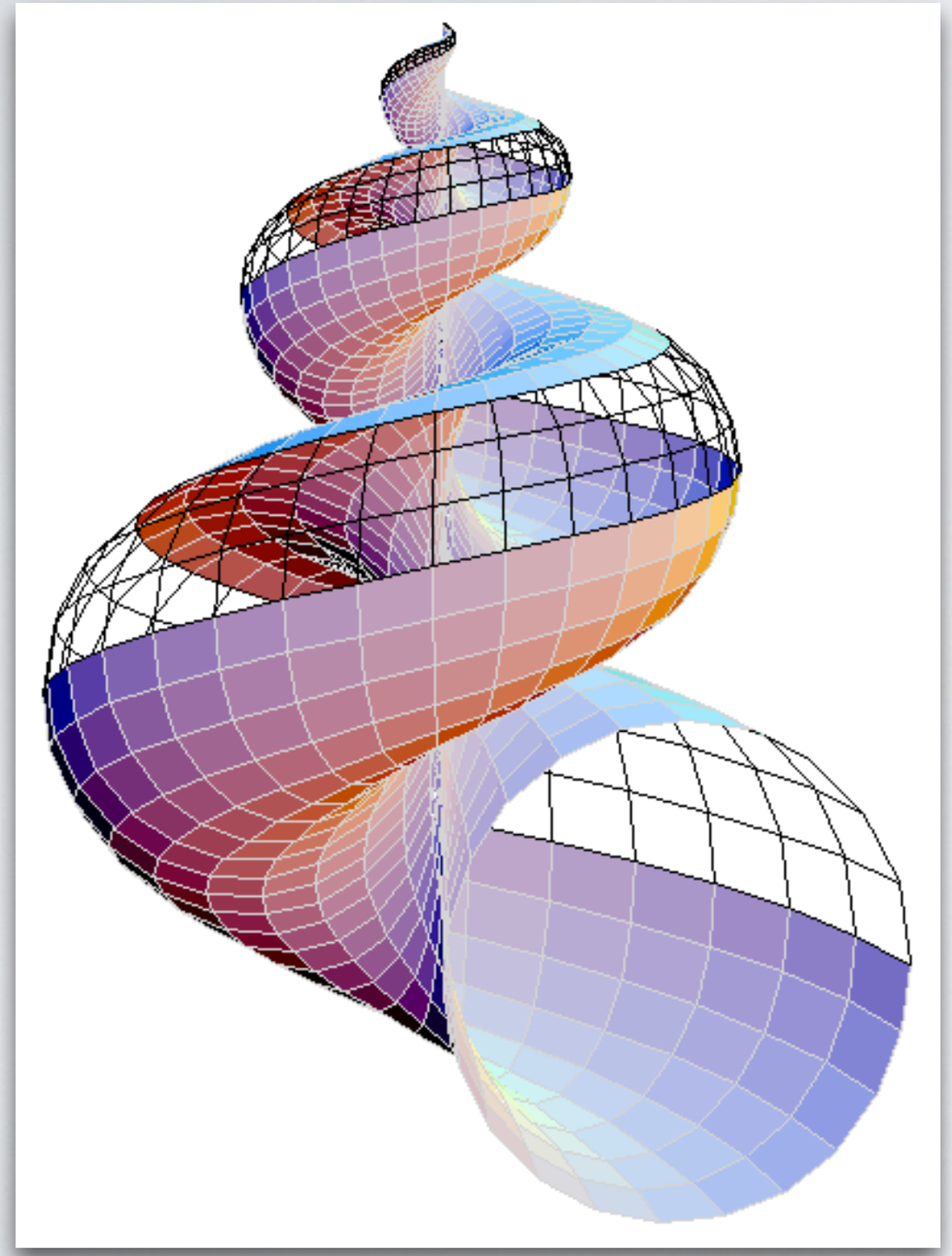
TOOLBOX FUNCTIONS

useful ways to modify and shape signals

Audubon (source)

University of Pennsylvania - CIS 700 Procedural Graphics
Rachel Hwang

THE SHAPE OF FUNCTIONS

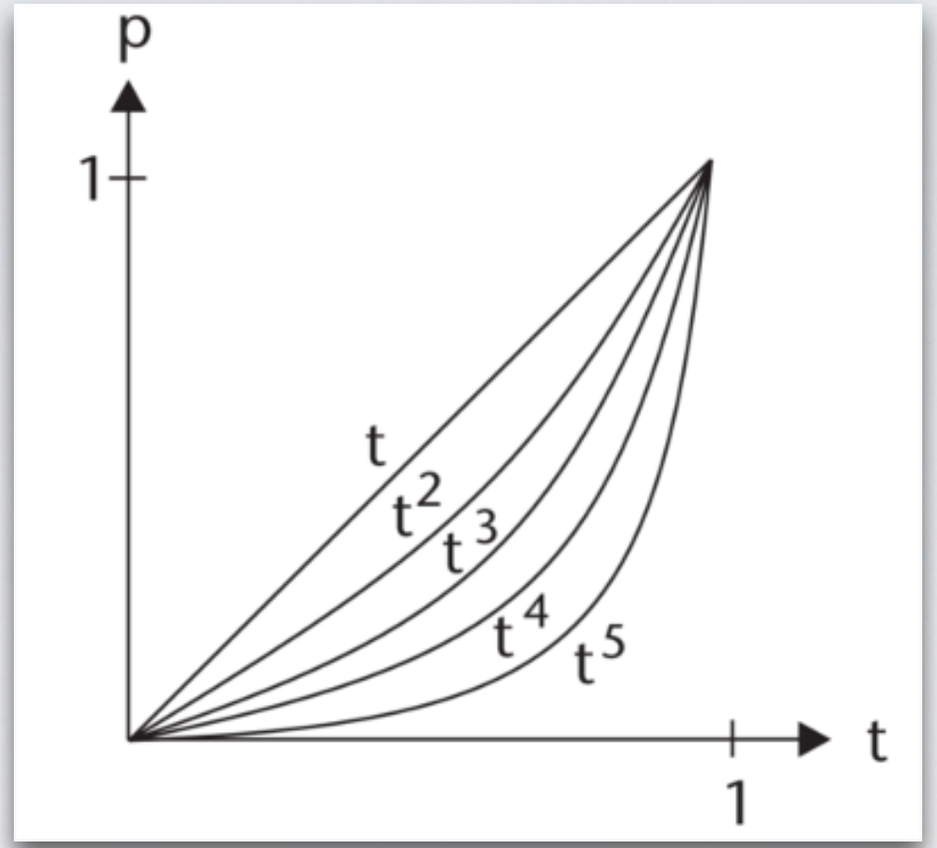


xahlee.info (source)

- Many natural systems are governed by structure that can be described functionally
- We can modify signals to produce natural-looking shapes, blends, animation, etc.
- Given an arsenal of functions and intuition about how to combine them, we can draw anything…

TRANSITIONS

BEYOND LINEAR



Penner (source)

- To move between values (norm 0-1), we have a lot of options.
- A nice place to start is Penner's easing functions
- Intended for animation but applicable for lots of graphics work

BASIC EASING

```
// t = time
// b = start value
// c = end value
// d = duration
float ease_linear(float t, float b, float c, float d) {
  return c * (t / d) + b;
}
```

- Different easing functions basically just modify t, we can simplify like this:

```
float ease_in_quadratic(float t) {
  return t * t;
}

// Do quadractic ease-in
t = t / d;
t = ease_in_quadratic(t);
float result = t * (c-b) + b;
```

BASIC EASING

- A complete set looks like this

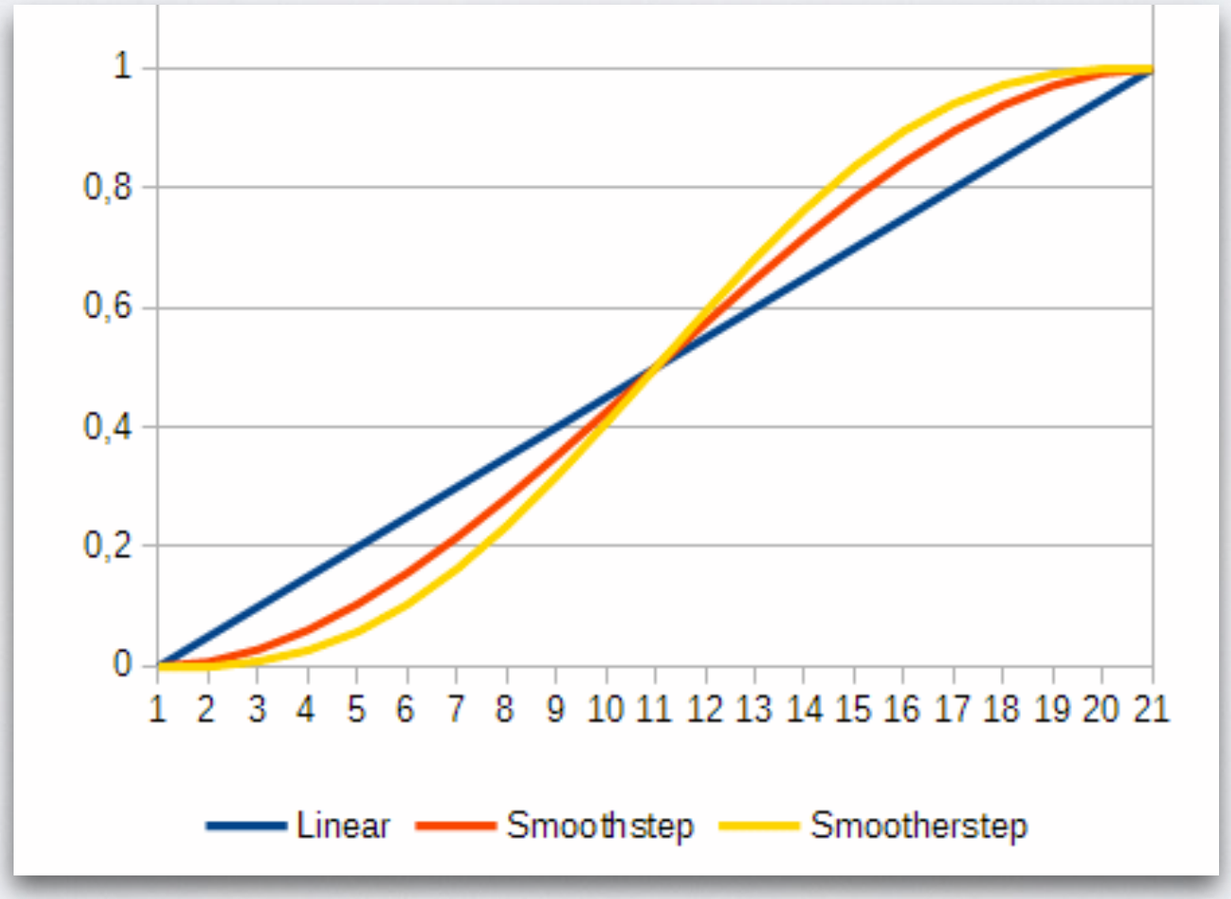
```
float ease_in_quadratic(float t) {
  return t * t;
}

float ease_out_quadratic(float t) {
  return 1 - ease_in_quadratic(1 - t);
}

float ease_in_out_quadratic(float t) {
  if (t < 0.5)
    return ease_in_quadratic(t * 2) / 2;
  else
    return 1 - ease_in_quadratic((1-t) * 2) / 2;
}
```

SMOOTH STEP

- Also known as Hermite blending
- smooth step has first order continuity. smoother step has second order continuity.
- step and smooth step are built in to glsl (lerp is "mix", fyi)

$$\text{smoothstep}(x) = 3x^2 - 2x^3$$

$$\text{smootherstep}(x) = 6x^5 - 15x^4 + 10x^3$$



Sol Tutorials (source)

SMOOTH STEP

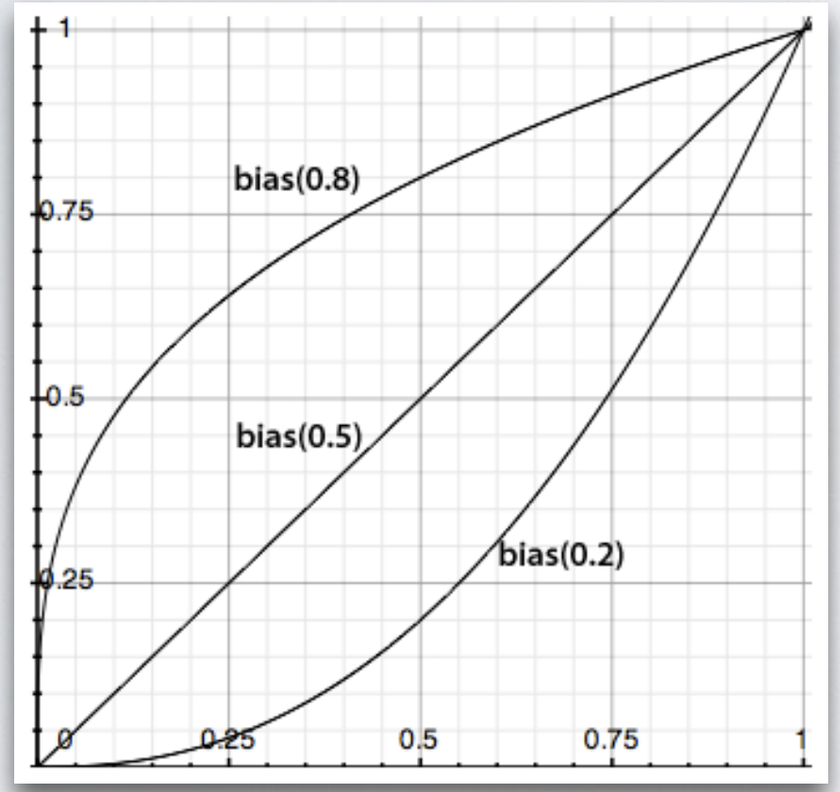
- Simple implementation

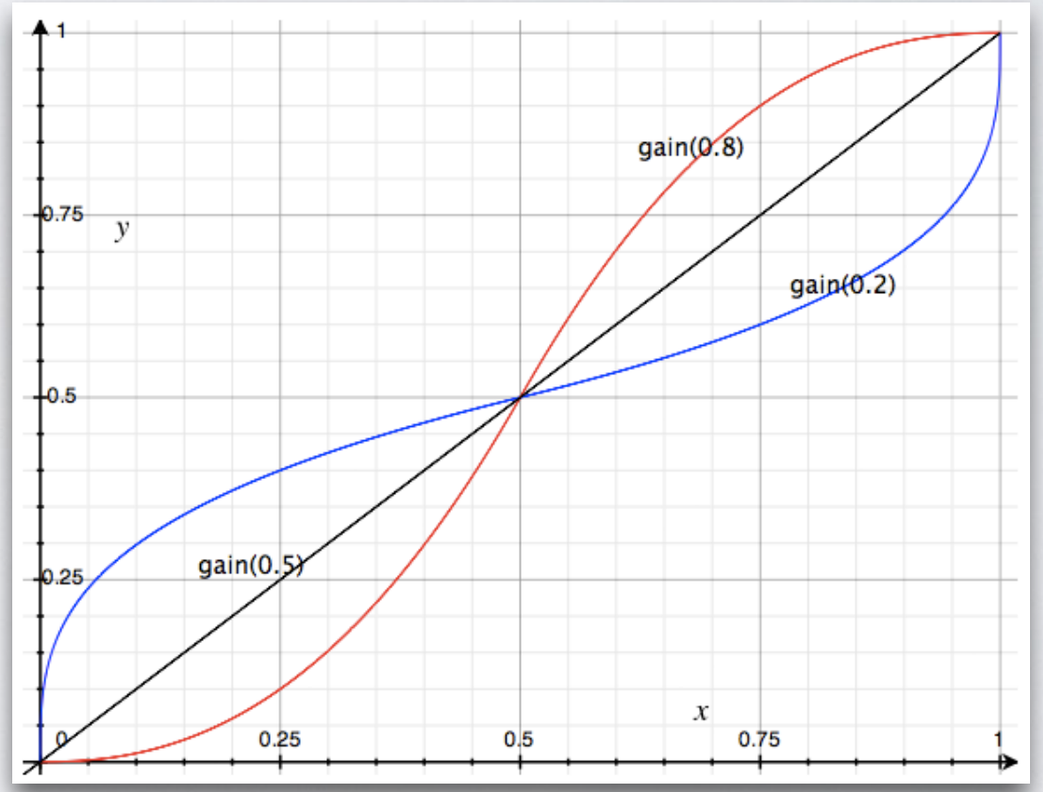
```
// Official AMD implementation
float smoothstep(float edge0, float edge1, float x)
{
    // Scale, bias and saturate x to 0..1 range
    x = clamp((x - edge0)/(edge1 - edge0), 0.0, 1.0);
    // Evaluate polynomial
    return x*x*(3 - 2*x);
}
```

```
float smootherstep(float edge0, float edge1, float x)
{
    // Scale, and clamp x to 0..1 range
    x = clamp((x - edge0)/(edge1 - edge0), 0.0, 1.0);
    // Evaluate polynomial
    return x*x*x*(x*(x*6 - 15) + 10);
}
```

BIAS AND GAIN

- But procedurals are all about parameter tuning!
- Perlin created a generalized form, the bias and gain functions.





Electric Fluid (source)

- Bias: How much time is spent at either end of the transition?
- Gain: How much time is spent in the middle of the transition?

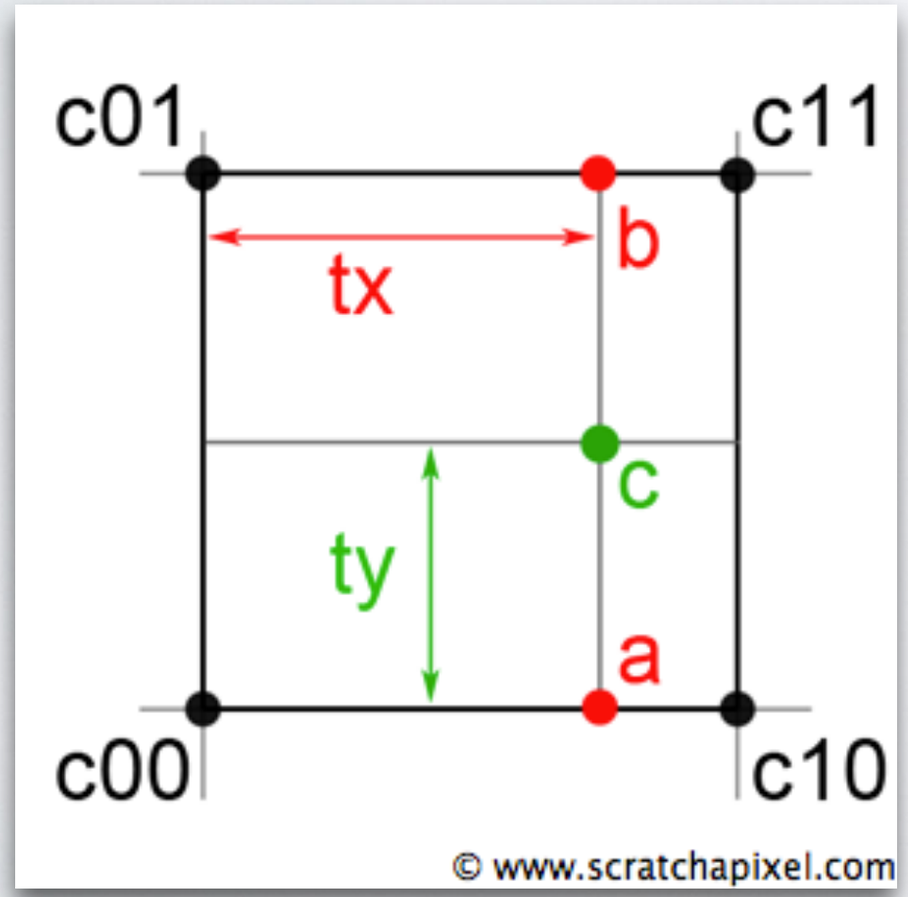
BIAS AND GAIN

- Simple implementation:

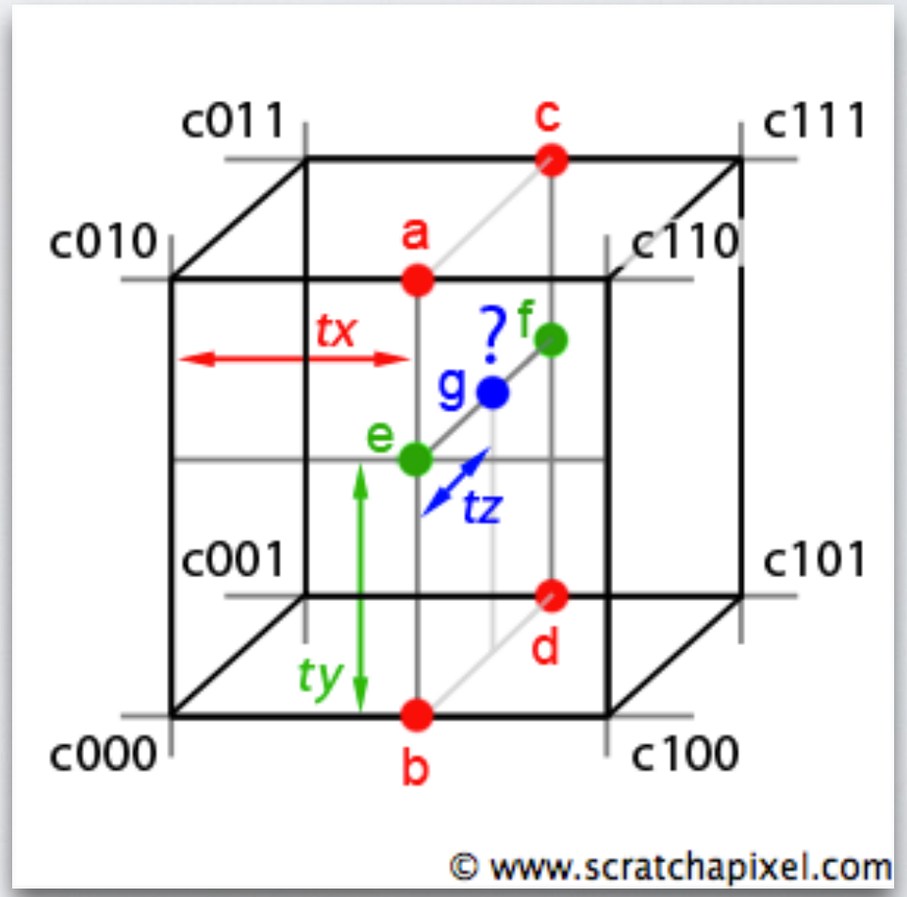
```
float bias (float b, float t) {
  return pow(t, log(b) / log(0.5f));
}

float gain (float g, float t) {
  if (t < 0.5f)
    return bias(1-g, 2*t) / 2;
  else
    return 1 - bias(1-g, 2 - 2*t) / 2;
}
```

HIGHER DIMENSIONS

- Interpolation can be extended to higher dimensions!
- Do simple 1D interpolation in each dimension, combining results with more interpolation. Think of it like a tournament bracket.
- How many leaps do you need for interpolation between n points?



bilinear (2D)

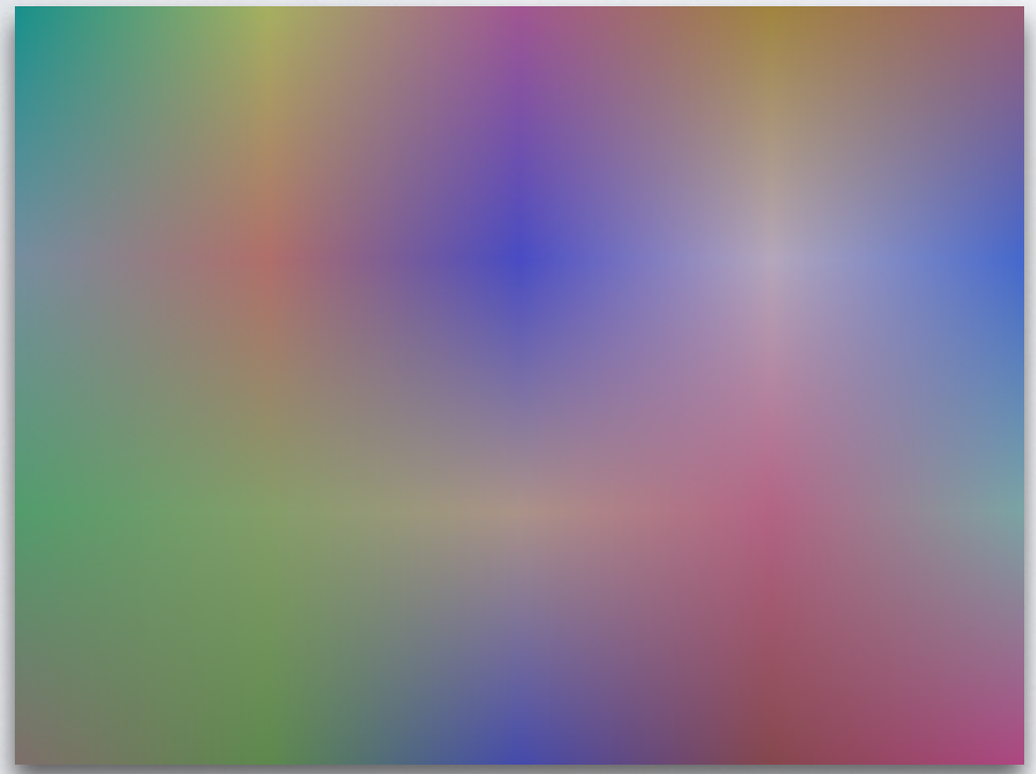


trilinear (3D)

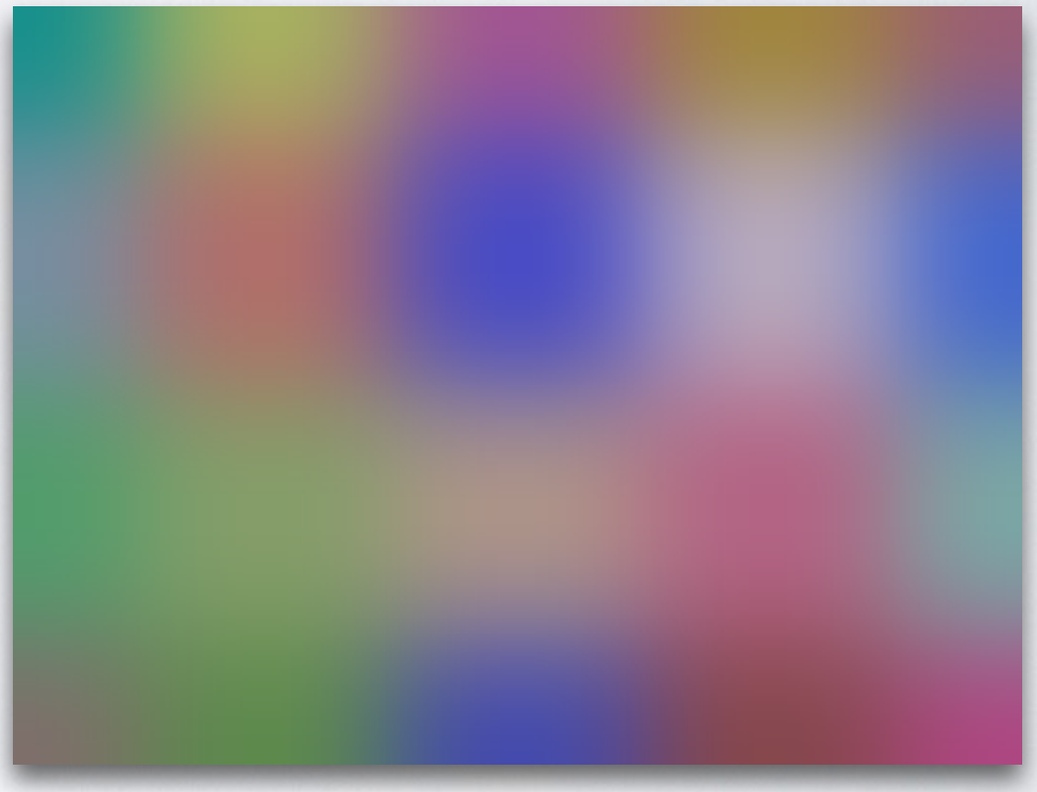
scratchapixel.com (source)

HIGHER DIMENSIONS

- We're not limited to linear interpolation in higher dimensions!
- Just replace lerp with a different interpolation technique.



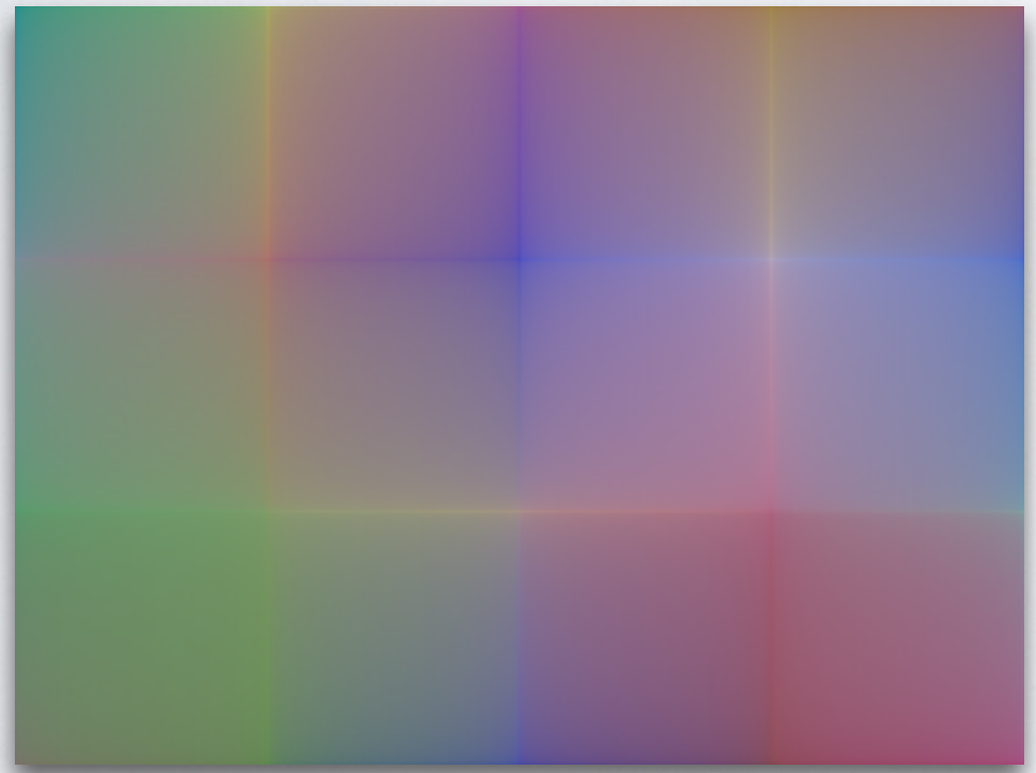
linear



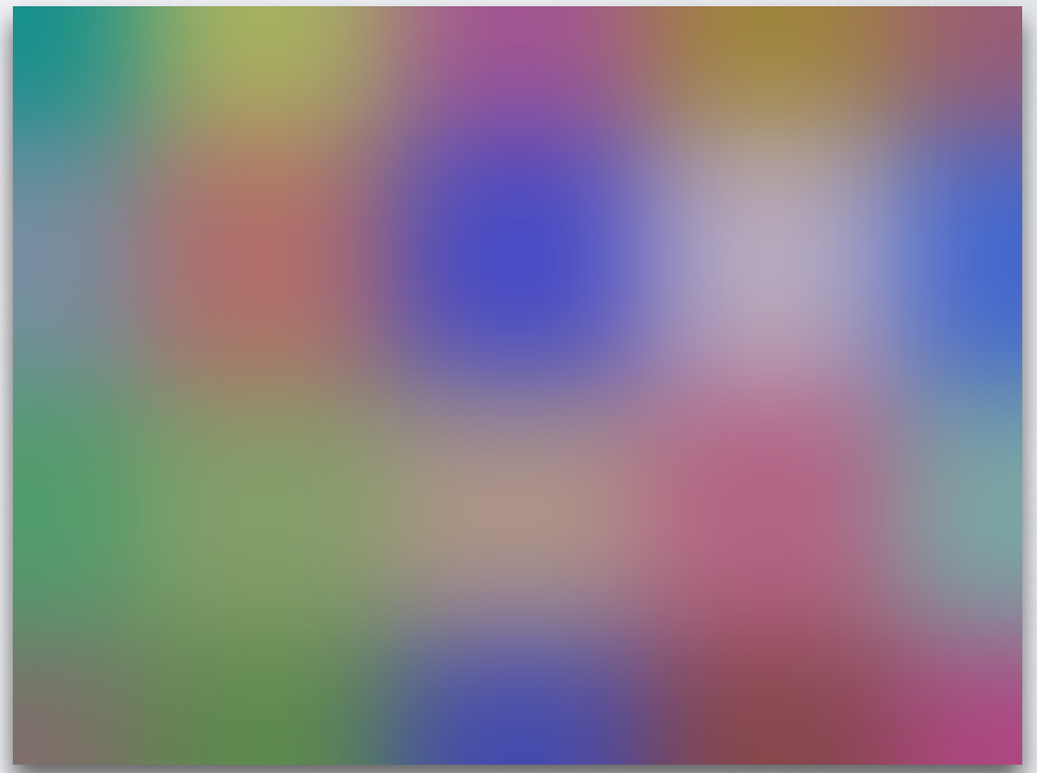
quadratic

HIGHER DIMENSIONS

- We're not limited to linear interpolation in higher dimensions!
- Just replace lerp with a different interpolation technique.



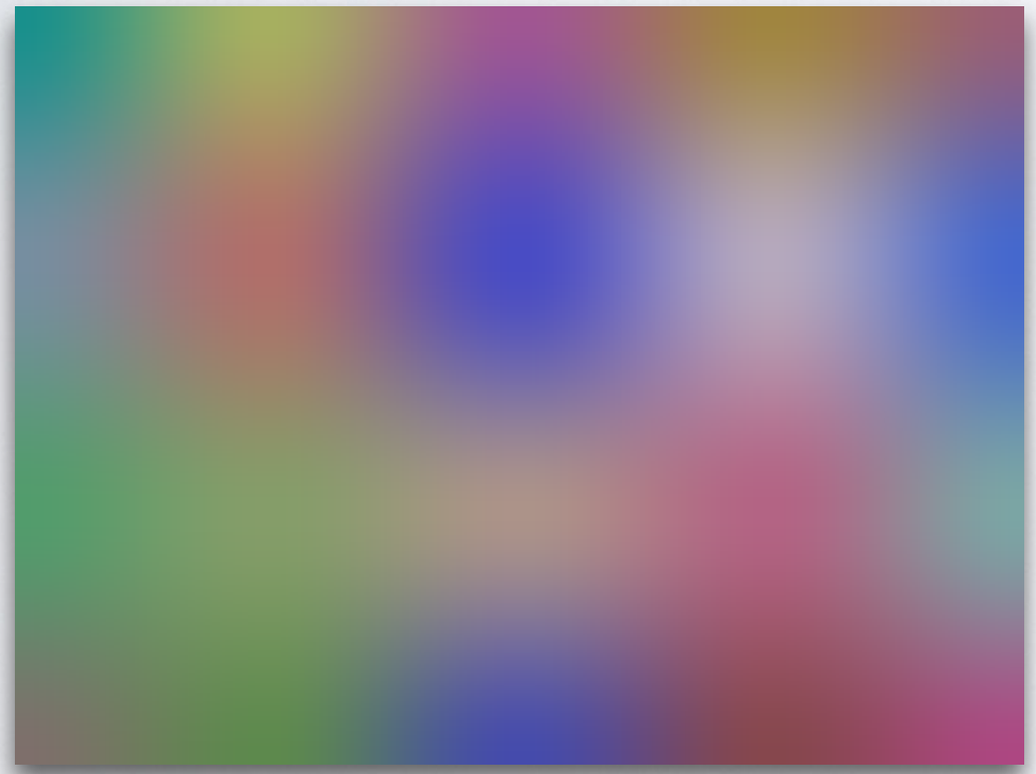
gain 0.25



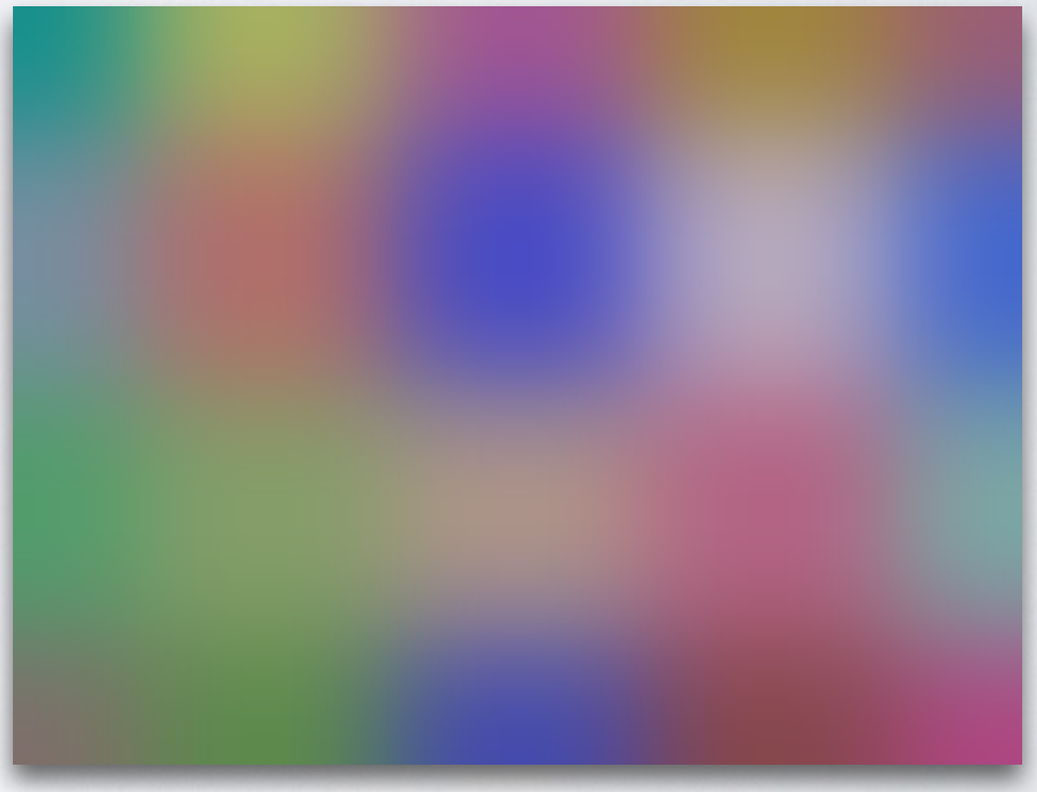
gain 0.75

HIGHER DIMENSIONS

- We're not limited to linear interpolation in higher dimensions!
- Just replace lerp with a different interpolation technique.



smooth step

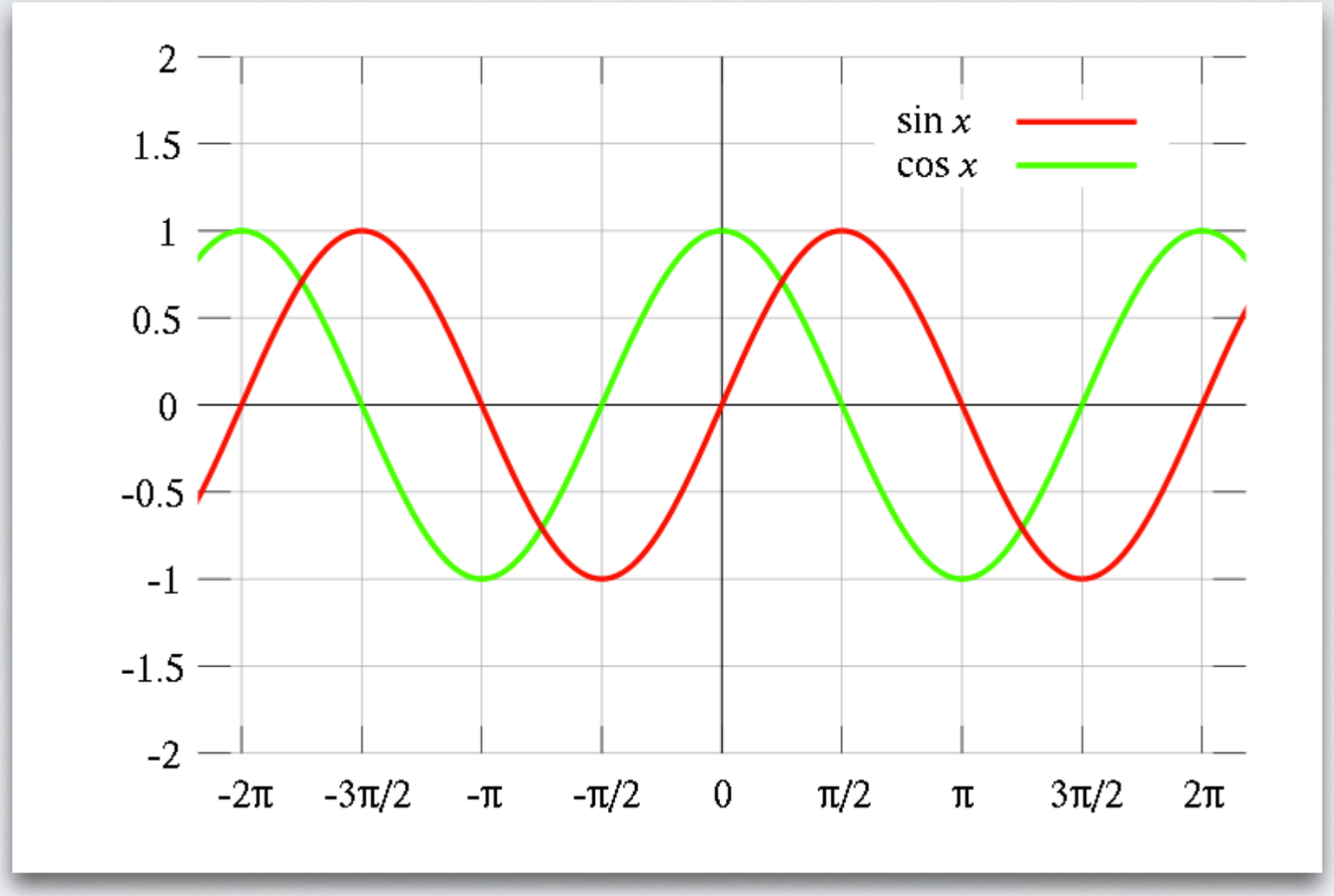


smoother step

WAVES

SIN AND COS

- Workhorse functions for producing smooth oscillation



wikimedia.org (source)

SQUARE WAVE

- Sharp oscillation between two values

Omegatron (source)

- How do we implement this?

SQUARE WAVE

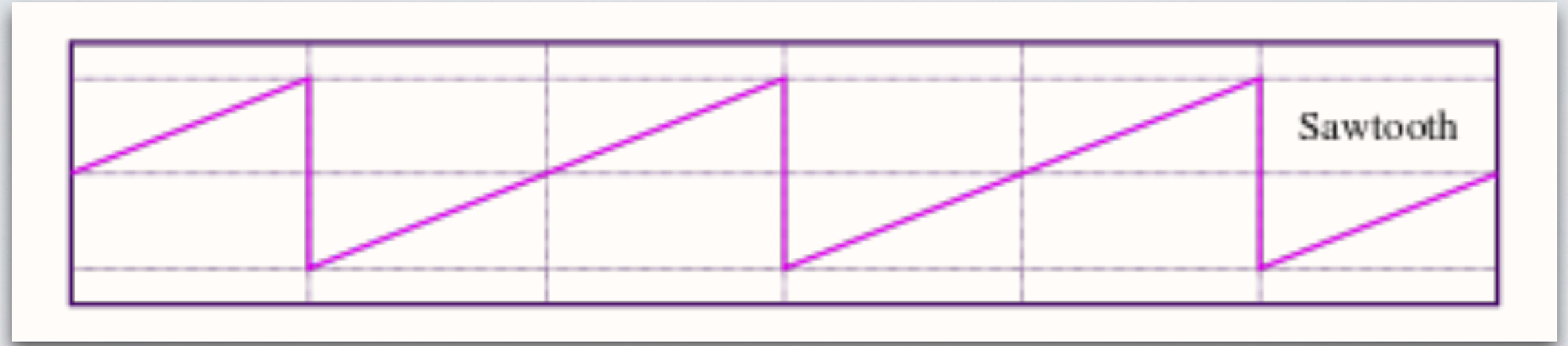
- Sharp oscillation between two values

Omegatron (source)

```
float square_wave(float x, float freq, float amplitude) {
  return fabs(floor(x * freq) % 2 * amplitude);
}
```

SAWTOOTH WAVE

- Jagged oscillation — value increases linearly then resets

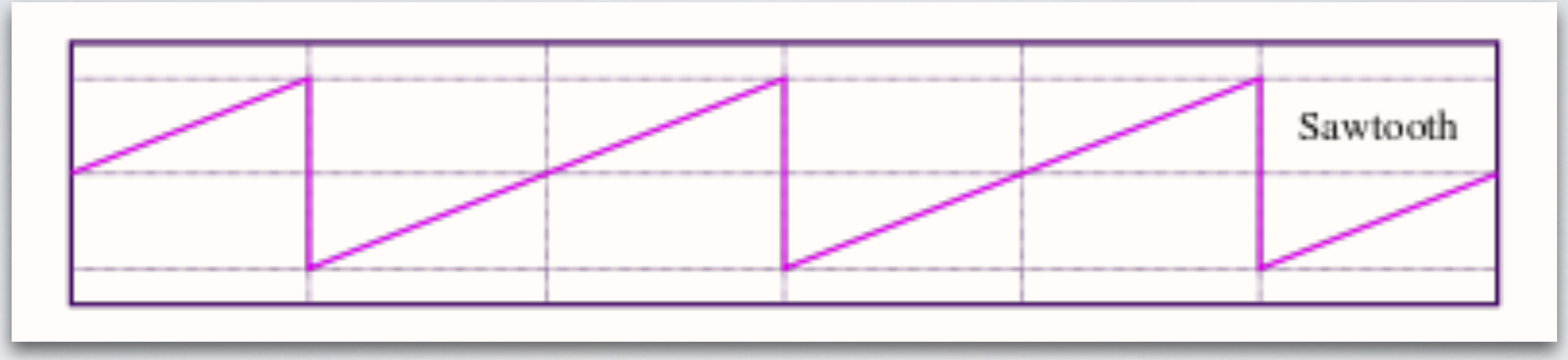


Omegatron (source)

- How do we implement this?

SAWTOOTH WAVE

- Jagged oscillation — value increases linearly then resets

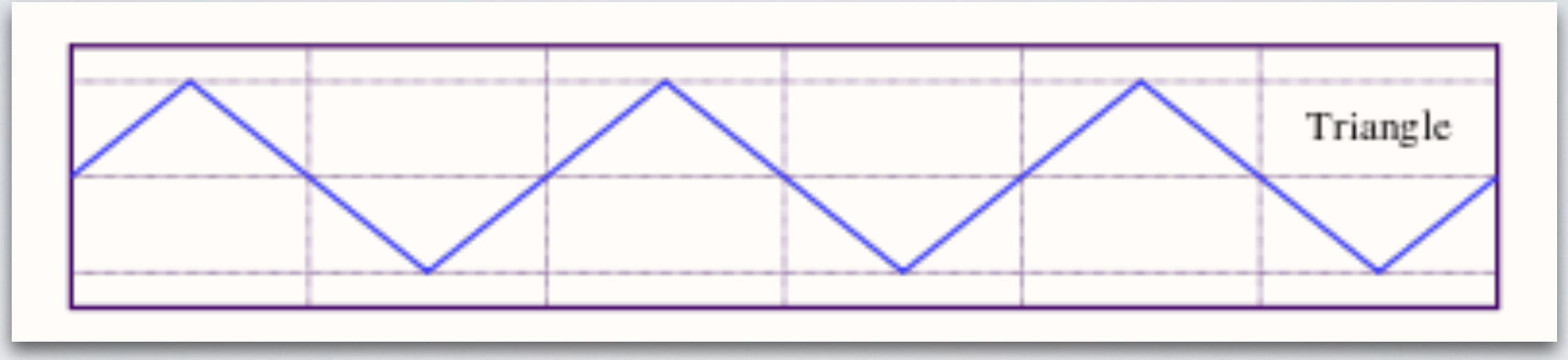


Omegatron (source)

```
float sawtooth_wave(float x, float freq, float amplitude) {
  return (x * freq - floor(x * freq)) * amplitude;
}
```

TRIANGLE WAVE

- Linear oscillation between two values

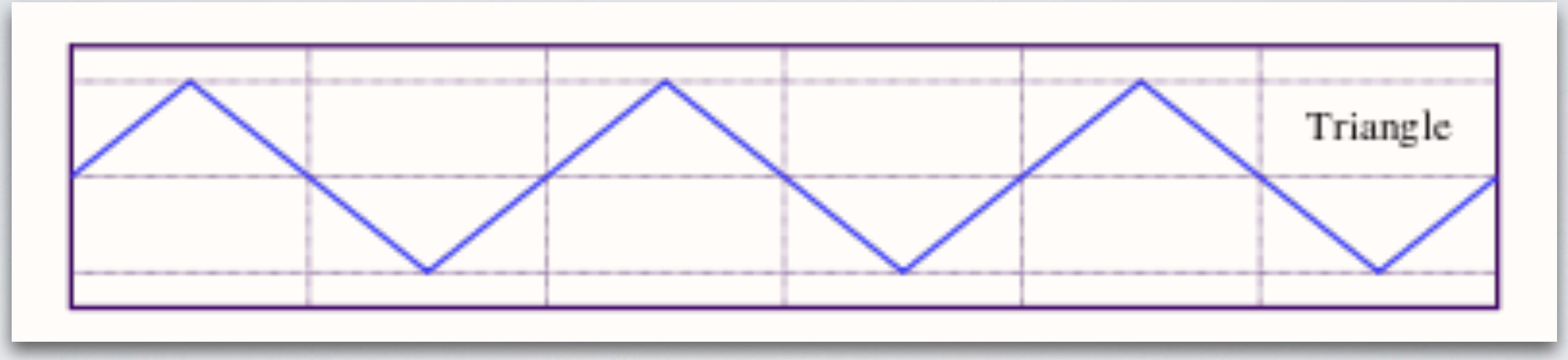


Omegatron (source)

- How do we implement this?

TRIANGLE WAVE

- Linear oscillation between two values



Omegatron (source)

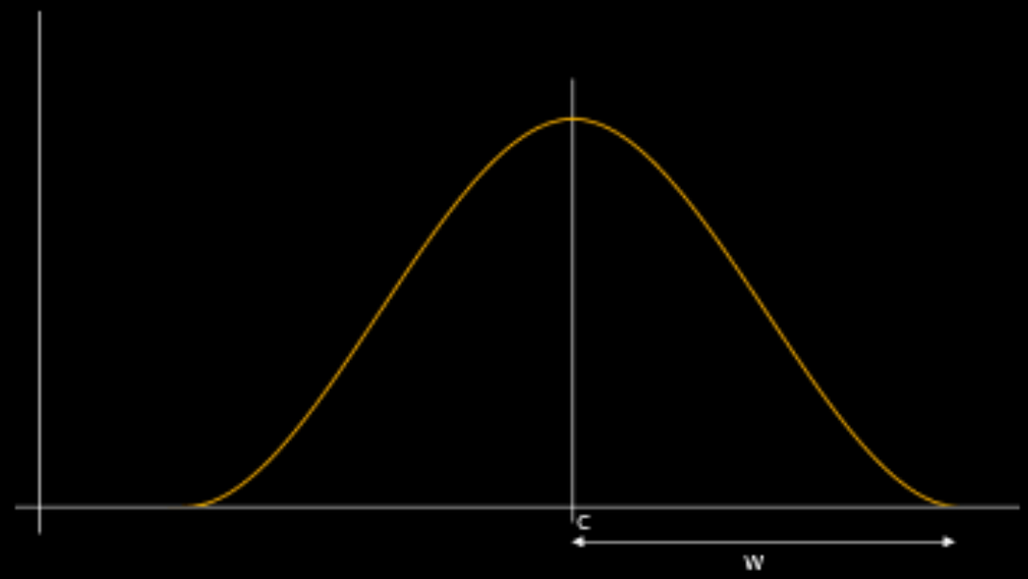
```
float triangle_wave(float x, float freq, float amplitude) {
  return fabs((x * freq) % amplitude - (0.5 * amplitude));
}
```

MISC (A LA I.Q.)

PULSE

- Mimics a gaussian shape.
- **c** controls centering, **w** controls taper length
- Useful for isolating a feature, turning something on and off,

```
float cubicPulse( float c, float w, float x )
{
    x = fabsf(x - c);
    if( x>w ) return 0.0f;
    x /= w;
    return 1.0f - x*x*(3.0f-2.0f*x);
}
```

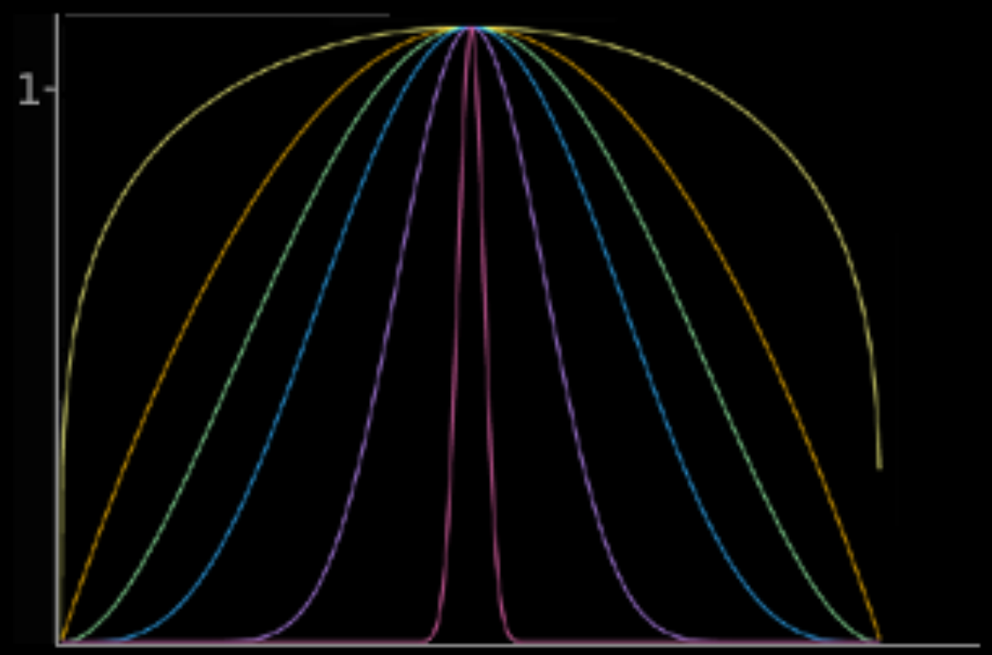


Íñigo Quílez (source)

PARABOLA

- Remaps the [0, 1] interval such that
- parabola(0) = parabola(1) = 0, parabola(1/2) = 1
- **k** controls steepness
- Similar to pulse, use for symmetric shapes or on/off.

```
float parabola( float x, float k )
{
    return powf( 4.0f*x*(1.0f-x), k );
}
```

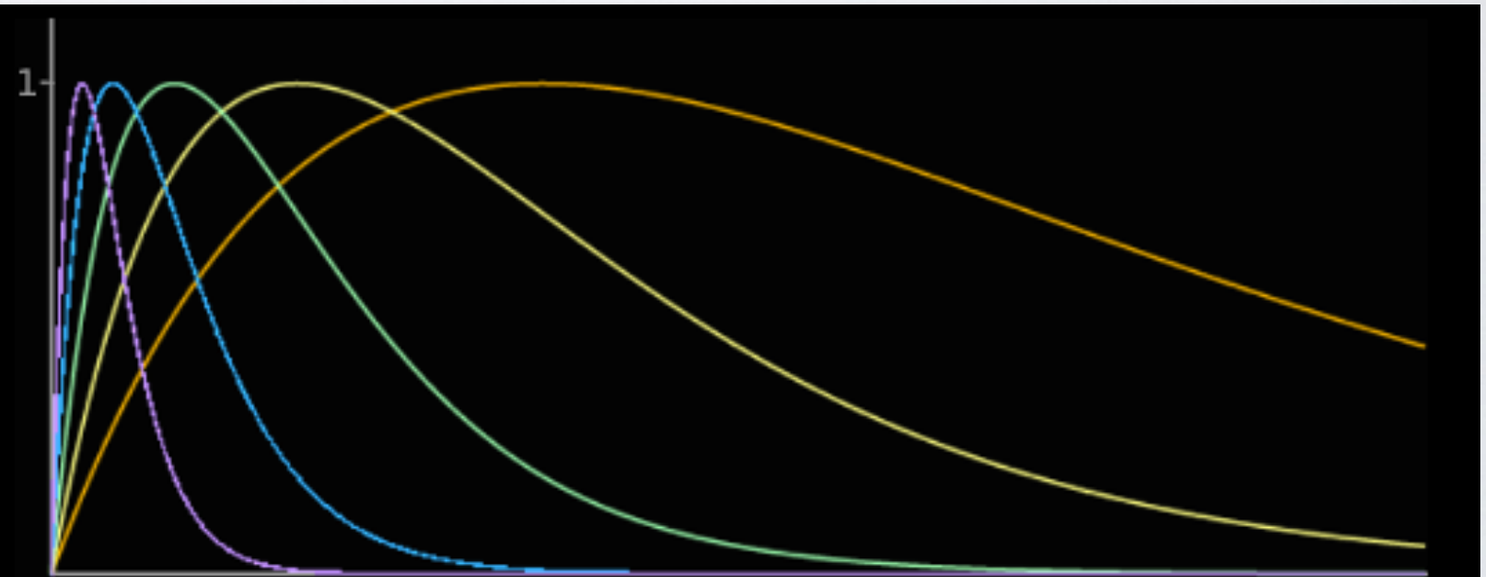


Íñigo Quílez (source)

IMPULSE

- Reach a maximum value, then gradually decay
- k controls the rate of decay

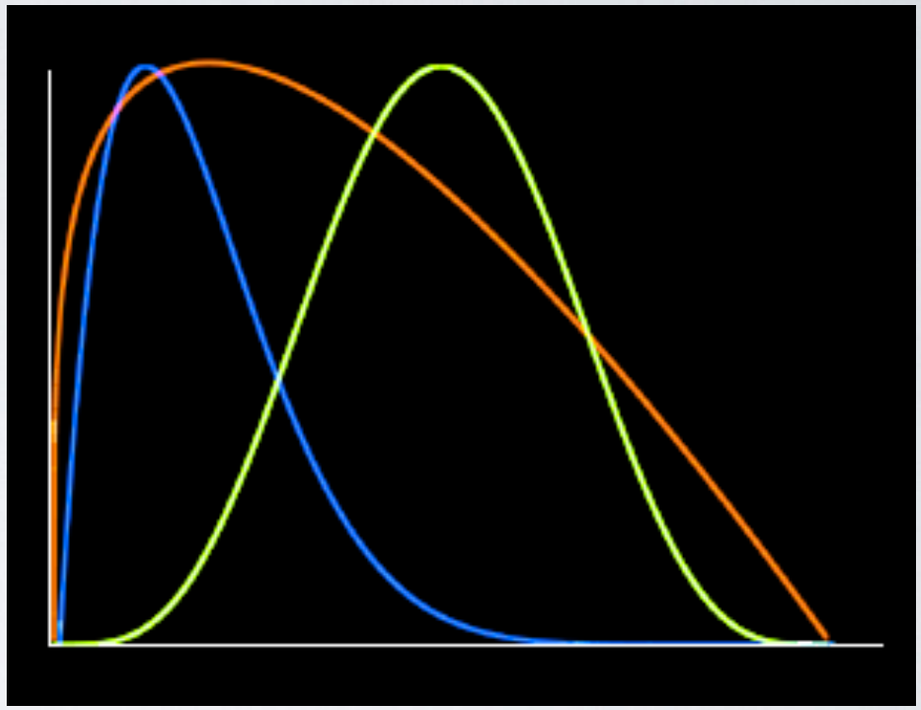
```
float impulse( float k, float x )
{
    const float h = k*x;
    return h*expf(1.0f-h);
}
```



Íñigo Quílez (source)

POWER CURVE

- Skew a pulse-like curve to one side
- a and b control the bias toward either side
- Good for creating shapes, nudging signals off-center.



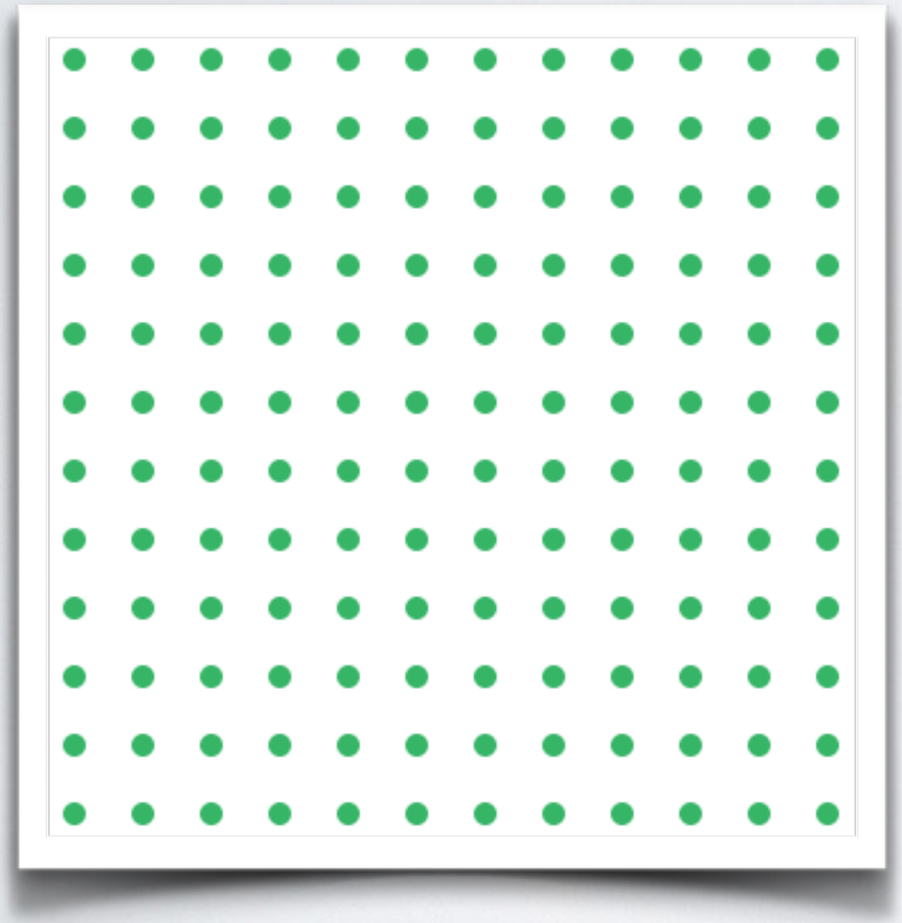
```
float pcurve( float x, float a, float b )
{
    float k = powf(a+b,a+b) / (pow(a,a)*pow(b,b));
    return k * powf( x, a ) * powf( 1.0-x, b );
}
```

Íñigo Quílez (source)

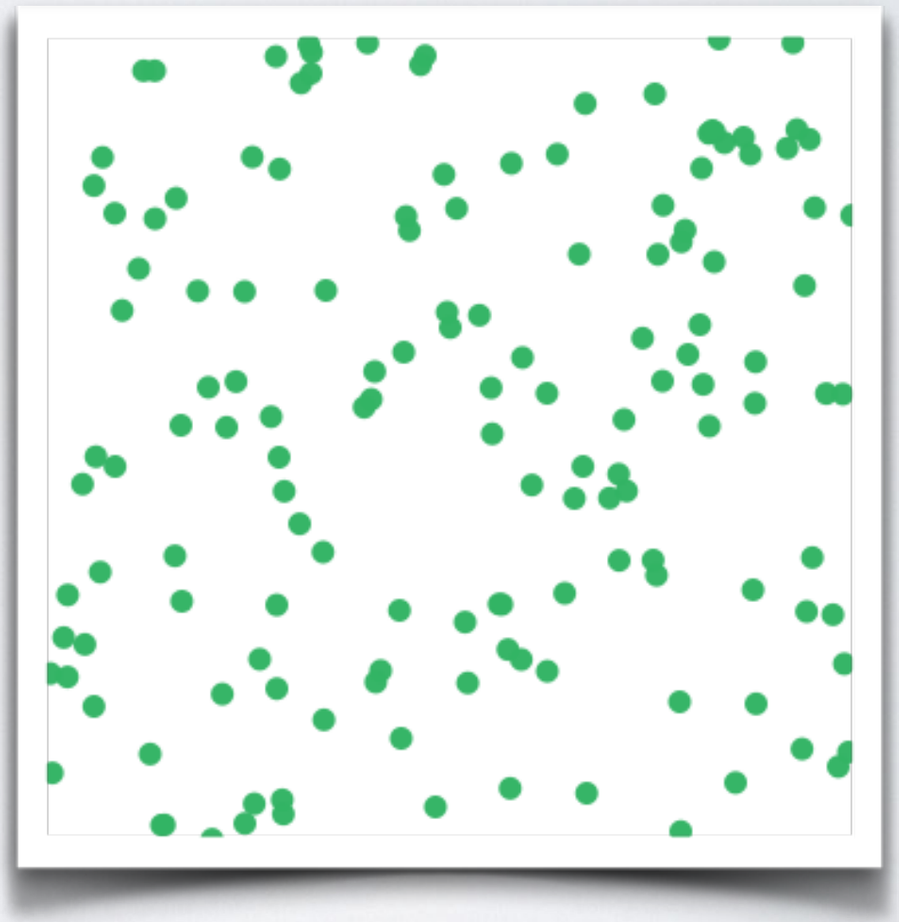
SCATTERING

RANDOM PLACEMENT

- We often want to scatter objects in a random-looking way
- But as we've discussed, pure random is usually not what we want
- Similar problem to supersampling with raytracing methods



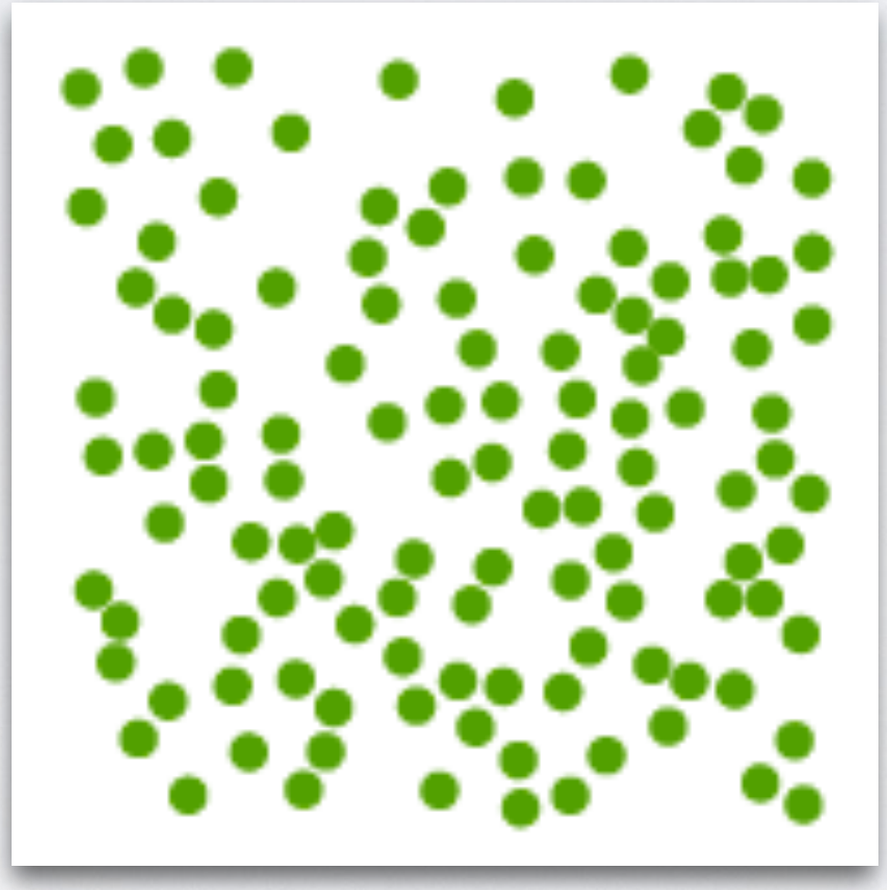
uniform



random

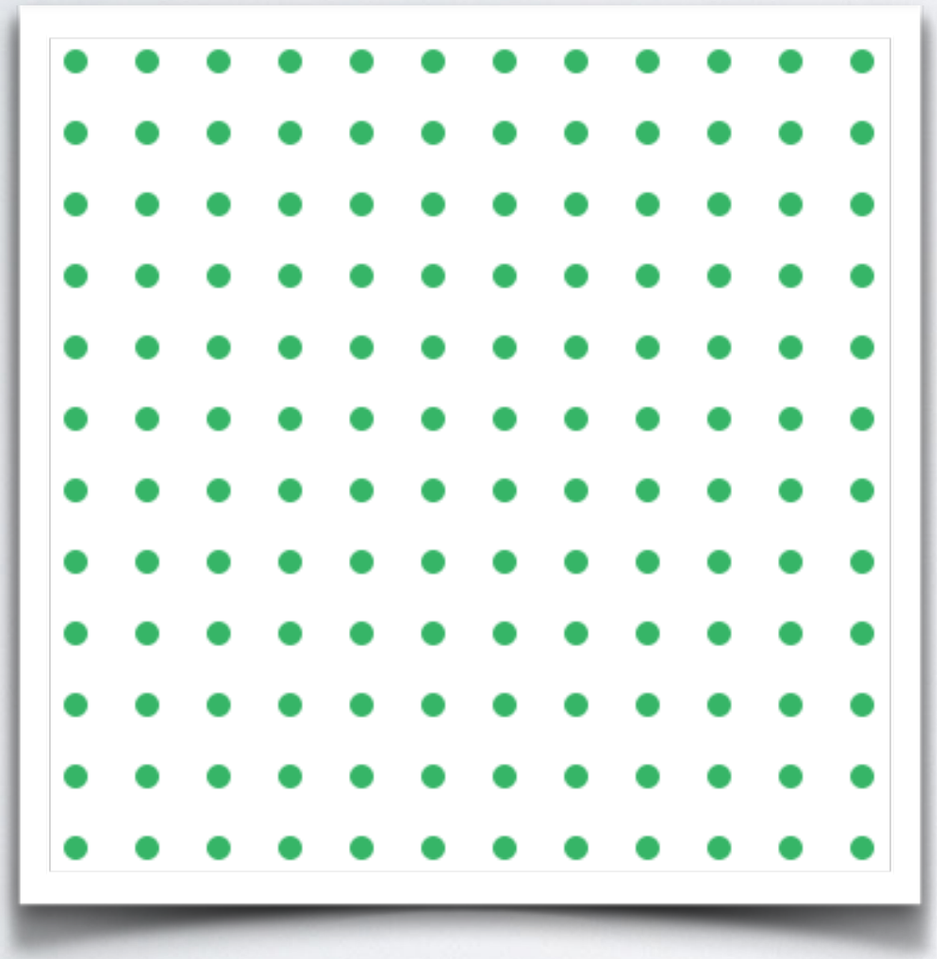
POISSON-DISK SAMPLING

- We can enforce minimum distance simply by checking for neighbors whenever we place an object
- Downside: this is expensive O(n^2) unoptimized since we have to check for neighbors.
- Optimization: use a uniform grid to put a bound on the number of checks to make O(n).

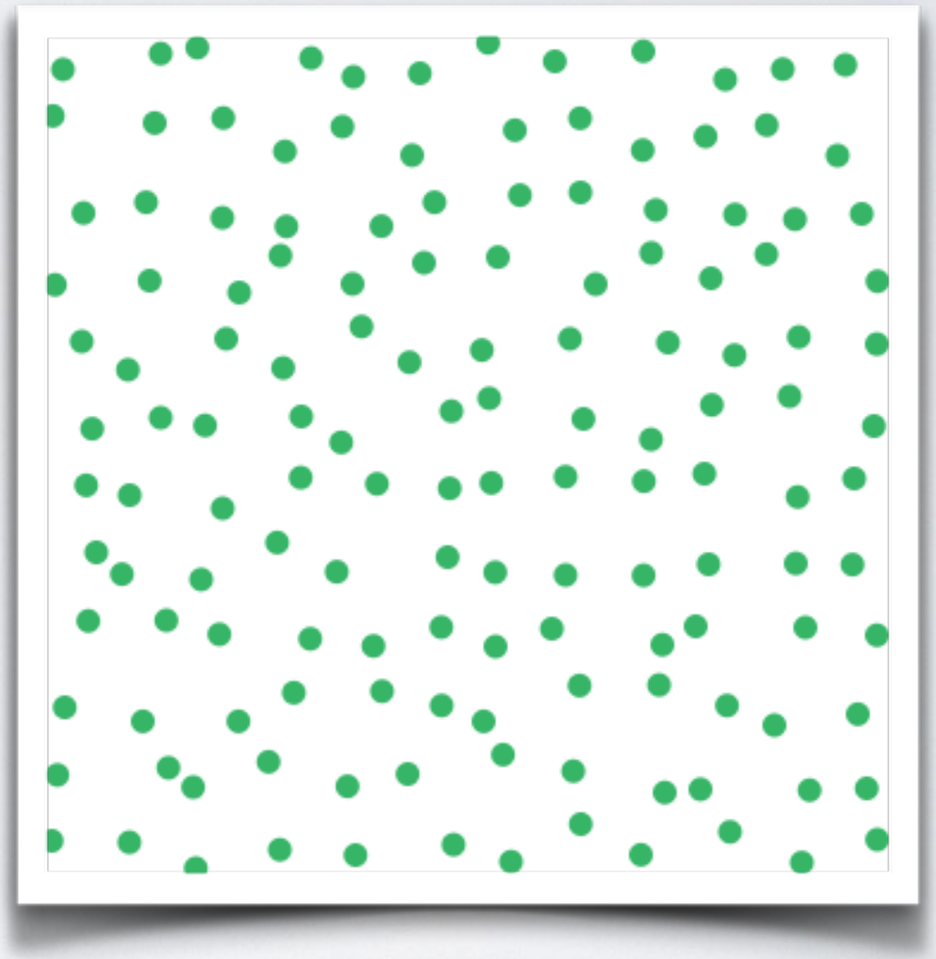


JITTER

- Cheaper solution: apply a random offset to the uniform distribution
- We may cause some collisions, but we can tune to "cheat".



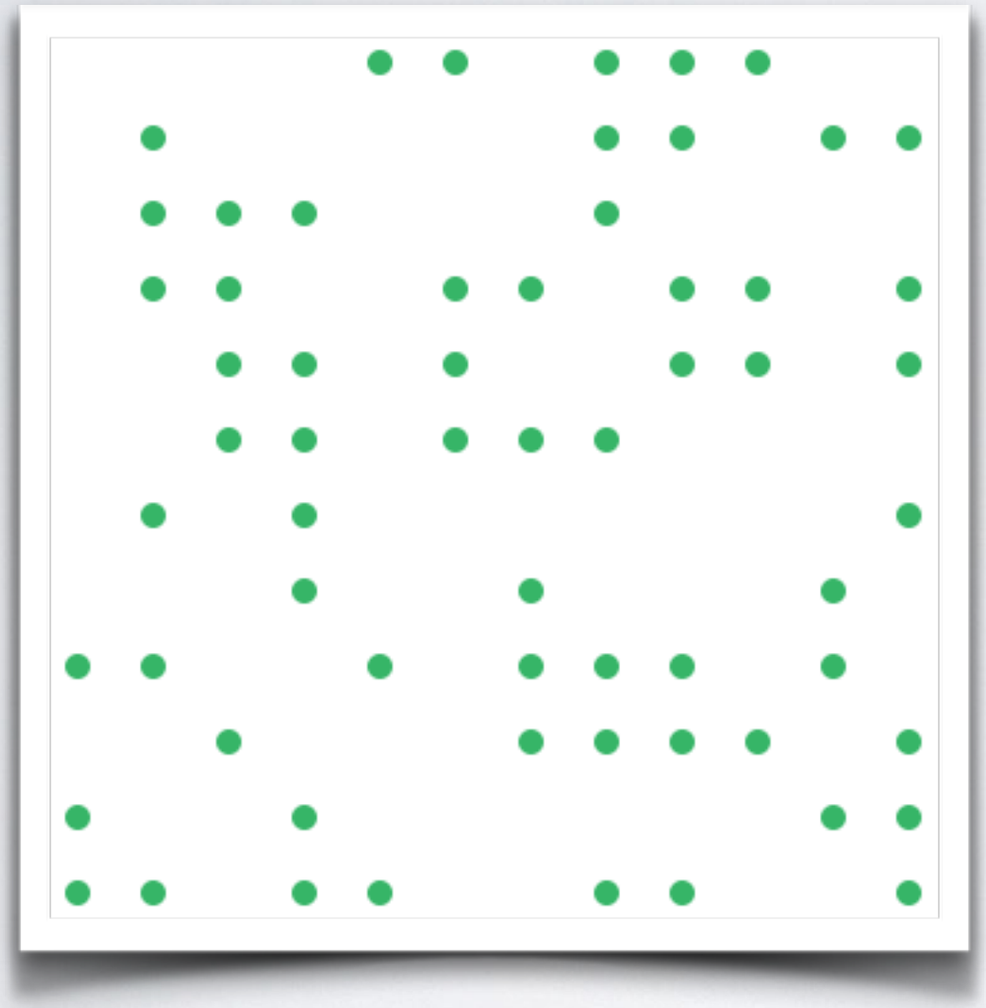
uniform



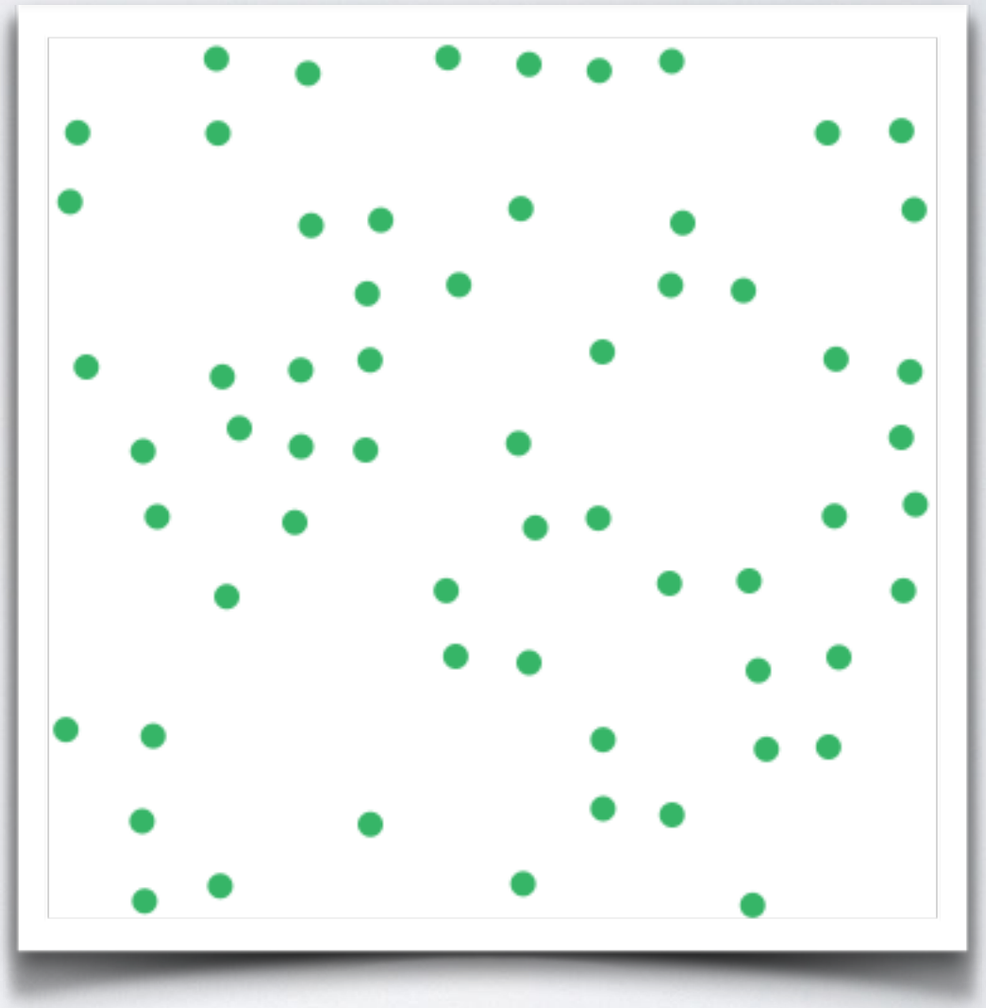
jittered

OUR FRIEND NOISE!

- We can use a threshold noise value to include or exclude objects at regularly-space, or jittered sample positions.

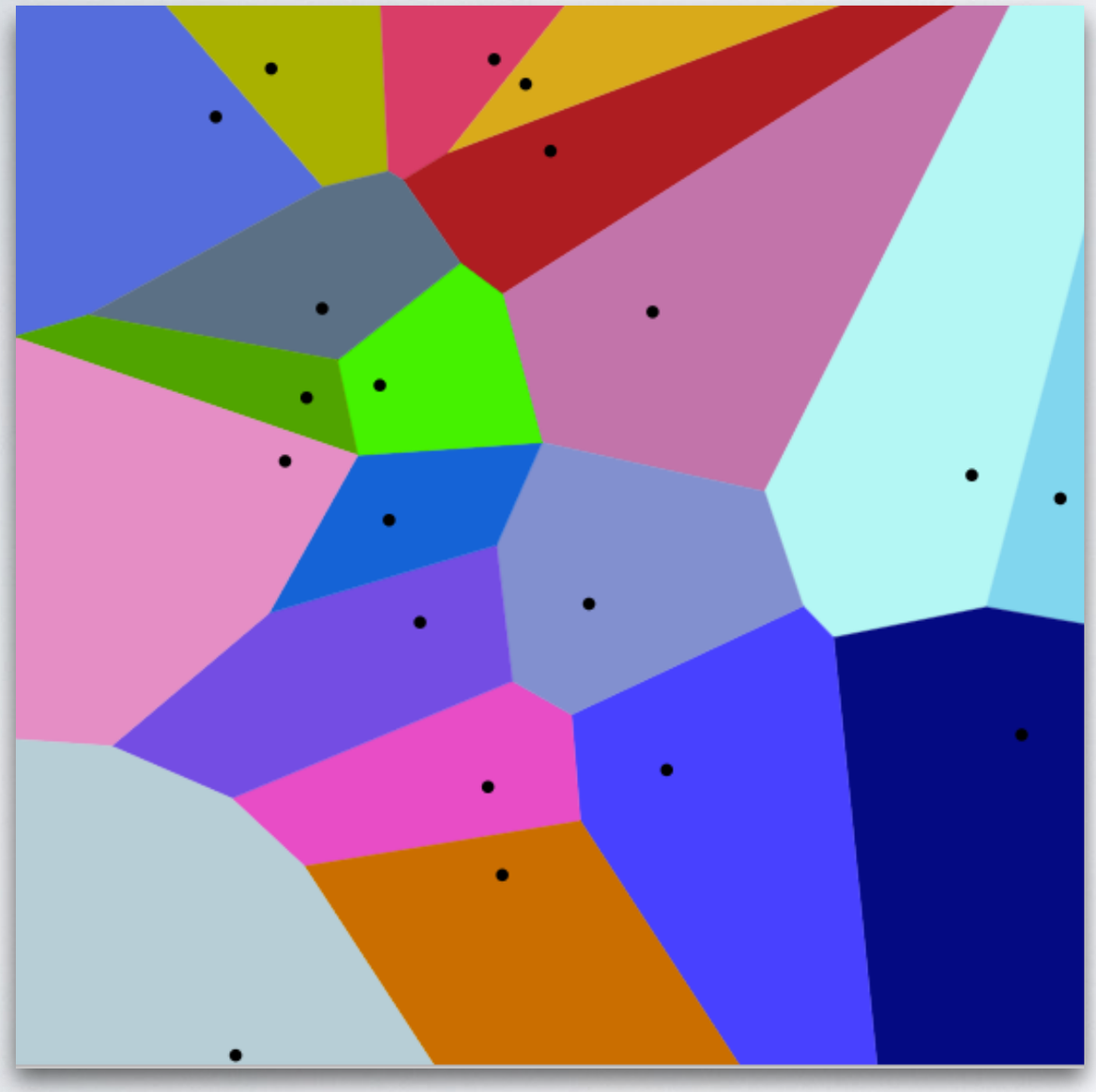


uniform w/ noise threshold



jitter w/ noise threshold

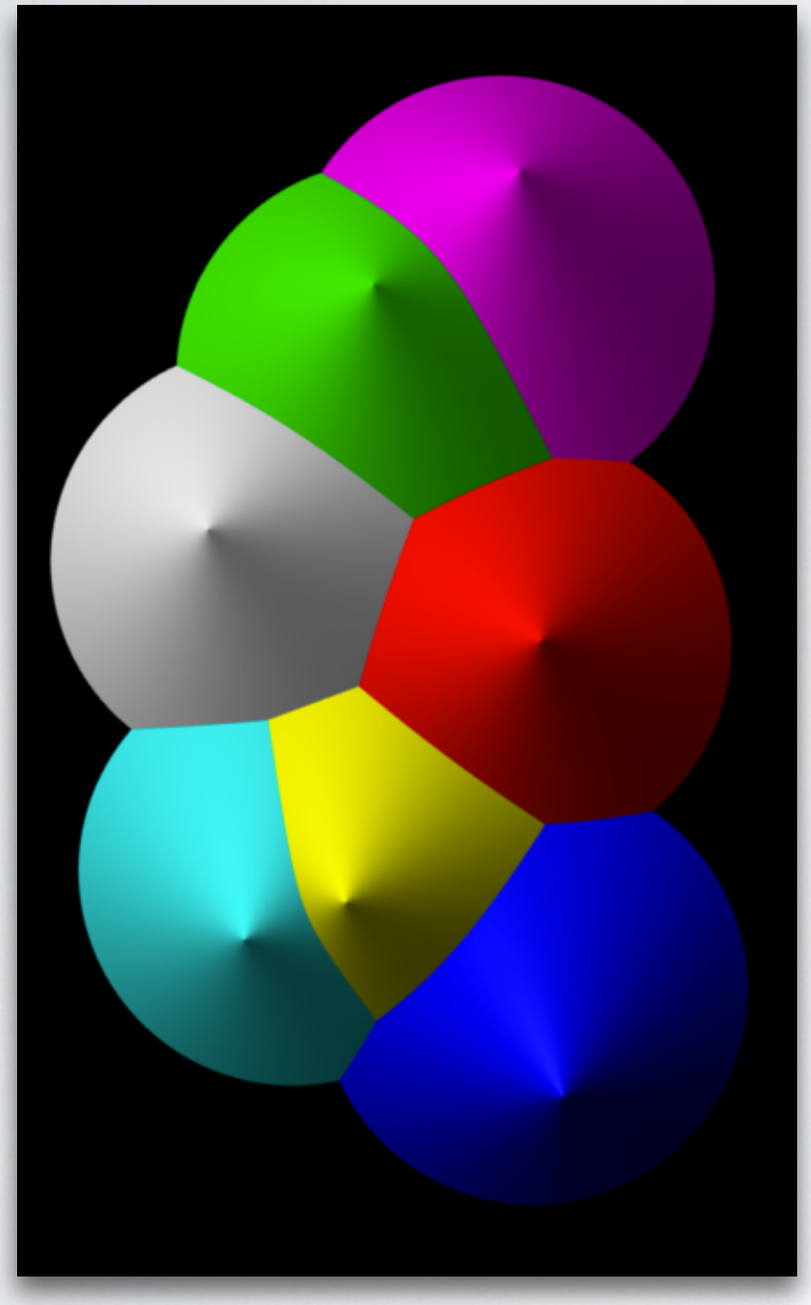
VORONOI DIAGRAMS



Balu Ertl (source)

- Partitioning of a plane and set of n points on the plane into n convex polygons, (aka cells or regions).
- Each region has a single **generating point**
- Each point in a region is guaranteed to be closer to the region's generating point than any of the other n possible generating points.
- Line segments are equidistant to two points. Nodes (corners) are equidistant to three (or more) points.
- Concept can be extended to higher dimensions.

VORONOI DIAGRAMS

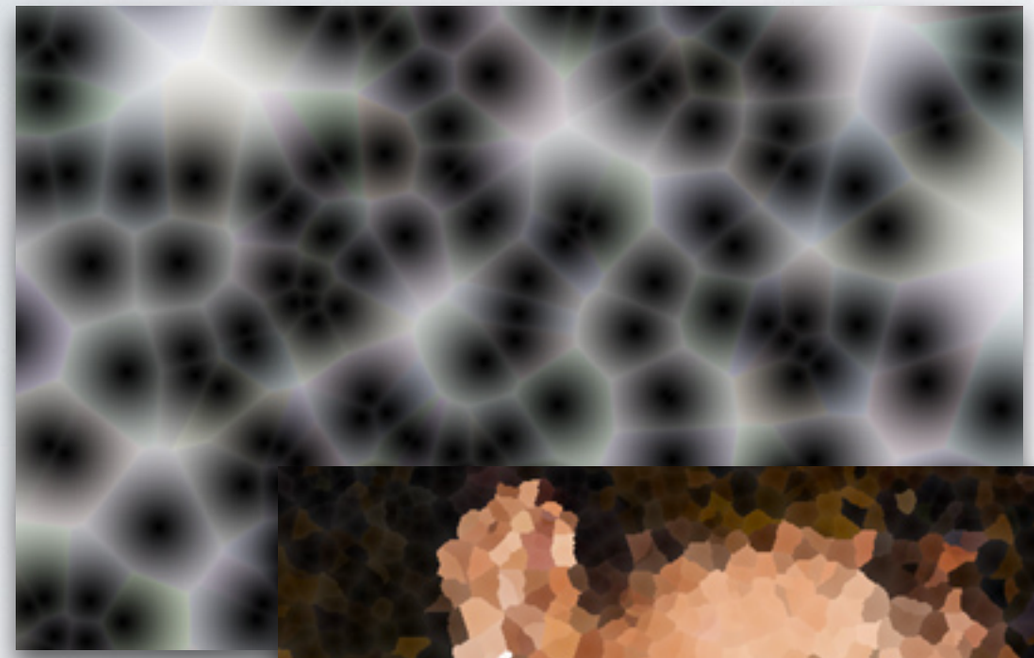


Egor Larionov (source)

- Approaches
  - Delaunay Triangulation
  - Fortune's Algorithm
  - Rendering cones… Penn example!

VORONOI APPLICATIONS

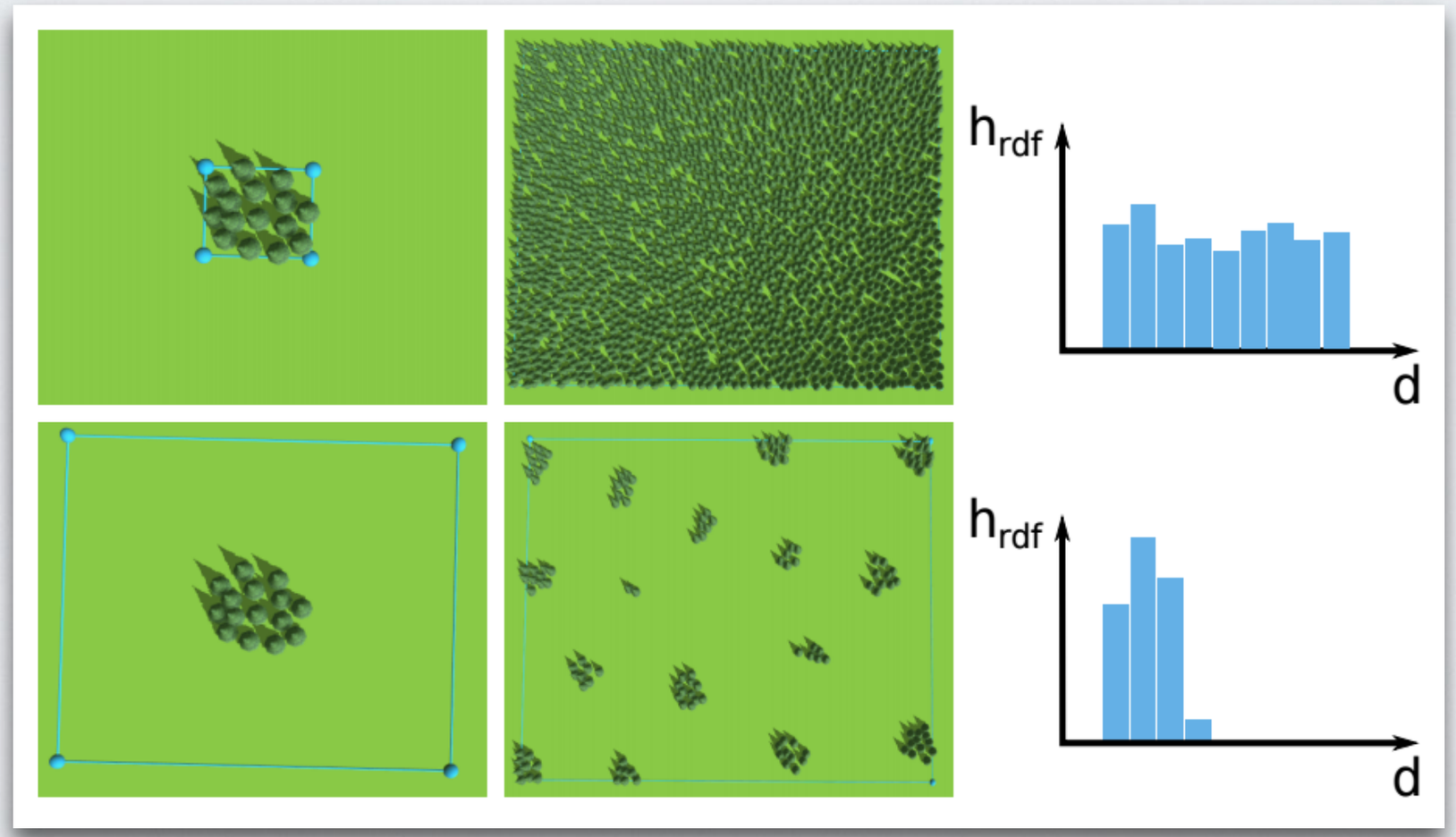
- All sorts! Cell-like patterns, mosaics, fracturing, regions of influence



Íñigo Quílez (source)

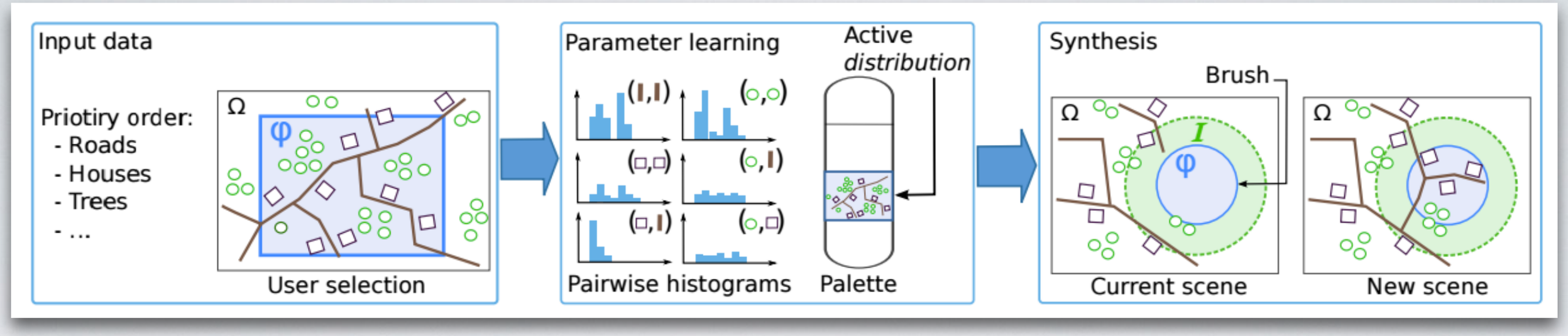
WORLDBRUSH

- Plausible distributions of objects are powerful in creating realism!
- An example-based interactive tool for synthesizing procedural virtual worlds (by. Maud Emilien, Ulysse Vimont, Marie-Paule Cani, Pierre Poulin, Bearish Benes)
- Created a "brush" for placing objects of different types using a "palette" of distributions



World brush (source)

WORLDBRUSH



World brush (source)

1. Takes input data and priority order for categories of objects.
2. Computes pairwise histograms by distance to create an "active distribution"
    - Users can create multiple distributions to create a palette.
3. In the current scene, synthesize content in brush footprint using the active distribution and considering surrounding objects as context.

WORLDBRUSH

Worldbrush: Interactive example-based synthesis of procedural virtual worlds

Arnaud Emilien [1,2], Ulysse Vimont [1], Marie-Paule Cani [1], Pierre Poulin [2], Bedrich Benes [3]

[1] University Grenoble-Alpes, CNRS (LJK), and Inria
[2] LIGUM, Dept. I.R.O., Université de Montréal
[3] Purdue University

World brush (source)

Without a lot of machine learning, how can we achieve realistic looking distributions?

IN SUMMARY

- We often want to vary an attribute through time or space (or some other value!)
- Lerp is only the beginning!
    - Many different interpolation methods represent different rates of growth
    - We can simply manipulate our **t** value
    - We can ease-in, ease-out, or both
    - Interpolation works for higher dimensions!
- Scattering and object placement
    - Lots of techniques to simulate "random" placement! Poisson-disk, jitter, noise.
    - Voronoi diagrams can create cellular images / influence fields
    - Well-designed placement makes a huge difference!

REFERENCES

- Interesting shaders
    - Smooth min and smooth max visualization
    - Example function and derivative viewer
- Explanations and demos
    - Awesome function reference page
    - Chapter on easing functions / tweening
    - Easing functions cheat sheet
    - Tool for previewing and comparing beizer curves
    - Further slides on Worldbrush
    - Great step-by-step shader guide
- Papers
    - Research survey on modeling and rigging bird wings
    - WorldBrush paper

ASSIGNMENT

Audubon (source)

- Practice using blending and easing functions to procedurally arrange feathers to create a bird wing
- Use whatever functions you want to govern placement, orientation, and size
- For extra pizzazz, experiment with a colorful shader