f(x,y,z) = x^2 + y^2 + z^2$
- To find the rate of change for z
- Hold z constant
- Will have an x component of change
- And a y component of change
- Hold z constant, jigger round x and y a bit

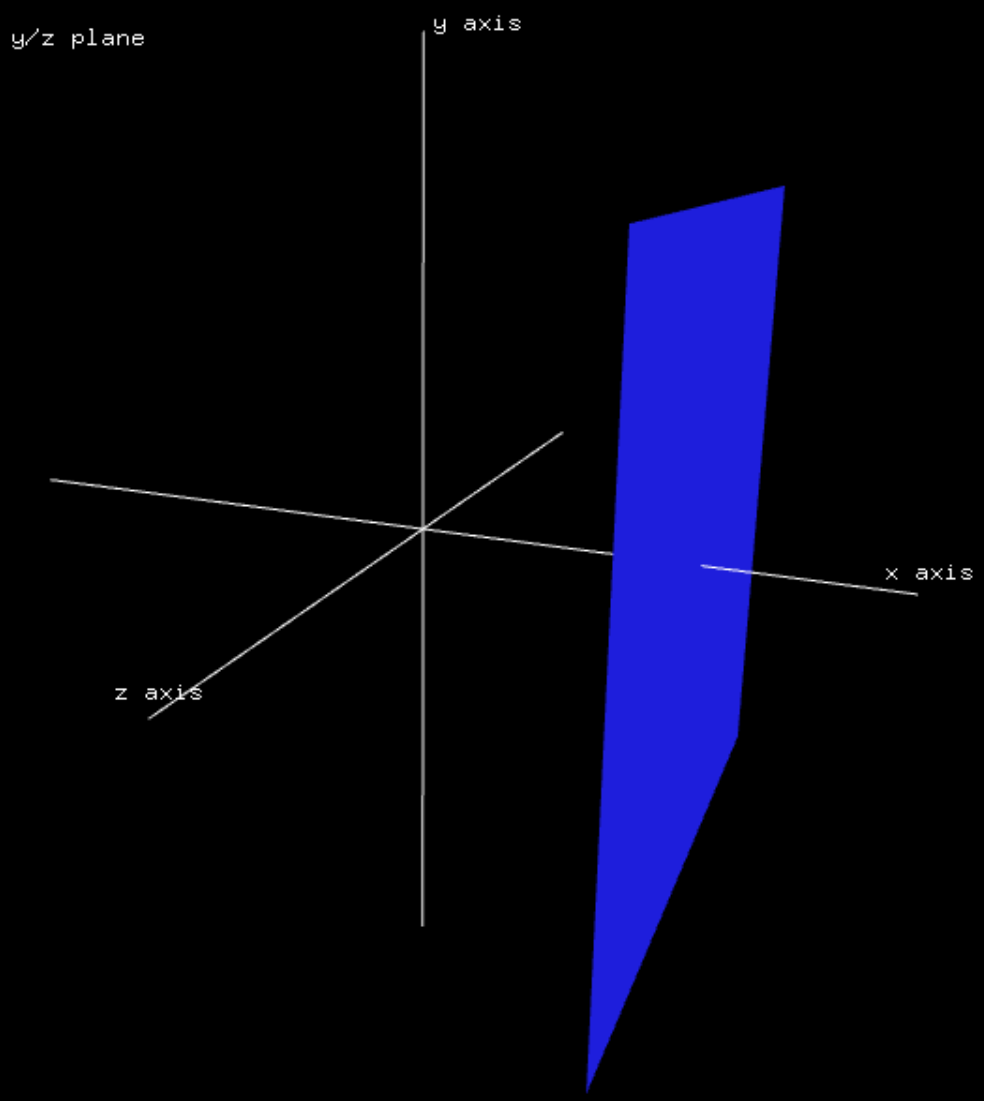
Curl in 3D

- Curl $\vec{v}(x,y,z) = \left(\frac{\delta z_1}{\delta y} - \frac{\delta y_1}{\delta z}, \frac{\delta x_1}{\delta z} - \frac{\delta z_1}{\delta x}, \frac{\delta y_1}{\delta x} - \frac{\delta x_1}{\delta y} \right)$
- Also need to find the rate of change for x
- Will vary relative to y and z locations
- The rate of change for y
- Will vary relative to x and z locations
- Let's call all these $\vec{v}(x,y,z) = (\Delta x, \Delta y, \Delta z)$
- Delta means "change in"

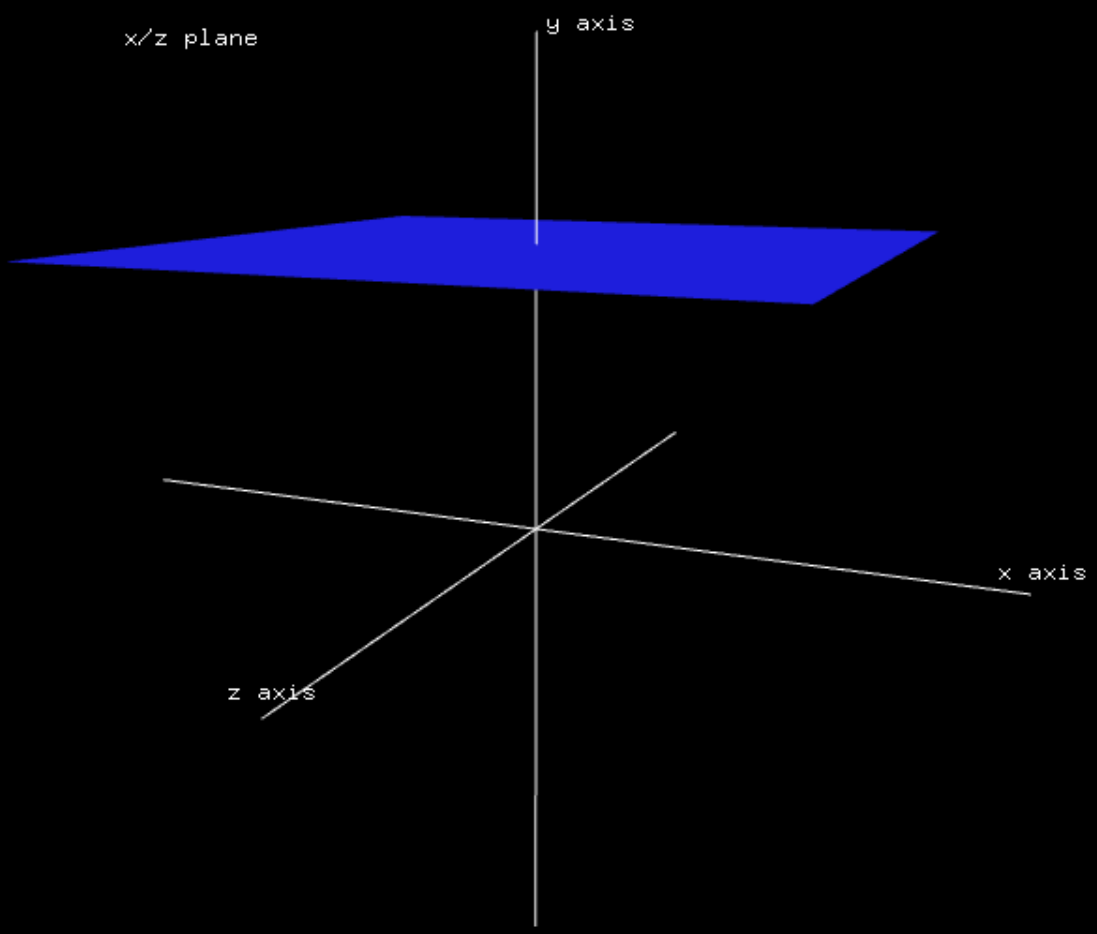
Curl 3D

- Curl $\vec{v}(x,y,z) = \left(\frac{\delta z_1}{\delta y} - \frac{\delta y_1}{\delta z}, \frac{\delta x_1}{\delta z} - \frac{\delta z_1}{\delta x}, \frac{\delta y_1}{\delta x} - \frac{\delta x_1}{\delta y} \right)$
- Notice $\Delta x = \frac{\delta z_1}{\delta y} - \frac{\delta y_1}{\delta z}$
- We only care about y/z plane
- Likewise $\Delta y = \frac{\delta x_1}{\delta z} - \frac{\delta z_1}{\delta x}$ x/z plane
- And $\Delta z = \frac{\delta y_1}{\delta x} - \frac{\delta x_1}{\delta y}$ all happening in the x/y plane
- Think of bottom parts as the 2D plane letters
- This enables a performance improvement when actually computing the values

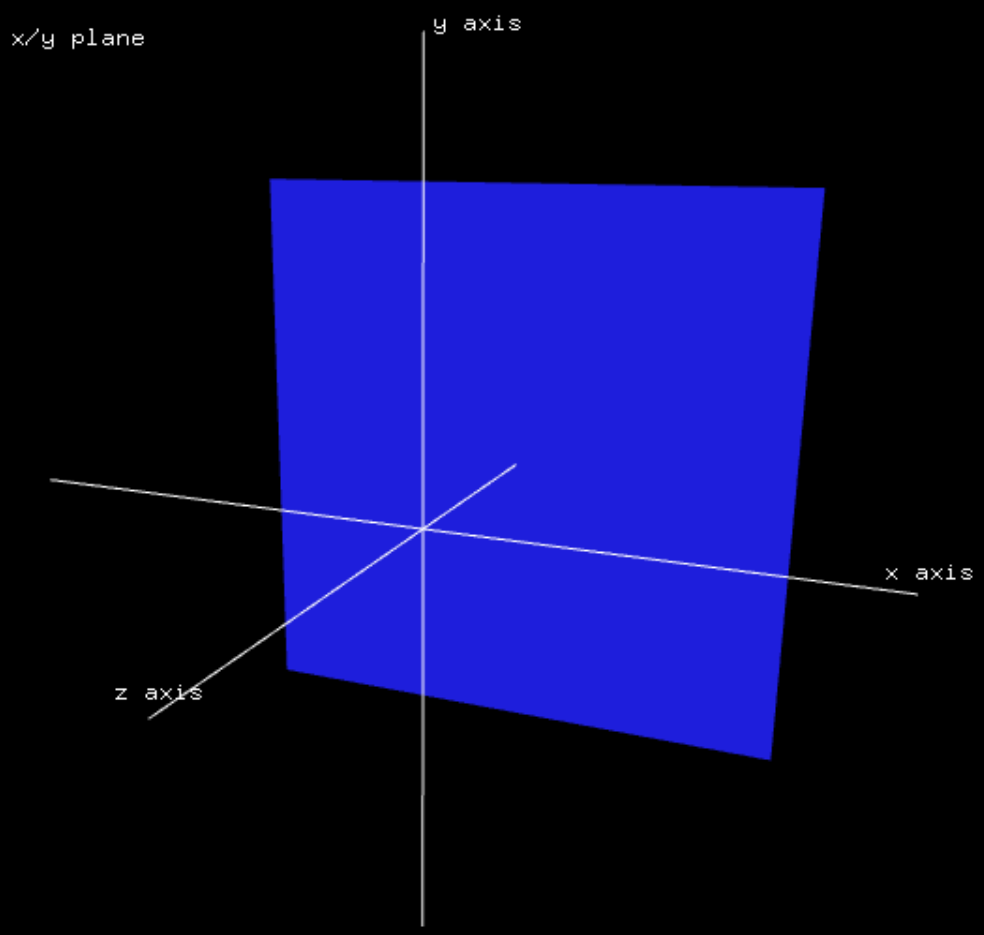
$$\Delta x = \frac{\delta z_1}{\delta y} - \frac{\delta y_1}{\delta z}$$



$$\Delta y = \frac{\delta x_1}{\delta z} - \frac{\delta z_1}{\delta x}$$



$$\Delta z = \frac{\delta y_1}{\delta x} - \frac{\delta x_1}{\delta y}$$



Curl in 3D

- Curl $\vec{v}(x,y,z) = \left(\frac{\delta z_1}{\delta y} - \frac{\delta y_1}{\delta z}, \frac{\delta x_1}{\delta z} - \frac{\delta z_1}{\delta x}, \frac{\delta y_1}{\delta x} - \frac{\delta x_1}{\delta y} \right)$
- Step 1: Calculate the gradients
- Step 2: Jigger the values round to fit curl definition
- If we think in terms of planes
- The gradients are just the same as the 2D case

Curl in 3D

- Lets take $\Delta x = \frac{\delta z_1}{\delta y} - \frac{\delta y_1}{\delta z}$
- Let $a = \frac{\partial z_1}{\partial y}, \quad b = \frac{\partial y_1}{\partial z}$
- Compute these two values as we did before
- For a, hold z fixed, move y around
- For b, hold y fixed, move z around
- We get $\Delta x = a - b$
- We now have $\vec{v}(x, y, z) = \left(\Delta x, \frac{\delta x_1}{\delta z} - \frac{\delta z_1}{\delta x}, \frac{\delta y_1}{\delta x} - \frac{\delta x_1}{\delta y} \right)$

Computing the Curl

- Continue on for the other components
- Step 1: Calculate the gradients
- Step 2: Jigger the values round to fit curl definition

Computing the Curl

```
ofVec3f ComputeCurl(float x, float y, float z)
{
    float eps = 1.0;
    float n1, n2, a, b;
    ofVec3f curl;

    n1 = noise(x, y + eps, z);
    n2 = noise(x, y - eps, z);
    a = (n1 - n2)/(2 * eps);

    n1 = noise(x, y, z + eps);
    n2 = noise(x, y, z - eps);
    b = (n1 - n2)/(2 * eps);

    curl.x = a - b;

    n1 = noise(x, y, z + eps);
    n2 = noise(x, y, z - eps);
    a = (n1 - n2)/(2 * eps);

    n1 = noise(x + eps, y, z);
    n2 = noise(x + eps, y, z);
    b = (n1 - n2)/(2 * eps);

    curl.y = a - b;

    n1 = noise(x + eps, y, z);
    n2 = noise(x - eps, y, z);
    a = (n1 - n2)/(2 * eps);

    n1 = noise(x, y + eps, z);
    n2 = noise(x, y - eps, z);
    b = (n1 - n2)/(2 * eps);

    curl.z = a - b;

    return curl;
}
```

$$\Delta x = \frac{\delta z_1}{\delta y} - \frac{\delta y_1}{\delta z}$$

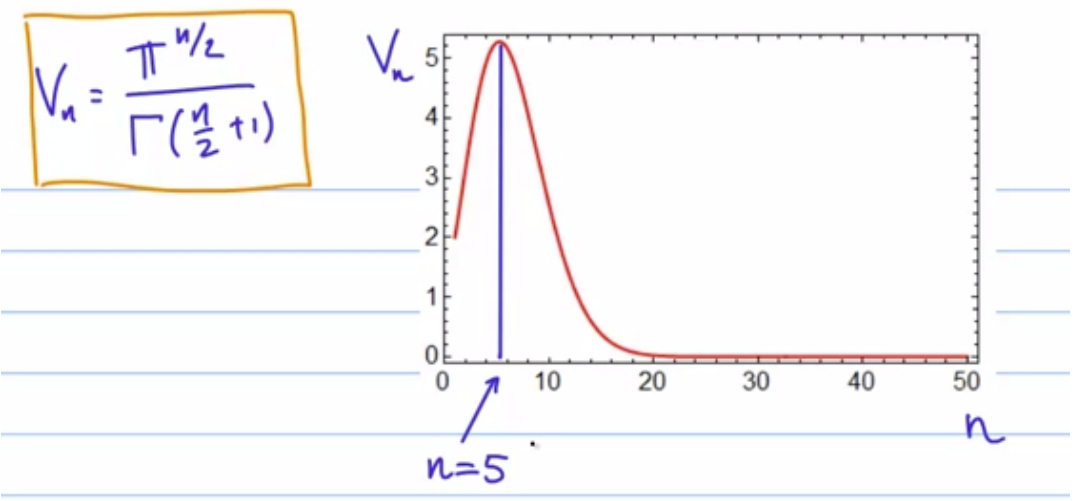
$$\Delta y = \frac{\delta x_1}{\delta z} - \frac{\delta z_1}{\delta x}$$

$$\Delta z = \frac{\delta y_1}{\delta x} - \frac{\delta x_1}{\delta y}$$

$$\vec{v}(x,y,z) = \left(\frac{\delta z_1}{\delta y} - \frac{\delta y_1}{\delta z}, \frac{\delta x_1}{\delta z} - \frac{\delta z_1}{\delta x}, \frac{\delta y_1}{\delta x} - \frac{\delta x_1}{\delta y} \right)$$

Mathematical Fact

- An infinite dimensional unit sphere (with radius = 1) has no volume



Computational Considerations

- Typically store noise values in floating point textures
- Can calculate noise on the fly
- Noise scales really well, examples use 64x64 noise textures
- Turn on bilinear interpolation and texture wrapping
- The 3D case could also use GL_TEXTURE_3D
- Can also use point sprites, billboarding, blending etc

Computational Considerations

- The 2D case uses one 2D texture
- My 3D case uses three 2D textures, one for each plane
- When sampling from textures add half a textel width

```
vec2 Grad(float x, float y)
{
   float h = 1/dims.x * 0.5;
   float eps = 1.0/dims.x;
   float n1, n2, dx, dy;
   vec2 dx0, dx1, dy0, dy1;
   dx0 = vec2(x, y - eps) + h;
   dx1 = vec2(x, y + eps) + h;
…
   n1 = texture(uPtex, dx0).r;
   n2 = texture(uPtex, dx1).r;
   dx = n1 - n2;
```

Computational Considerations

- May also want to normalize curl values

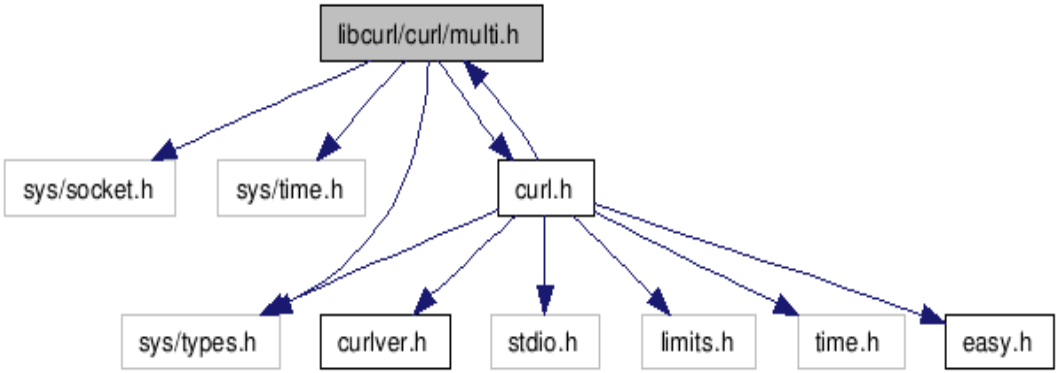
ofVec3f curl = CalculateCurl(x, y, z);

curl.normalize();

- Can skip the divison in finite differences
- Probably also want some fixed velocity for each particle
- Curl is a measure of rotational force, so particles will end up in loops

Computational Considerations

- I usually have a variable to modulate curl amount, eg from [0, 1]
- Also a time step value also from [0, 1]
- E.g.
- particle.xyz += (fixedVel + curl * curlAmt) * dt;
- Where curlAmt ranges [0,1]
- dt is time step also from [0,1]


libcurl/curl/multi.h
sys/socket.h
sys/time.h
curl.h
sys/types.h
curlver.h
stdio.h
limits.h
time.h
easy.h


NRG

References

- Robert Bridson, original paper + code http://www.cs.ubc.ca/~rbridson/
- Philip Rideout, code/tutorials http://prideout.net/blog/
- MiauMiau, WebGL code http://www.miaumiau.cat
- Vector Field http://tutorial.math.lamar.edu/Classes/CalcIII/VectorFields.aspx
- Partial Derivatives http://tutorial.math.lamar.edu/Classes/CalcIII/PartialDerivatives.aspx
- Curl http://tutorial.math.lamar.edu/Classes/CalcIII/CurlDivergence.aspx
- Bridson has a book on fluid simulation
- Rideout has a cool book on iPhone OpenGL

Me

- Blog http://petewerner.blogspot.com.au/
- Twitter https://twitter.com/dizzy_pete
- Tumblr http://i-am-noise.tumblr.com/
- Vimeo https://vimeo.com/dizzypete
- Github https://github.com/petewerner