

Procedural Animation

Particles Trajectories

Particles Systems History

One of the first animated model in CG



[Particle systems - A Technique for Modeling a Class of Fuzzy Objects, William T. Reeves, Lucasfilm, 1983 (Star Trek II)]



[Karl Sims, Particle dreams, 1988]

Example of particle system

Free fall of spheres under gravity

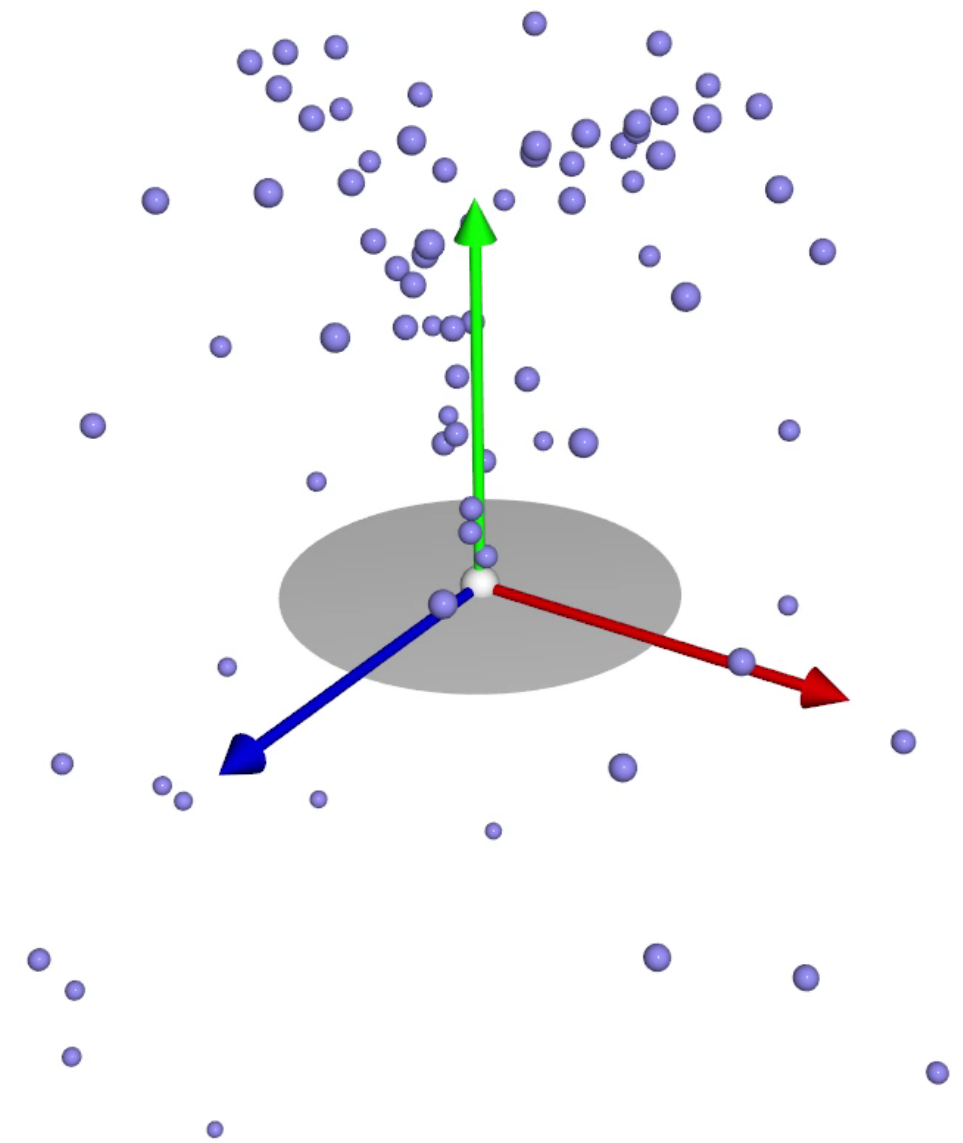
- Geometrical representation of each particle: sphere
- Equation of motion $p(t) = \frac{1}{2}gt^2 + v_0t + p_0$
- Initial position and speed may be placed at random position
- Each particle may have a different life time

Q. What are the parameters used for p_0 and v_0 in this example ?

$p_0 = \dots$

$v_0 = \dots$

Note: Coordinate system $(x, y, z) = (\text{red}, \text{green}, \text{blue})$.



Bouncing spheres

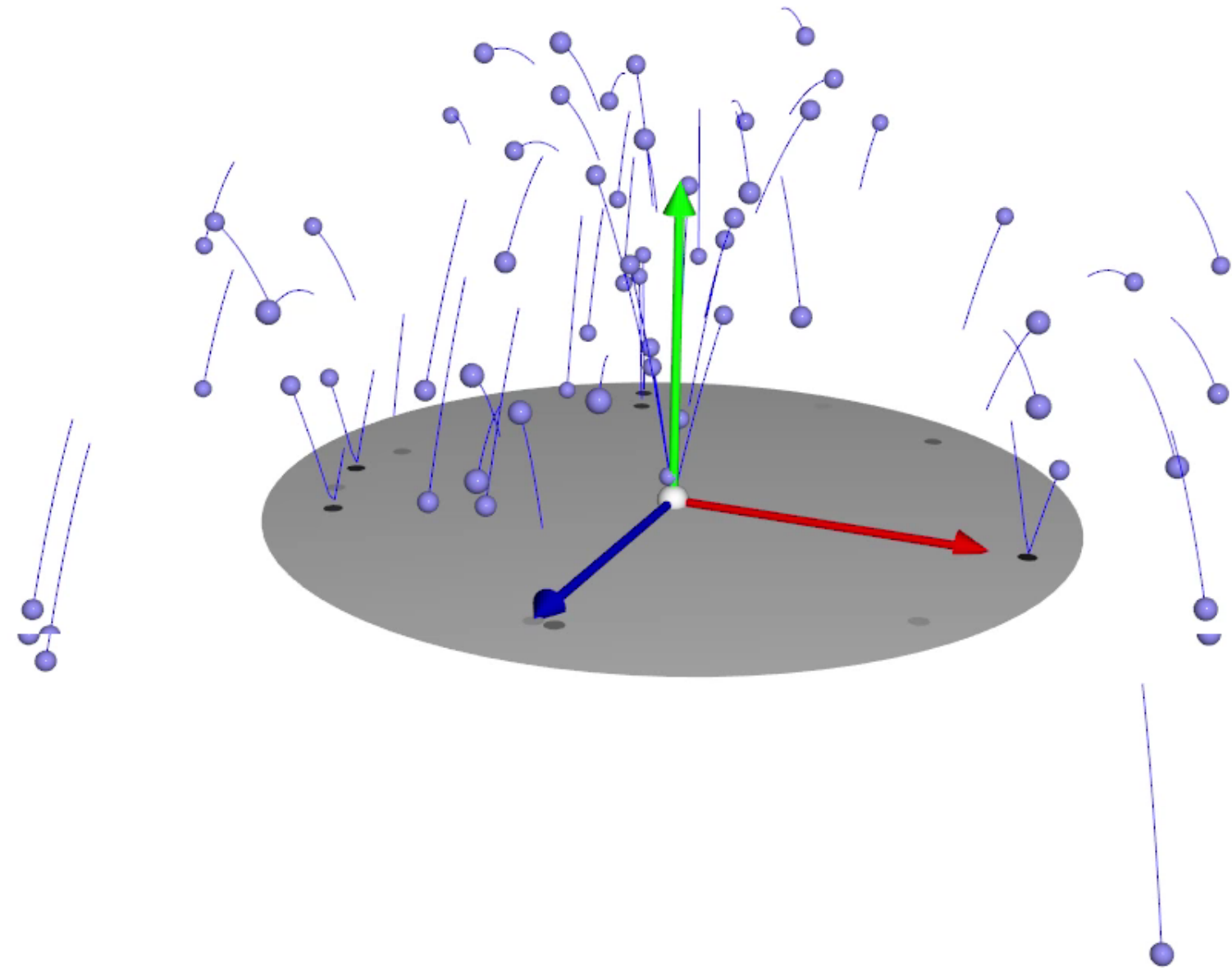
Q. What is the equation of motion ?
(taking into account the bouncing)

$$p(t) = \dots$$

Help:

- Consider a particle emitted at time $t = 0$
- At what time t_i , the particle touches the ground ?
- What is the new speed after impact ?
- Express the complete equation of trajectory
(piecewise definition)

Bonus: How the "impact shadow" effect
is computed ?

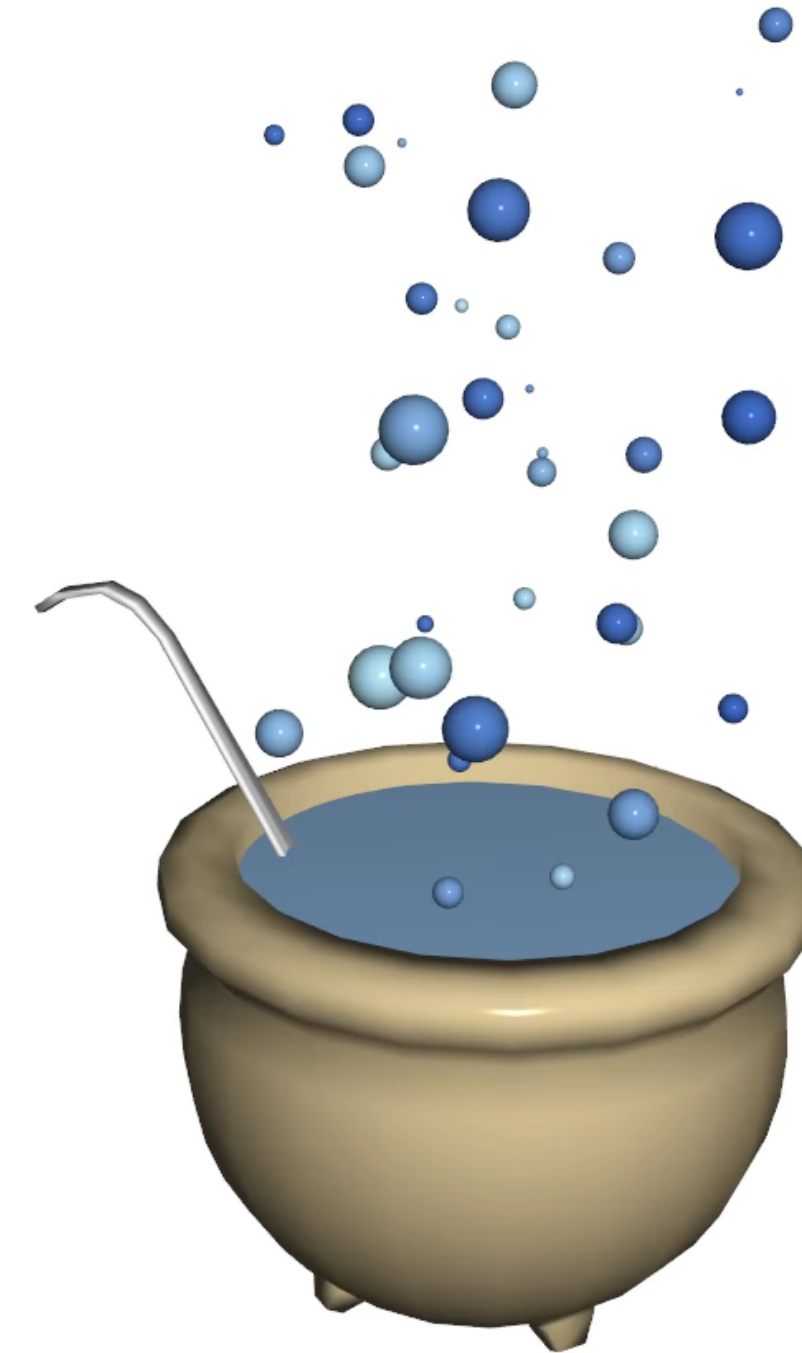


Arbitrary motions

Trajectories are not restricted to physics-based equations

Q. *What are the parameters (and initial value) associated to each particle ?*

- position = ...
- radius = rand([0,0.1])
- ...
- *What is the equation of motion of a particle ?*
 - $p(t) = \dots$



Billboards, Impostors, Sprites

Particles can be displayed as small images/thumbnails (instead of simple sphere)

In practice

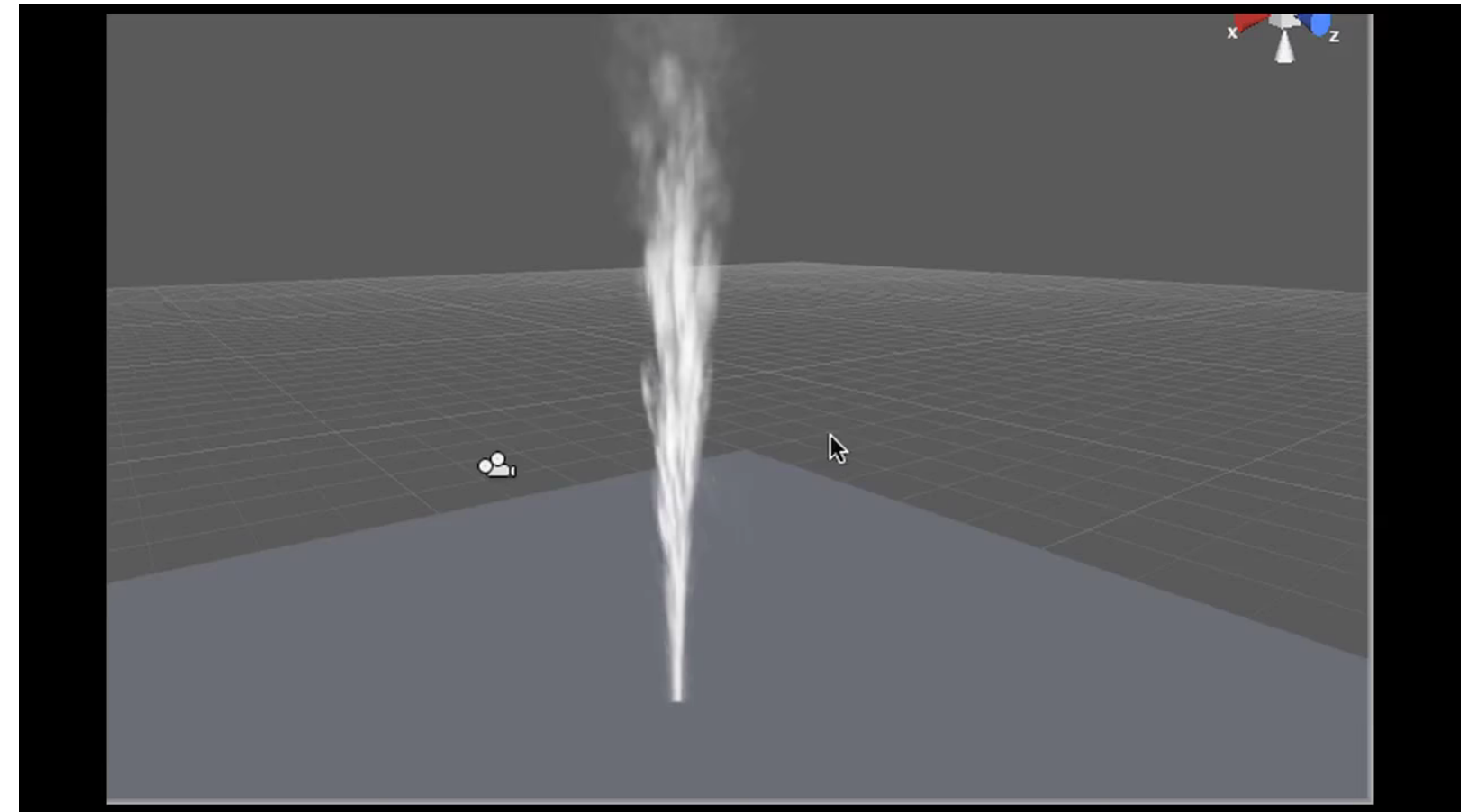
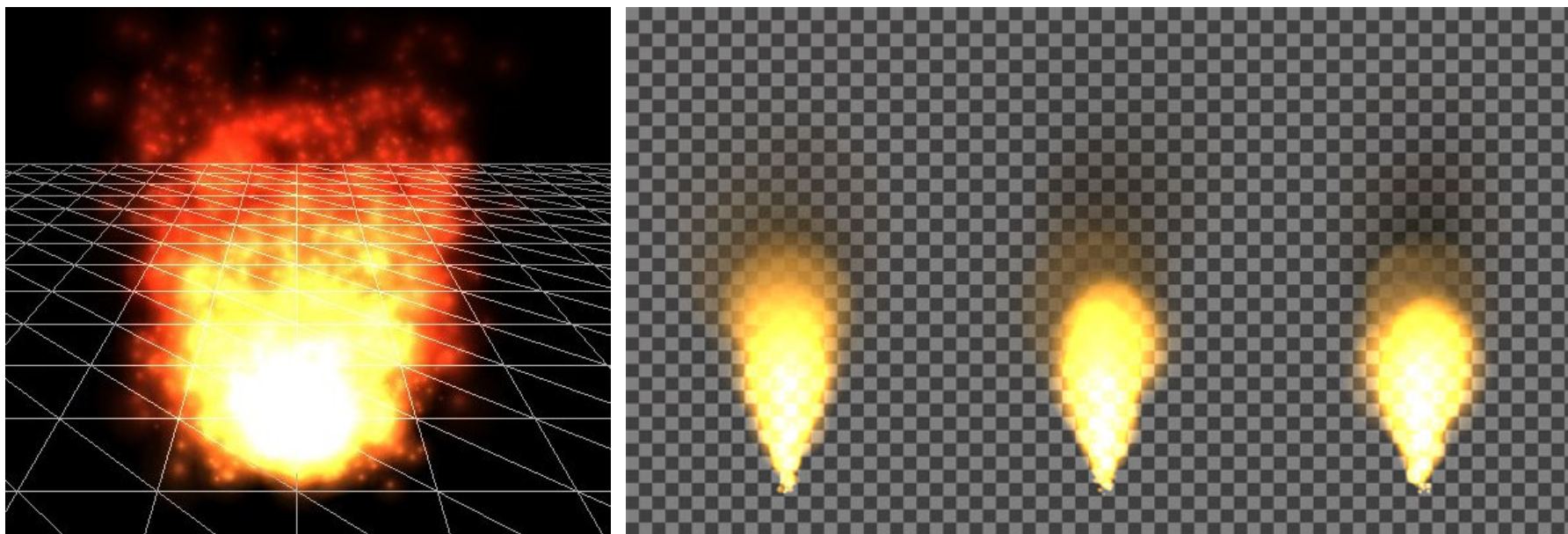
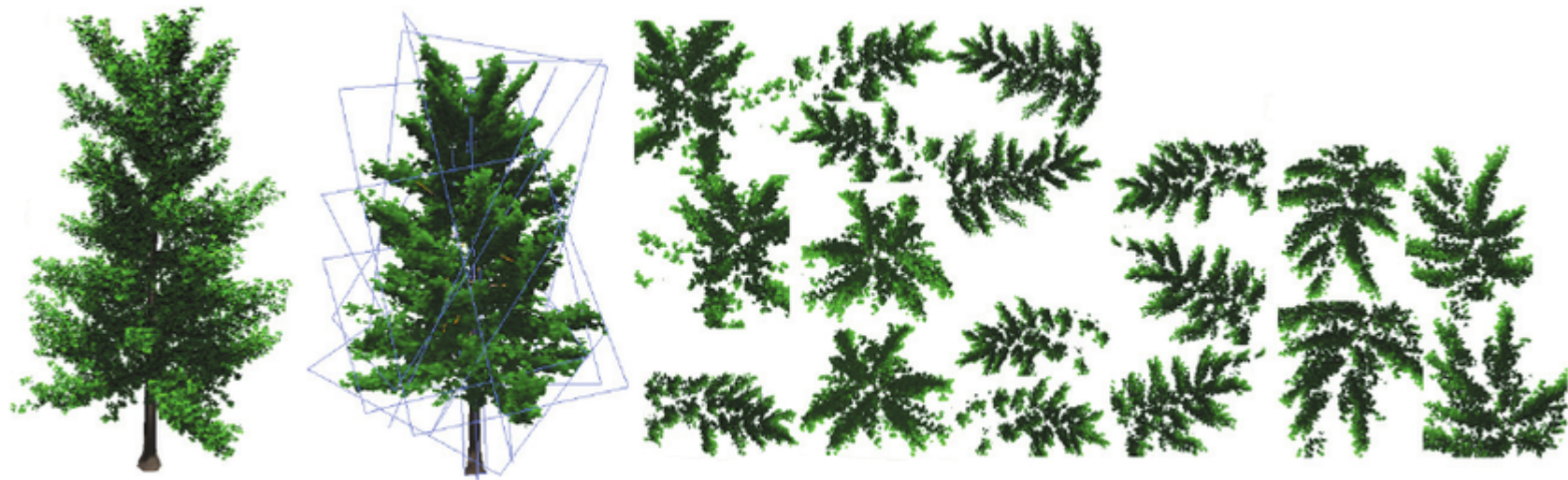
- Each particle is displayed as a quadrangle
- A texture is mapped on the quad
- The texture can contain transparency : quad geometry is invisible
 - Fake a more complex geometry
- The quad can be facing the camera at all time (billboard)
- The texture can be animated (sprite)
- Texture can be adapted to the point of view (impostors)



Usage of billboards

Large use of billboards for complex models

Vegetation, fire, smoke, etc.



Use case in production

How to model the *horse heads made of water* ?



The Lord of the Rings

Use case in production

How to model the *horse heads made of water* ?



The Lord of the Rings

Use case in production

Full Making-Of



The Lord of the Rings

Implementation - transparency in OpenGL

Transparency is not straightforward using projection/rasterization rendering.

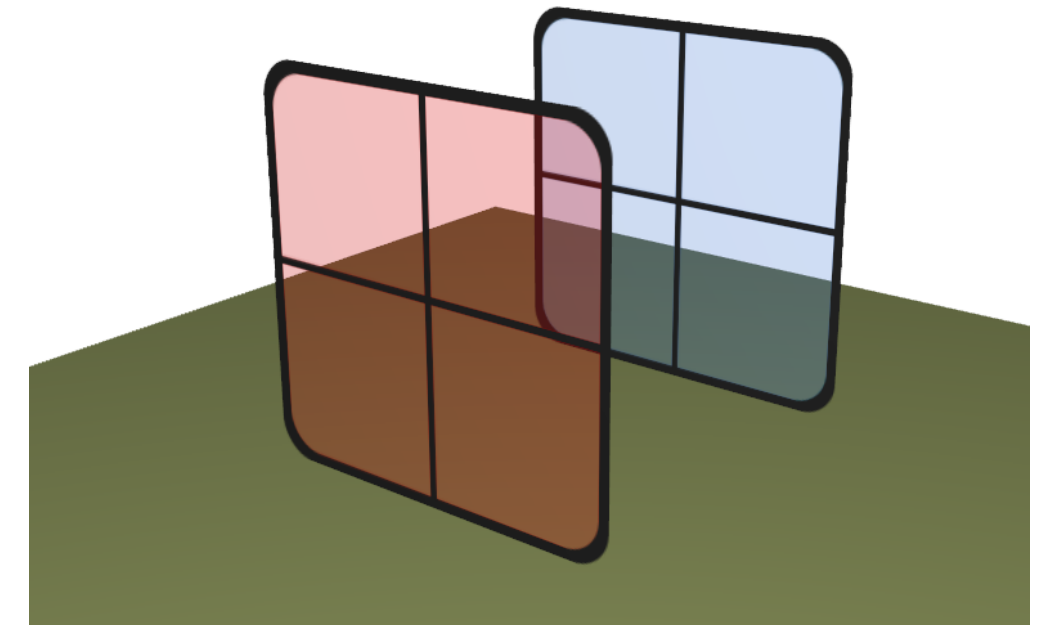
No perfect solution

⇒ Possibility to blend color for each pixel using α channel (rgba)

Need to be explicitly activated in OpenGL

```
glEnable(GL_BLEND);  
glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);  
... Draw Calls ...  
glDisable(GL_BLEND);
```

- Various blending are possible for multiple effects (see [glBlendFunc](#))
- For transparency: (*GL_SRC_ALPHA*, *GL_ONE_MINUS_SRC_ALPHA*)
⇒ $\text{newColor} = \alpha \text{ currentColor} + (1 - \alpha) \text{ previousColor}$



Important note:

The **order** of the draw calls is important (previousColor/currentColor)

The depth-buffer cannot handle the notion of transparency

Implementation - transparency in OpenGL

General Process:

- 1. Draw all **non-transparent** objects
- 2.a Activate color blending, deactivate depth buffer write
- 2.b **Sort** transparent objects by depth to the camera
$$p_{proj} = Proj \times ModelView \times p$$
$$z_{depth} = p_{proj}.z / p_{proj}.w$$
- 3. **Draw transparent objects** from furthest to closest

```
... Draw all non-transparent objects ...
glEnable(GL_BLEND); // Color Blending
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glDepthMask(false); // do not write on depth buffer
... Sort transparent objects by depth ...
... Draw all transparent objects from furthest to closest ...
glDisable(GL_BLEND);
// do not forget to re-activate depth buffer write
glDepthMask(true);
```

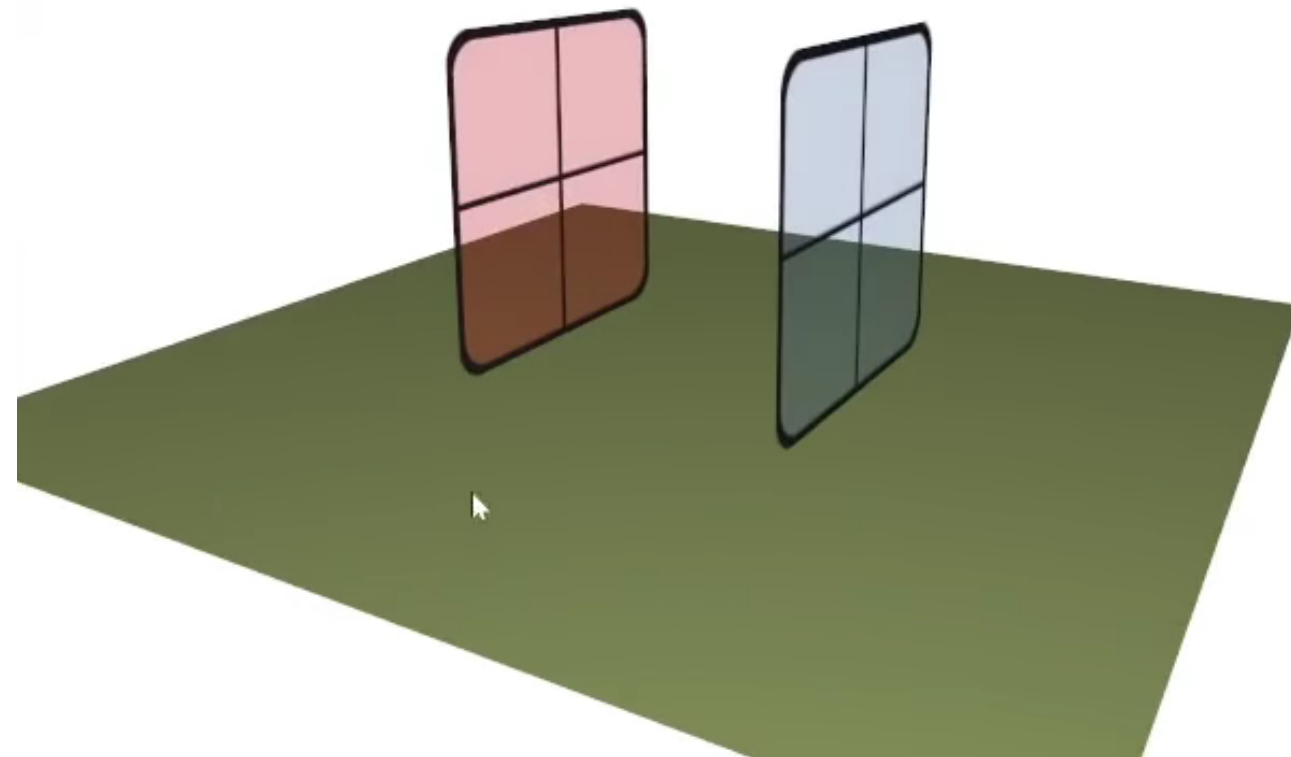
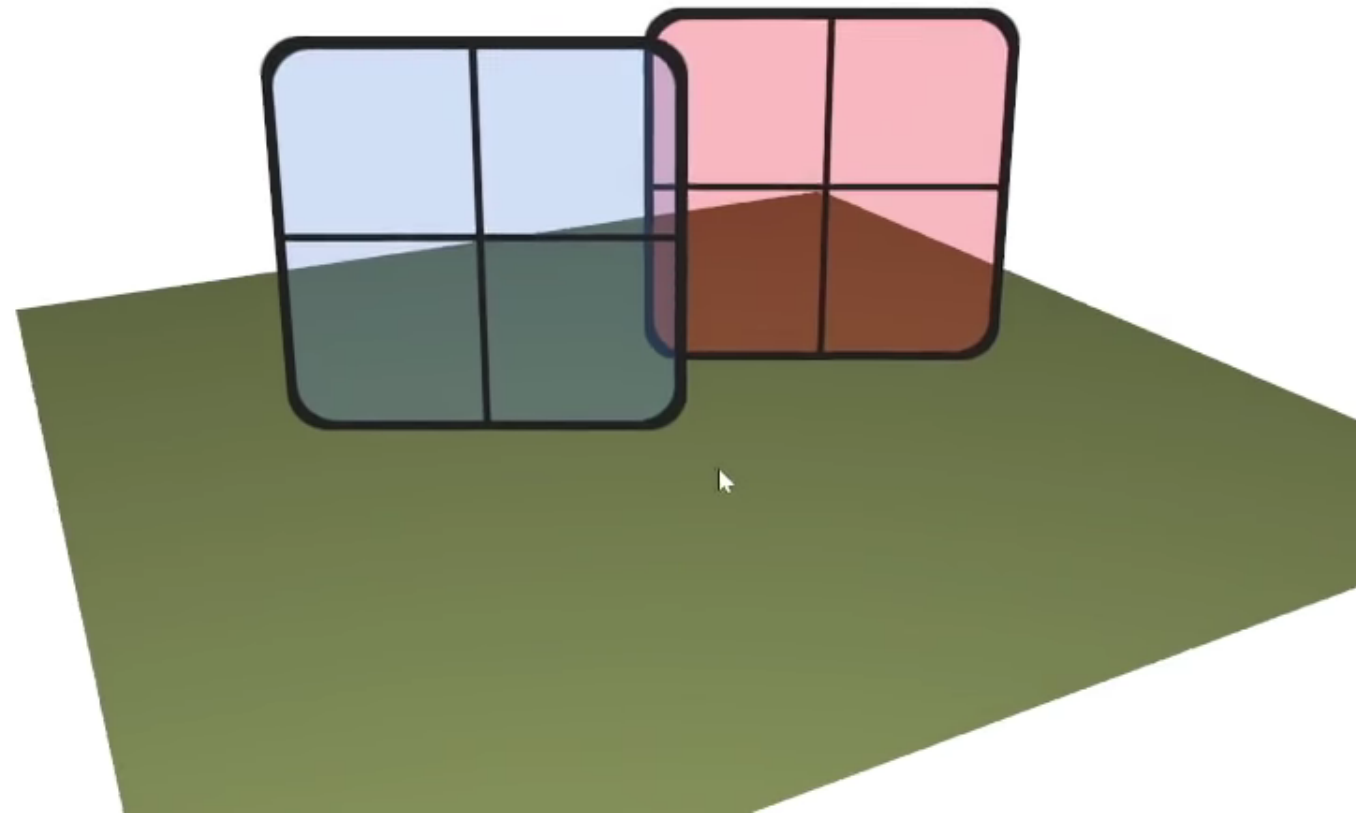


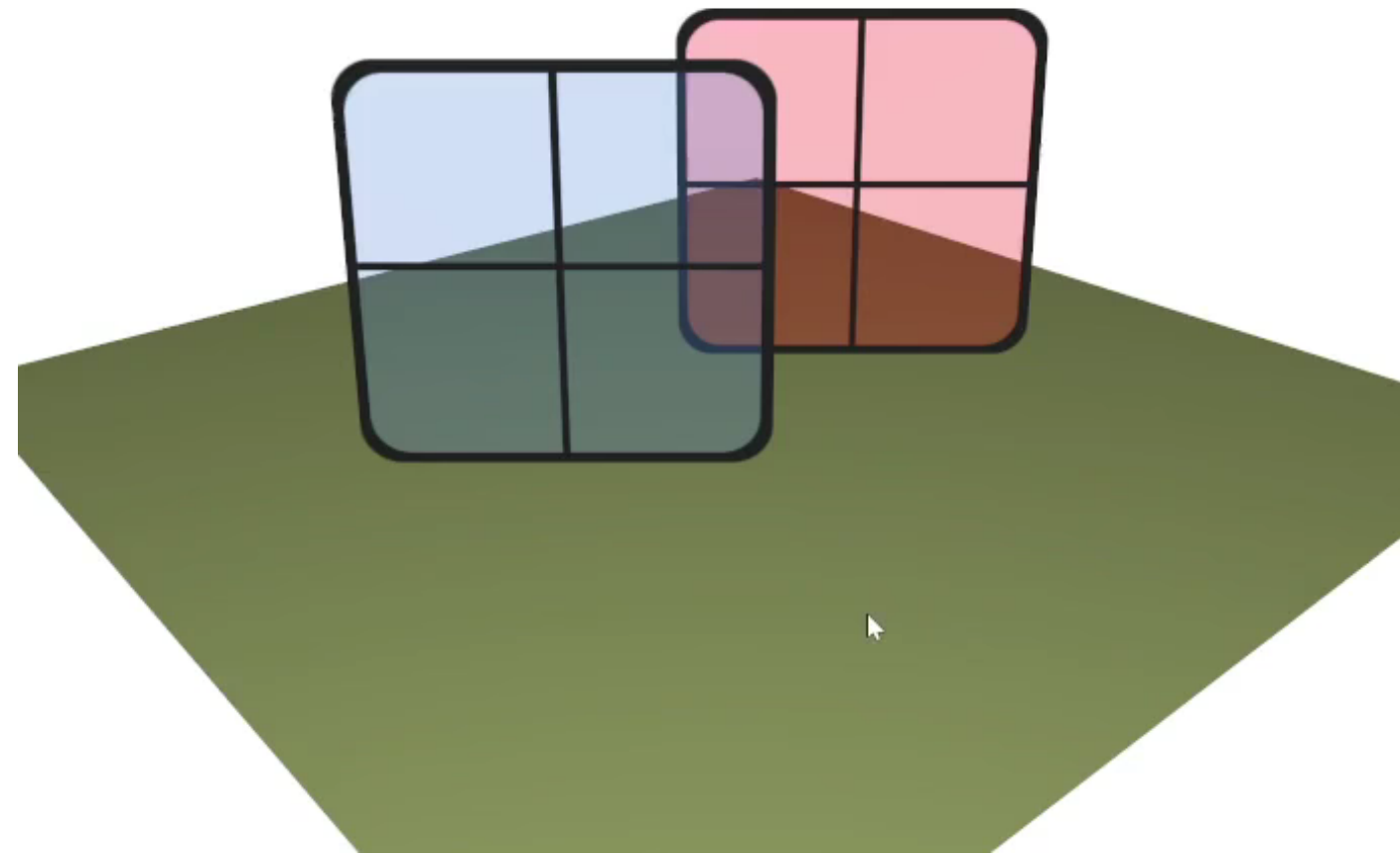
Image Blending - common errors

Keeping depth-buffer write activated
(without sorting)



Strong artifact, cannot be used

No sorting, depth buffer write de-activated



Note:

Can be OK for blurry sprites (like smokes)
where element order is hard to distinguish.
Avoids the cost of the sorting.

Implementation - transparency in OpenGL

For impostors shape with fully opaque/transparent fragment:

discard can be used on transparent fragments

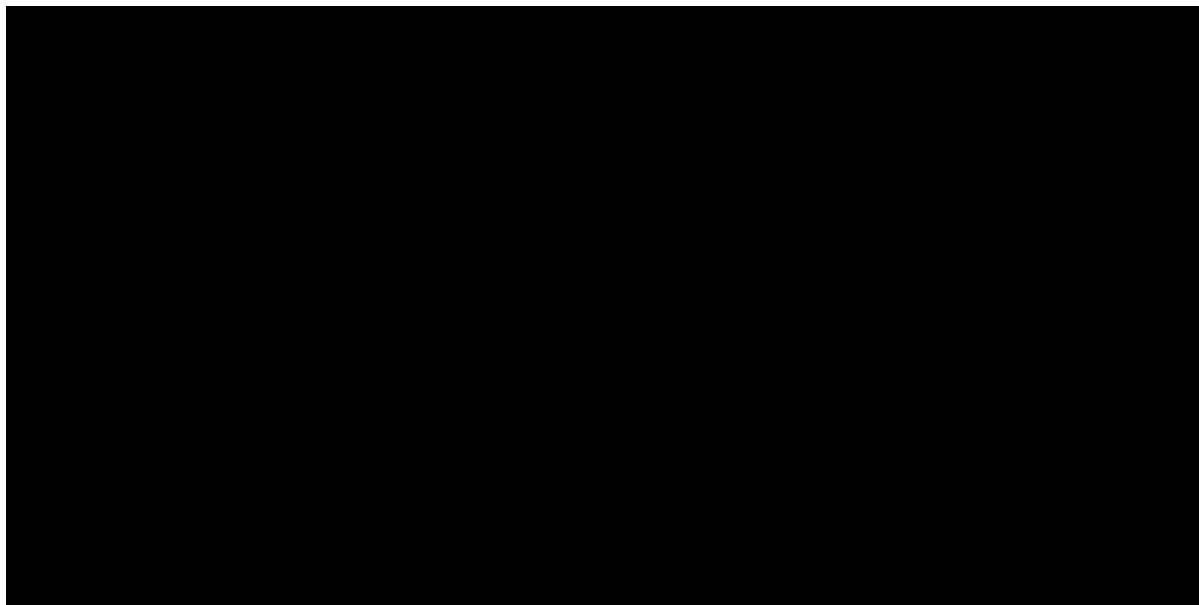
discard = Stop fragment shader execution (nothing is written)

(+) Generic, standard display

no need to sort shapes, depth-buffer works

(-) Possible aliasing artifacts

(-) Cannot be used on semi-transparent objects



In fragment shader:

```
float limit = 0.6;  
if( texture.a < limit) {  
    discard;  
}
```

Instancing

In particle system we need to display lots of similar shapes

Multiple draw calls ($N > 1000$) can be costly

each draw call requires synchronization b/w CPU-GPU

```
// Slow for large N
for(int k=0; k<N; ++k)
    someOperator(k)
    draw(shape) // ... glDrawElements(...)
```

Solution: Use Instancing

glDrawArrays → [glDrawArraysInstanced](#)

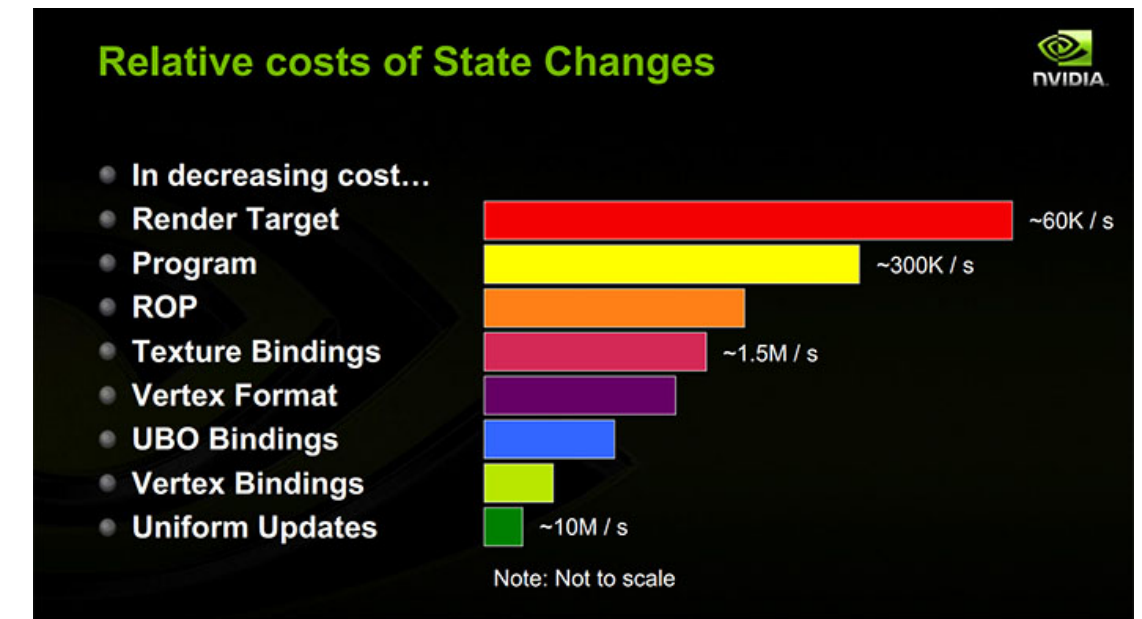
glDrawElements → [glDrawElementsInstanced](#)

In the C++ code

```
// 1 unique call, N is a parameter
drawInstanced(shape, N) // ... glDrawElementsInstanced(...,N)
// No more explicit loop
```

In the Shader:

```
// ...
someOperator(gl_InstanceID)
// ...
```



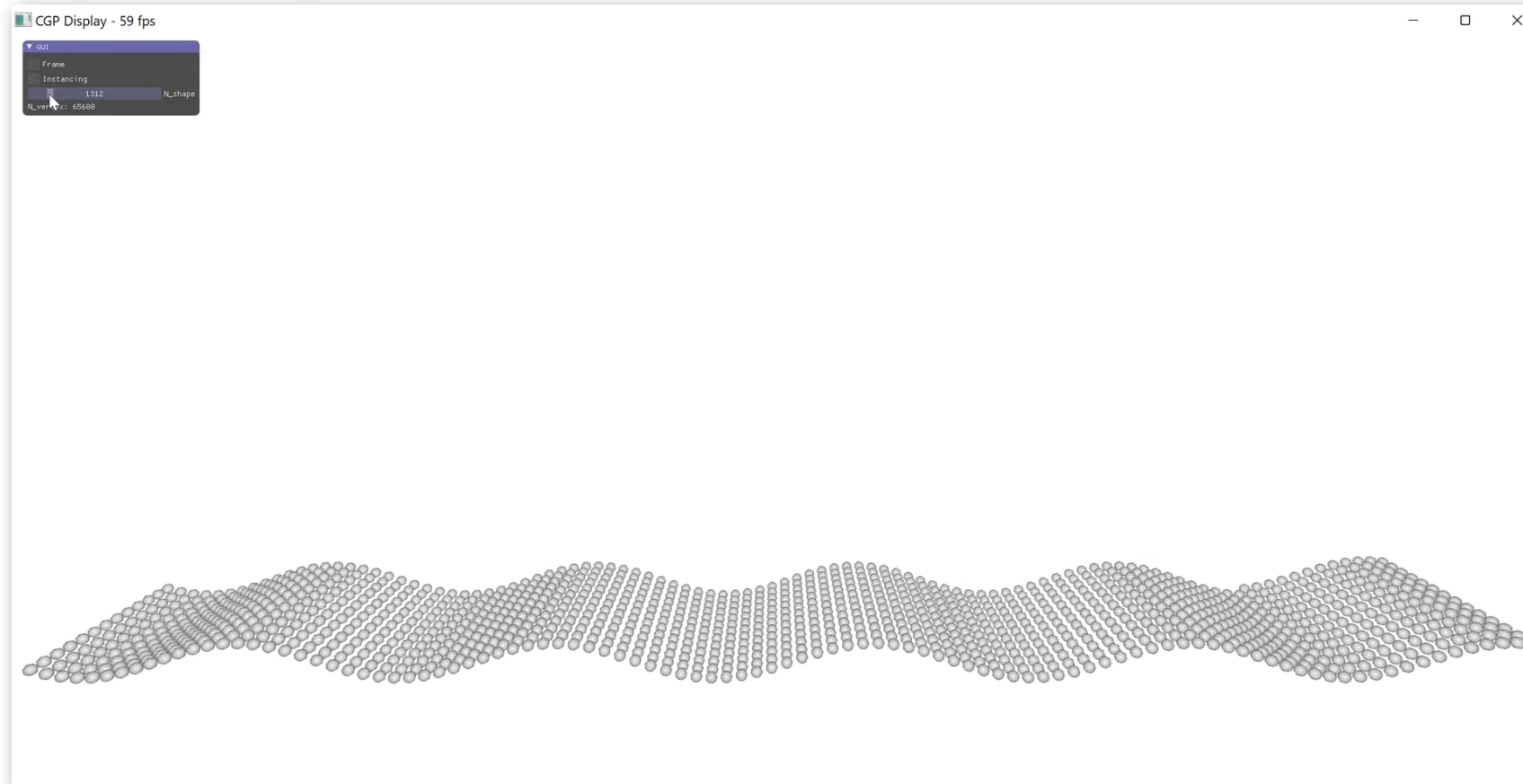
[NVIDIA]

(+) Full capability of your GPU

(-) Require a specific shader

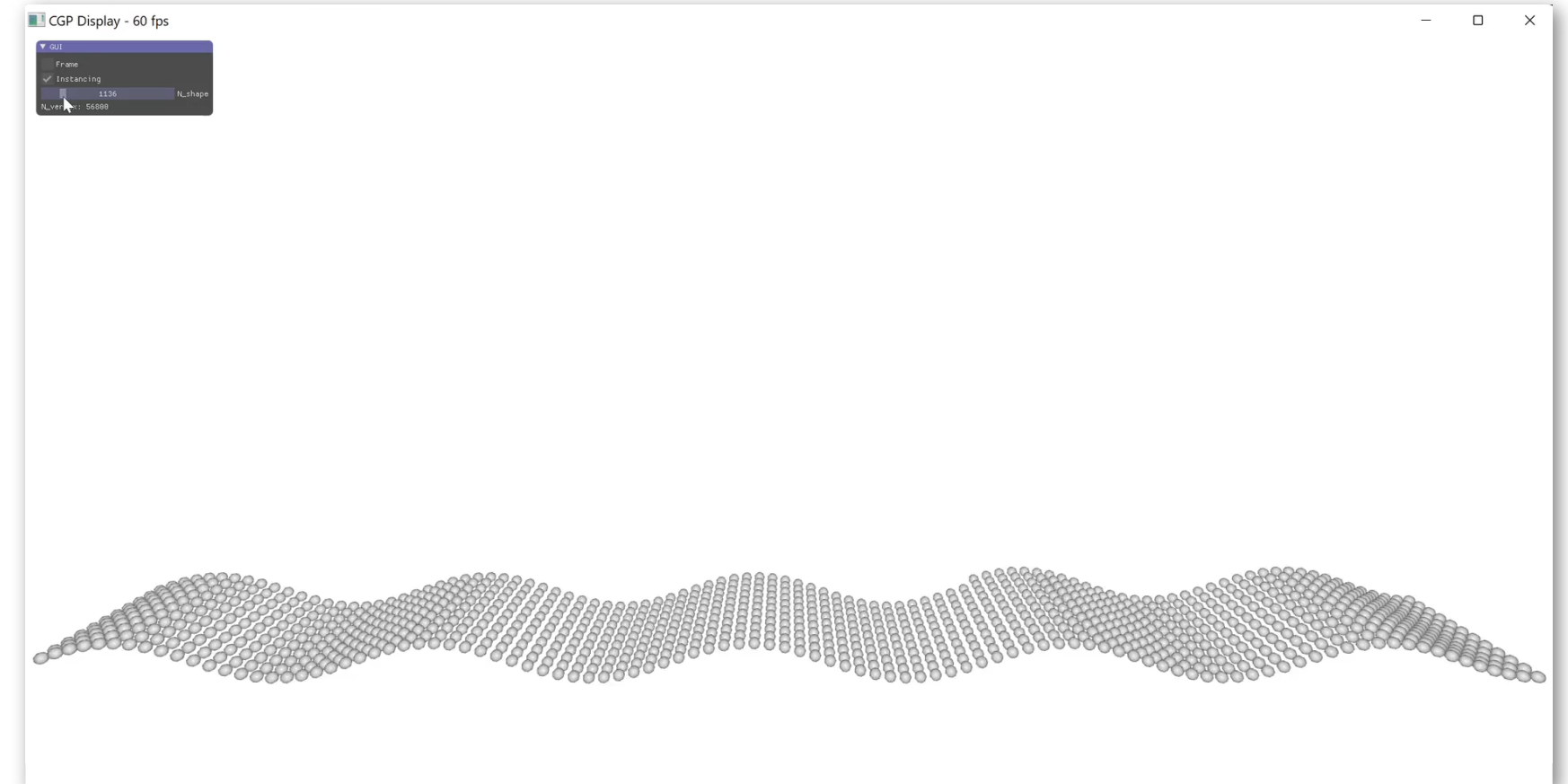
Instancing Demo

Without Instancing



Slow for 15,000 particles
#vertices=750,000

With Instancing



Can display 450,000 particles at 60fps (depends on GPU)
#vertices=22,000,000

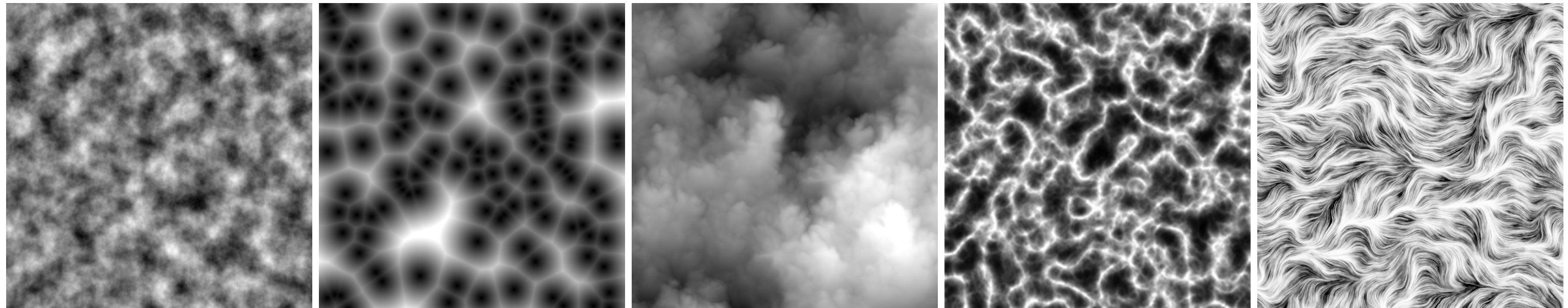
Procedural Animation

Noise function - Perlin Noise

Procedural Noise

What is a spatial/temporal procedural noise:

- Function with visible structure/pattern (limited frequency bandwidth)
- Without visible periodicity
- Deterministic (Same result from a given input)



Examples of 2D procedural textures

Create procedural noise

Ex. Continuous function $f(x)$ with limited frequency.

For integer value: $f(n) =$ pseudo-random, deterministic, value

ex. Hash Function: `float hash(float n) { return fract(sin(n) * 1e4); }`

Interpolate using smooth curve between each integer value (Smooth step, cubic polynomials, etc.)

Properties:

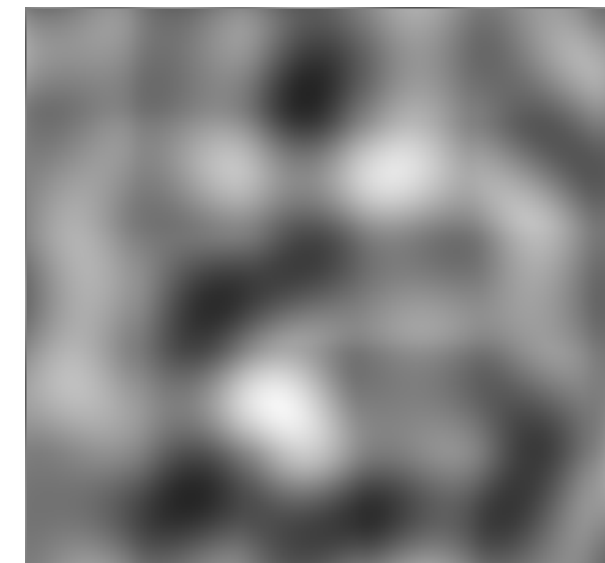
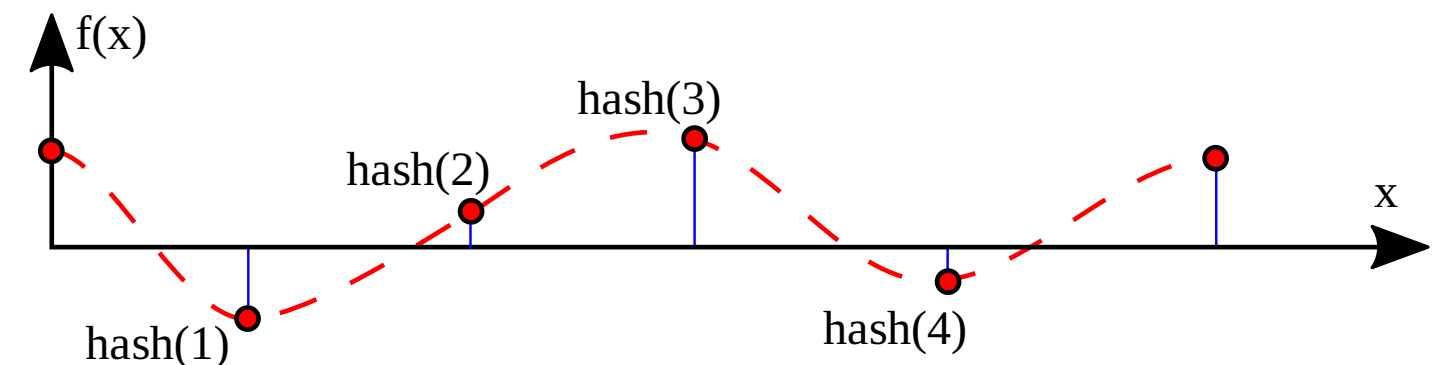
- Continuous function
- Looks non-periodic
- Frequency limited around 1

Can be computed in 2D, 3D, etc.

Simple algorithm - ex. in 1D

For a given x , $n = \text{floor}(x)$

- Evaluate hash function at n , $n + 1$
($(n \pm i)$ for further sampling)
- Compute interpolated value $f(x)$ at position x

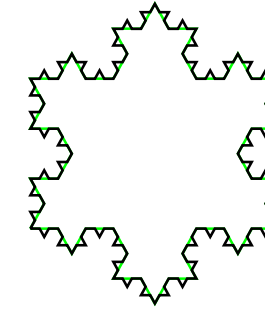
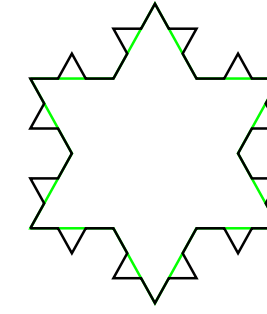
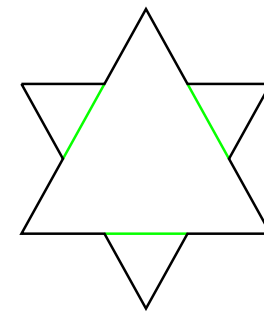
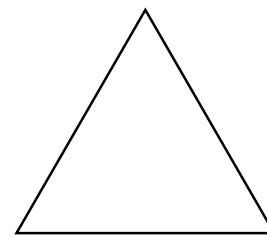


Add high frequency details: Notion of Fractal

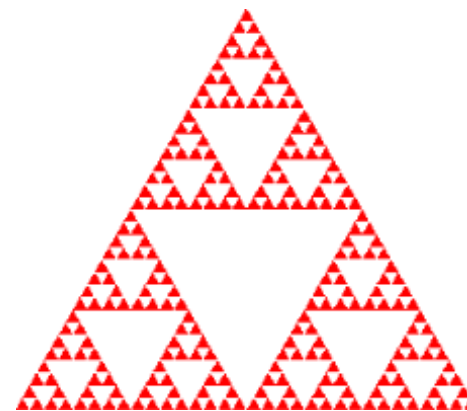
Idea: Recursively add self-similar details

Simple rule $\Rightarrow$ complex shapes

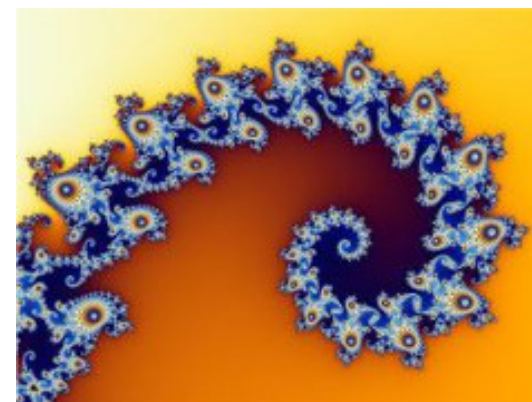
May look like complex natural details



Koch Snowflake



Sierpinski triangle, Shell:Oliva Porphyria



Standard fractals (Mandelbrots, Sierpinski, Newtons's root, etc) are very hard to control.

Not often used directly in CG.

Perlin Noise

First procedural noise proposed in

[Ken Perlin, *An Image Synthesizer*. SIGGRAPH 85]

Idea: Sum over pseudo-random function with increasing frequencies and decreasing magnitude

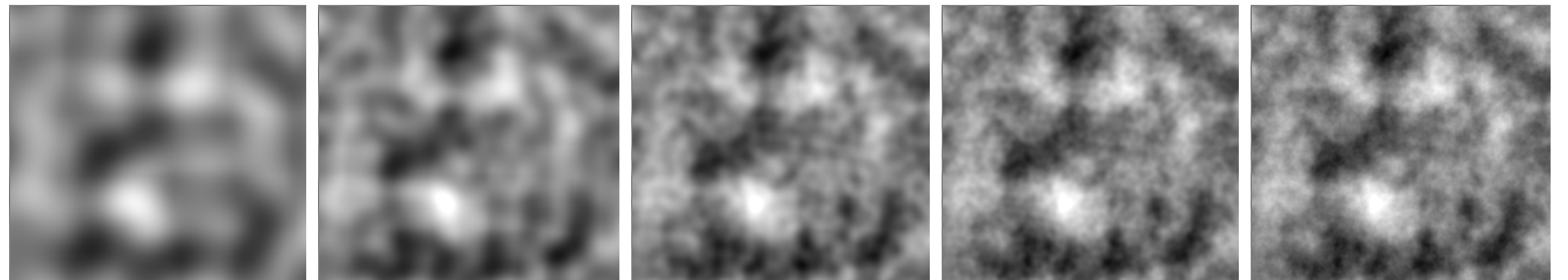
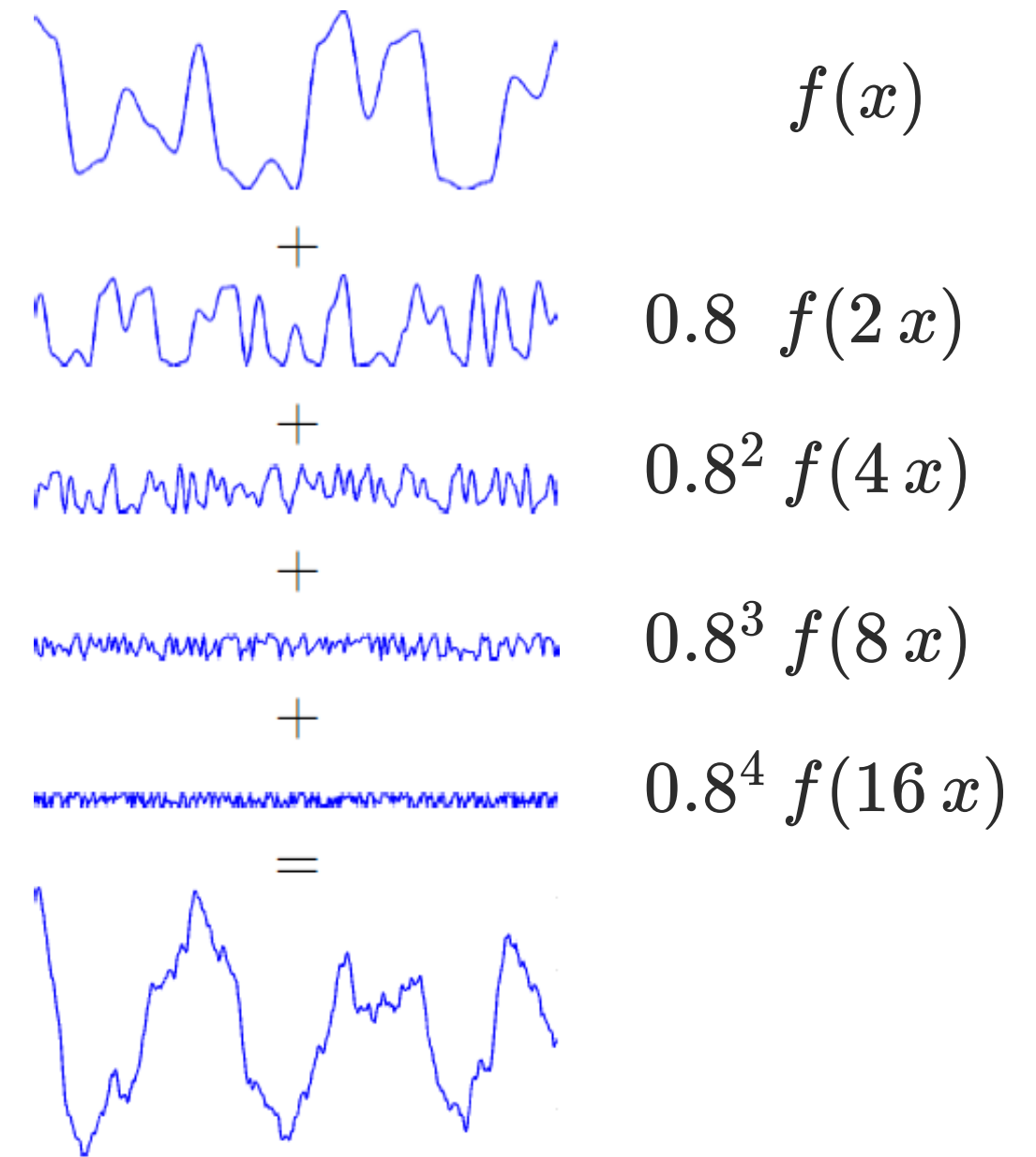
f : Smooth pseudo-random function

$$P(x) = \sum_{k=0}^N \alpha^k f(\omega^k x)$$

- N number of Octave

- α persistency ($1/\alpha$ attenuation)

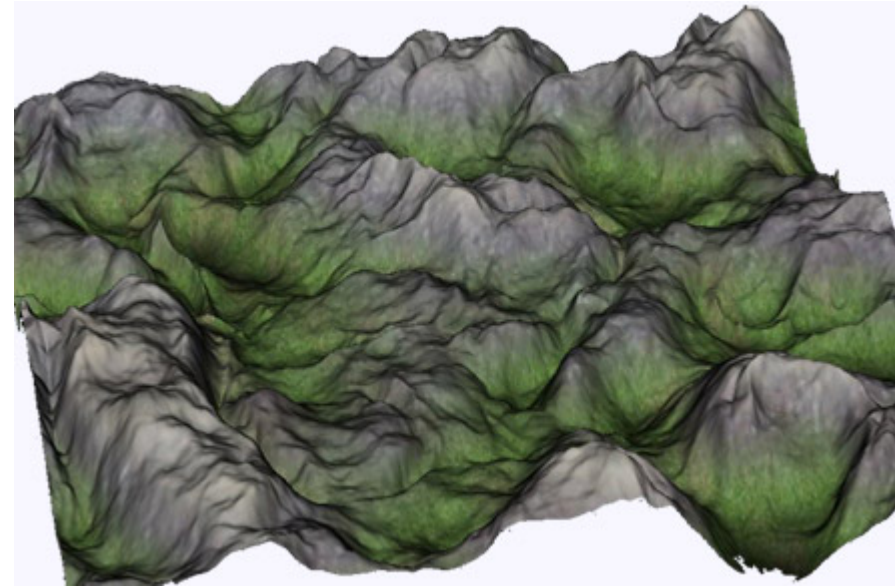
- ω frequency gain



Perlin noise usage

Direct use: $z = P(x, y)$

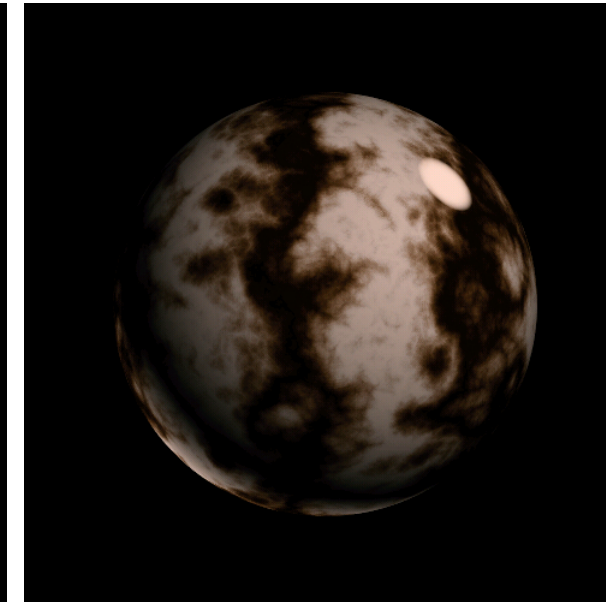
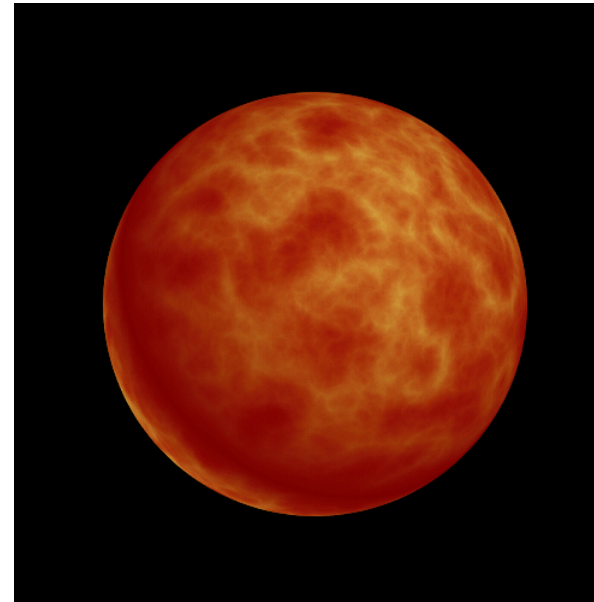
Mountain-looking terrain



Material texture

Ridge effect: $\sum_k \alpha^k |f(\omega^k x)|$

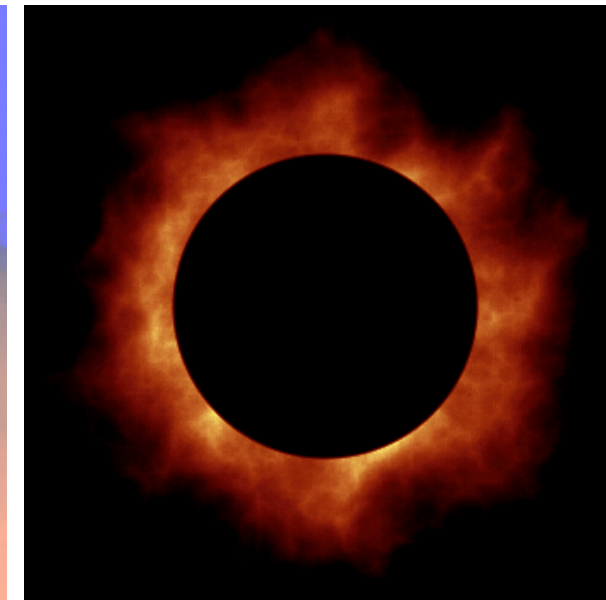
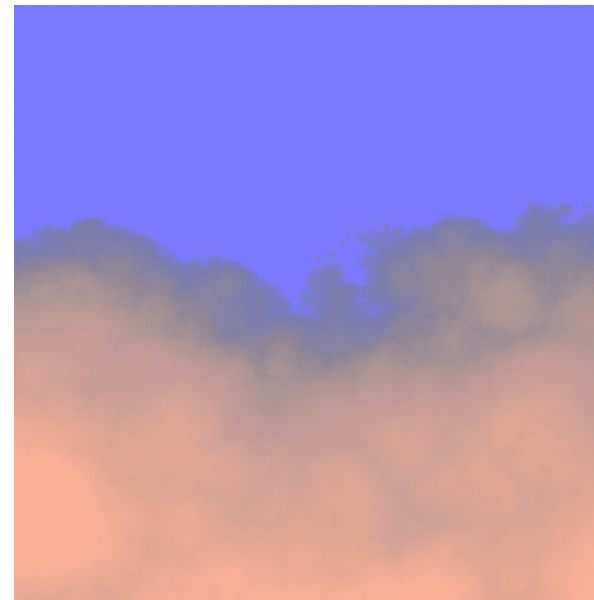
Marble effect: $\sin(x + \sum_k \alpha^k |f(\omega^k x)|)$



Animated textures

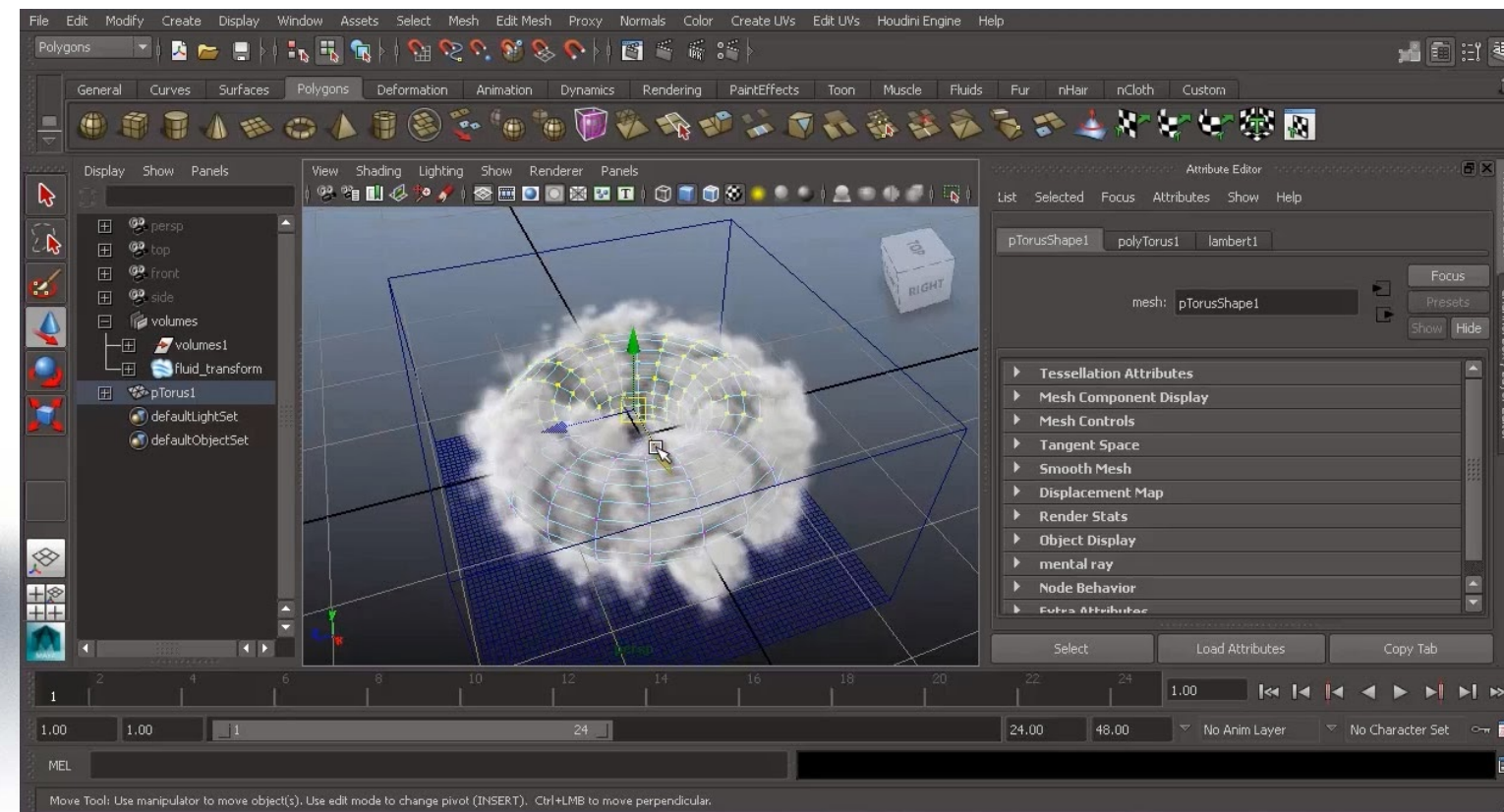
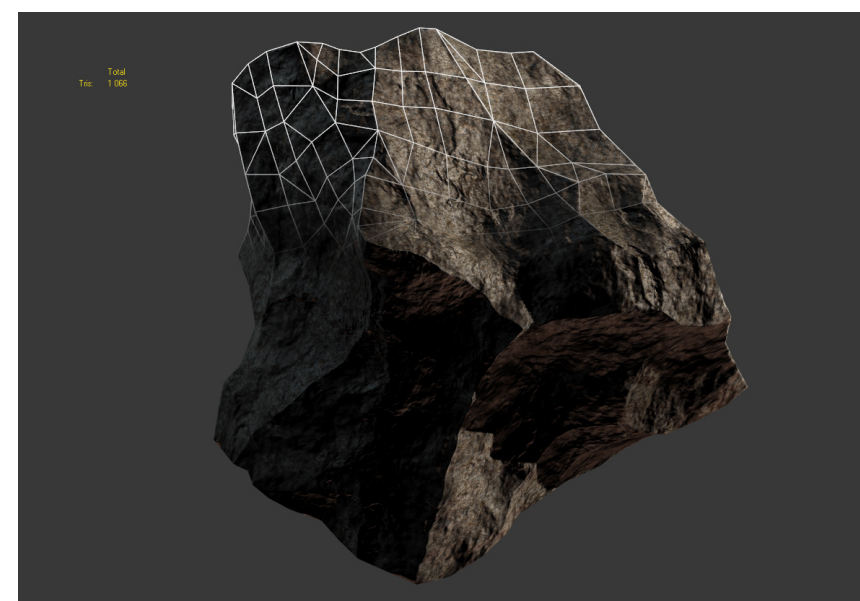
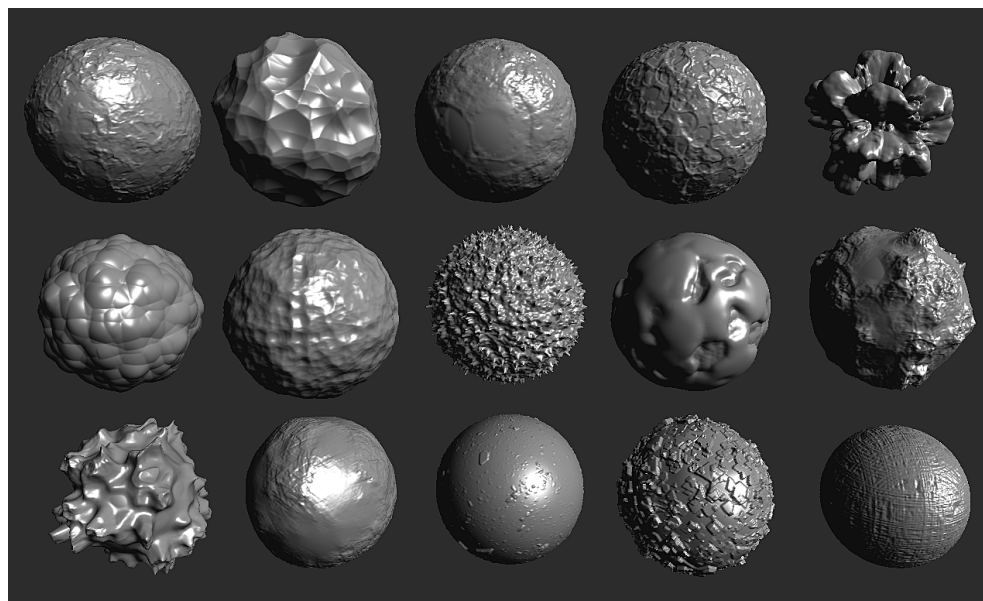
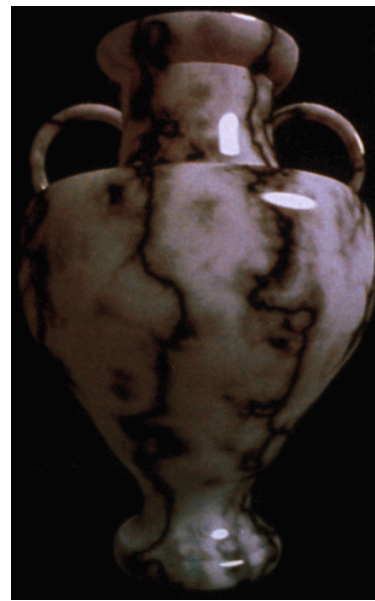
Translation: $P(x, y + t)$

Smooth evolution: $P(x, y, t)$



Perlin noise applications

In almost every complex shapes ...



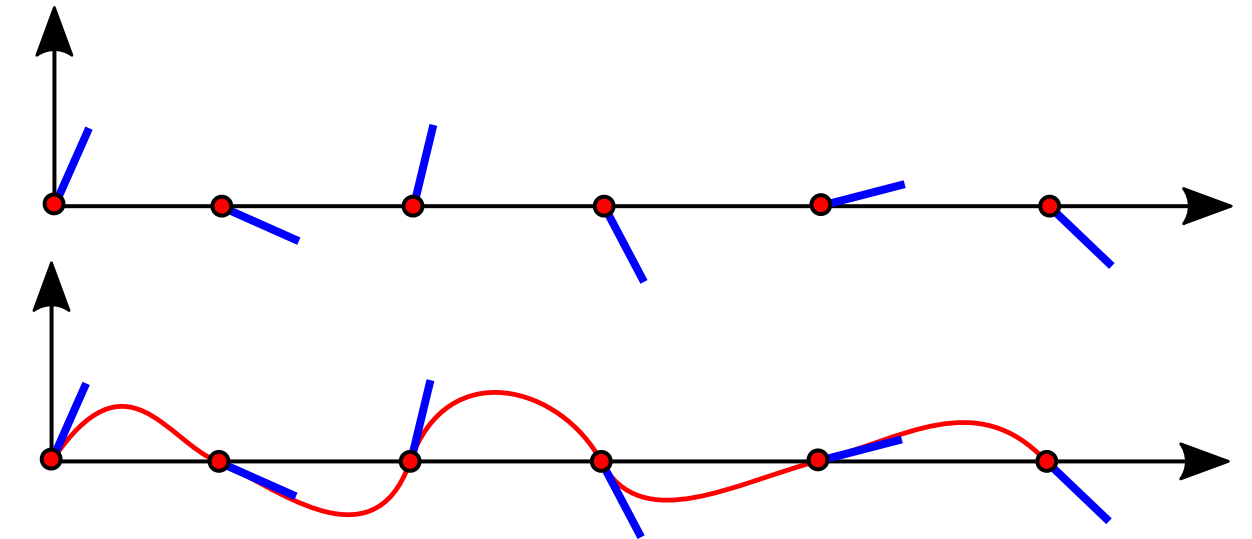
Look at [Shader Toy](#) + Noise [example](#)

Perlin Noise - Improvement

Gradient Noise

Use pseudo-random Gradients instead of Positions

Improved frequency control: Oscillate at period 1



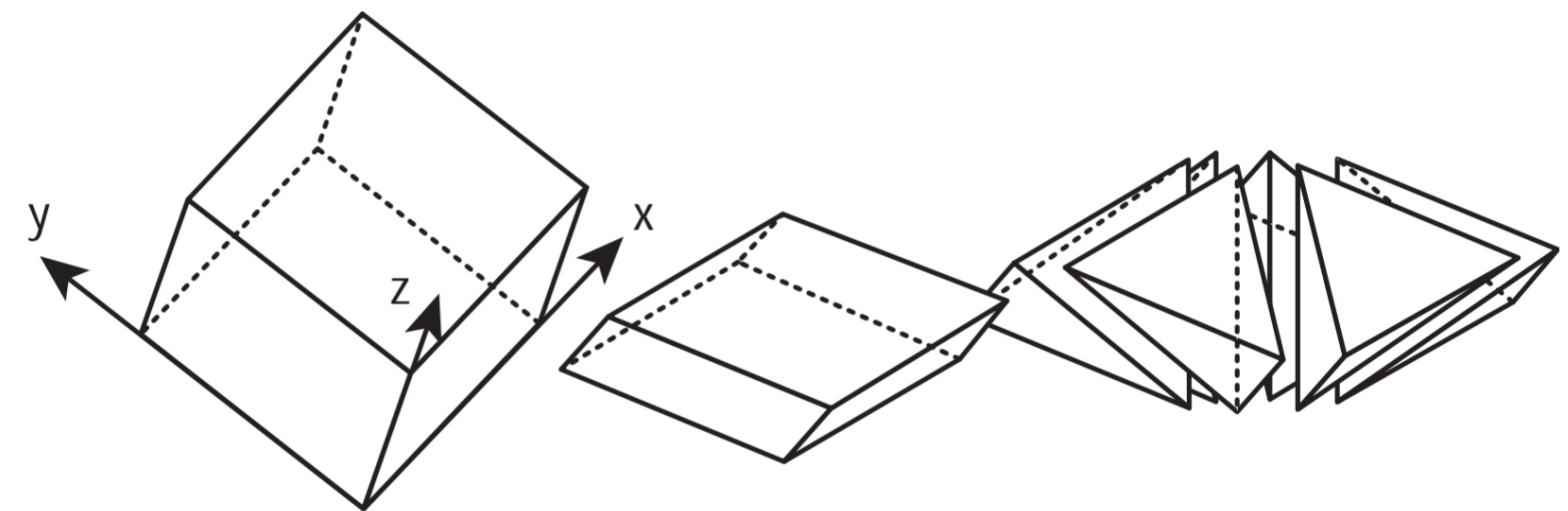
Simplex Noise

Simplex-based interpolation instead of grid interpolation

Avoids grid-direction artifact

Faster for high dimensions

[Stefan Gustavson Simplex noise demystified]



To go beyond:

[A Lagae et al., STAR in Procedural Noise Functions. EG 2010]

Perlin Noise Terrain

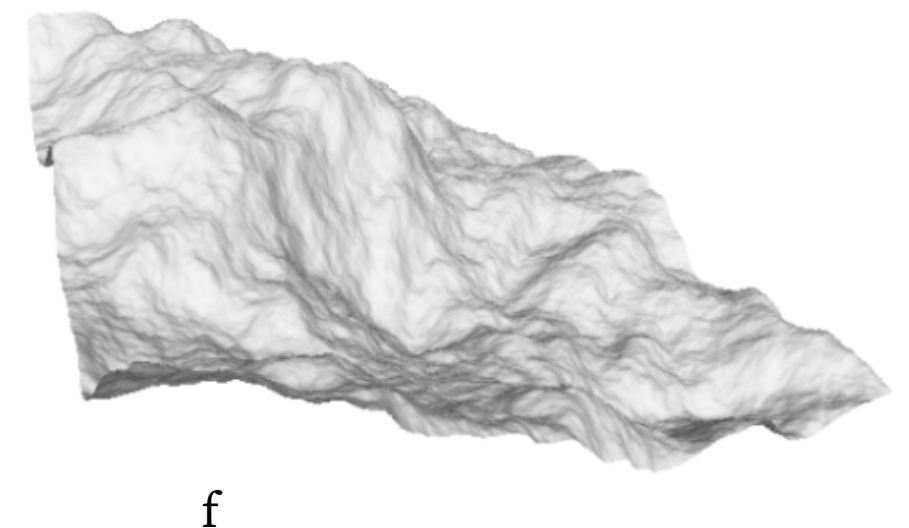
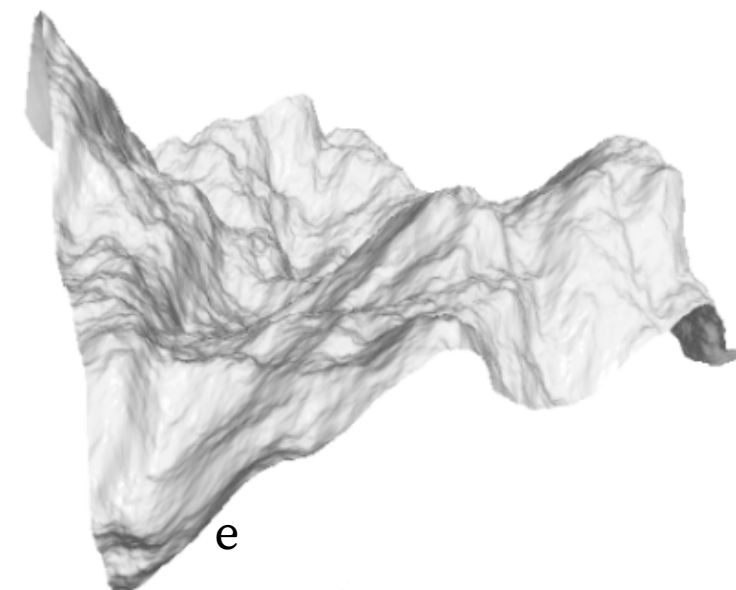
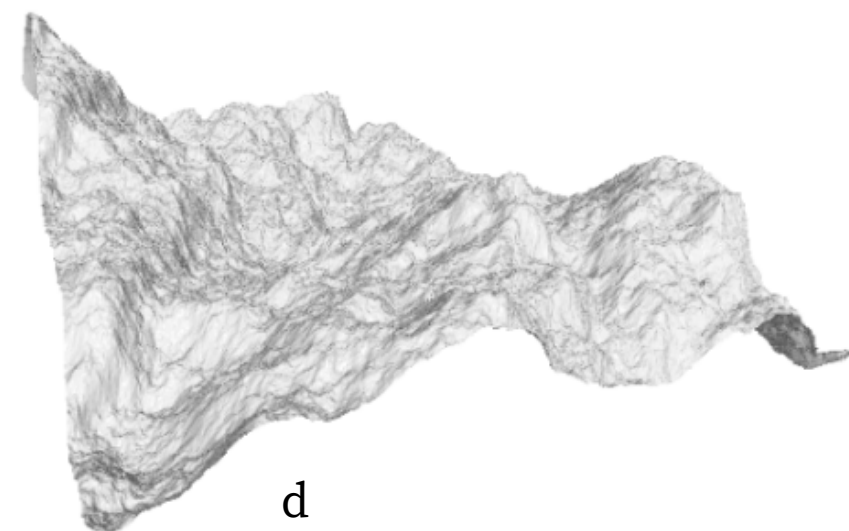
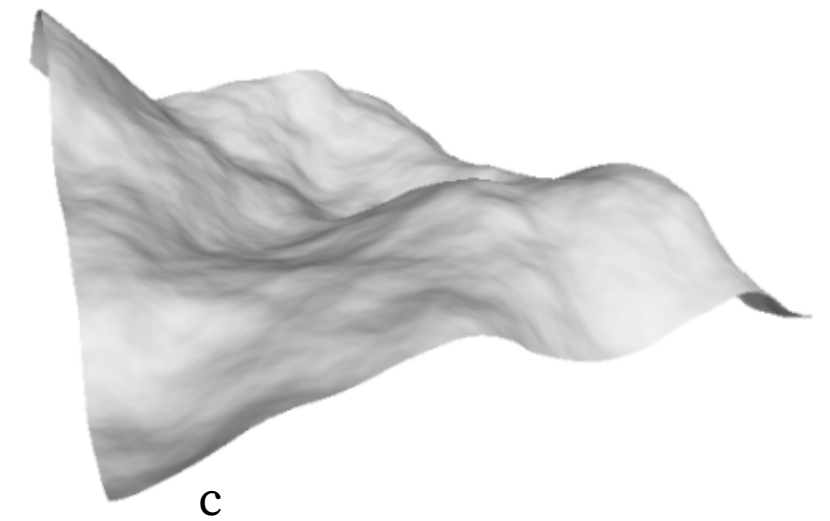
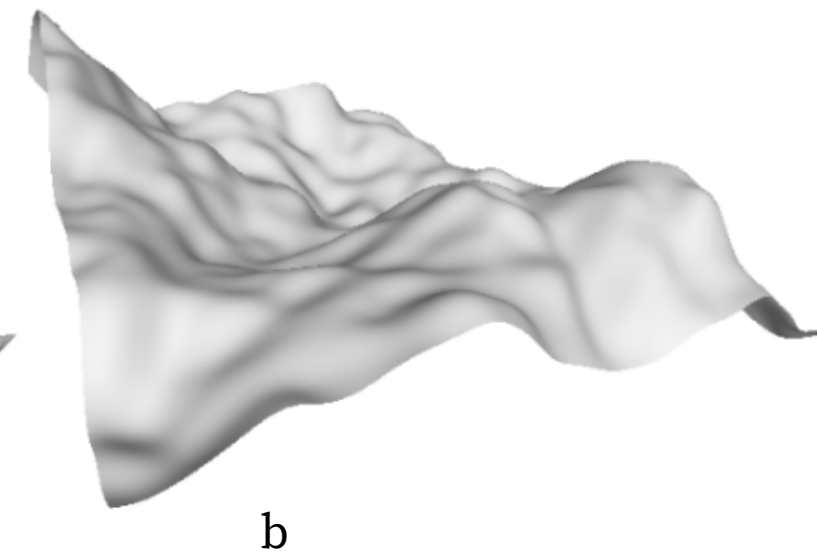
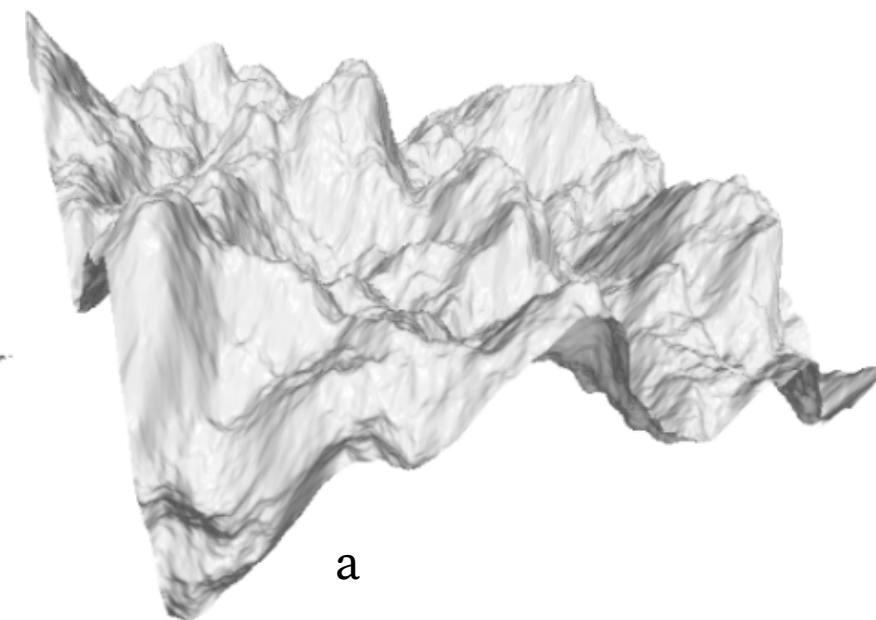
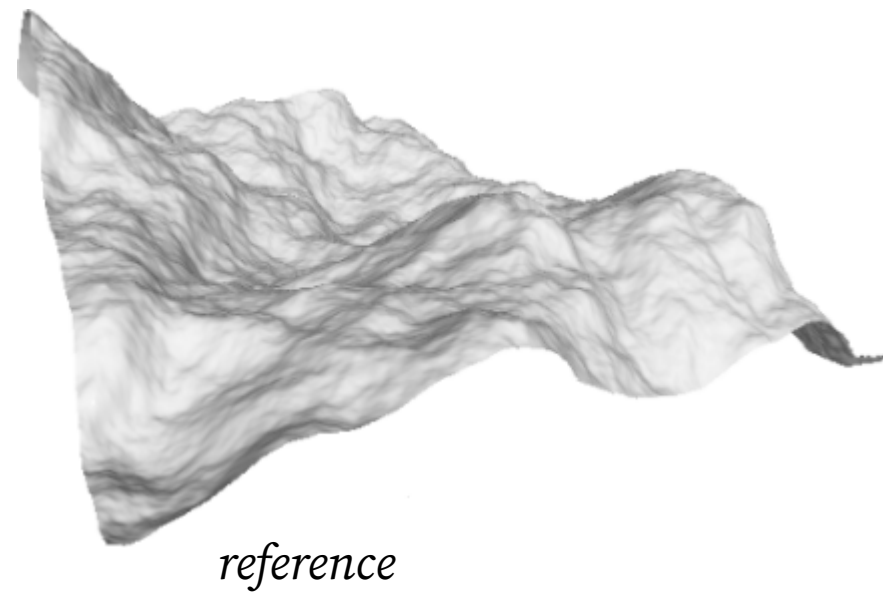
Consider the surface S defined as

$$S(u, v) = \begin{cases} x(u, v) = u \\ y(u, v) = v \\ z(u, v) = h P(s(u + o), s(v + o)) \end{cases}$$

The perlin noise

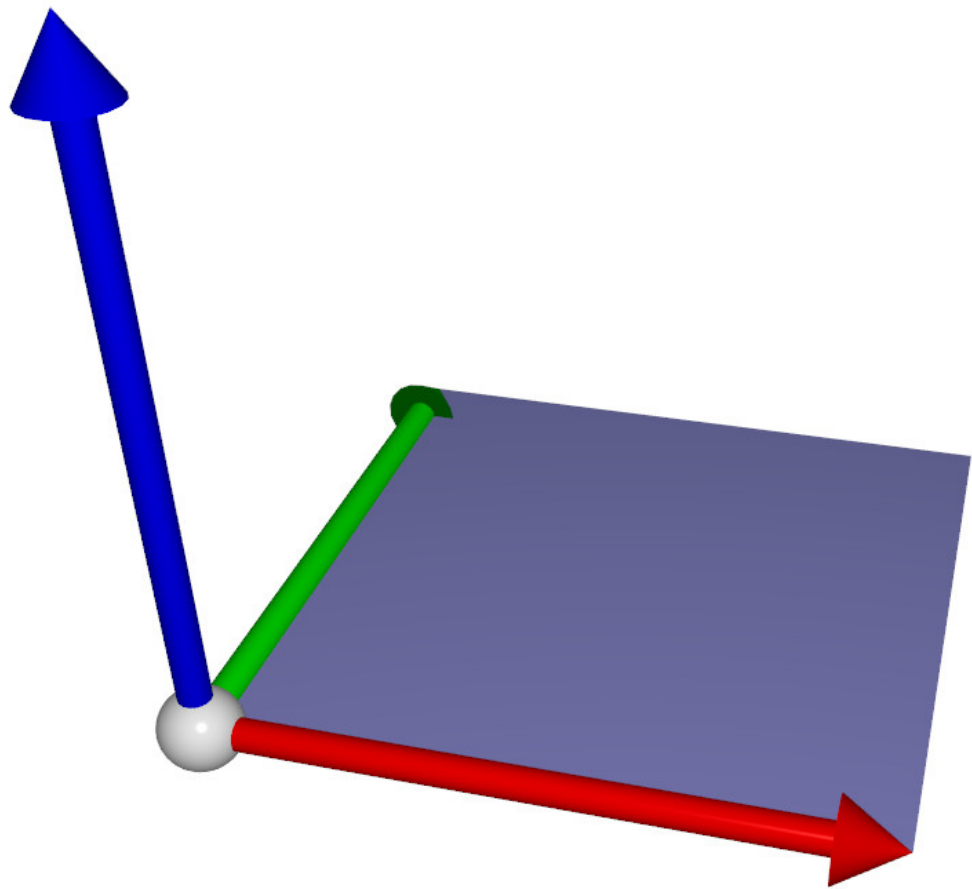
$$P(u, v) = \sum_{k=0}^N \alpha^k f(2^k u, 2^k v)$$

$$\begin{aligned} N &= 9 \\ \alpha &= 0.4 \\ h &= 0.3 \\ s &= 1 \\ o &= 0 \end{aligned}$$



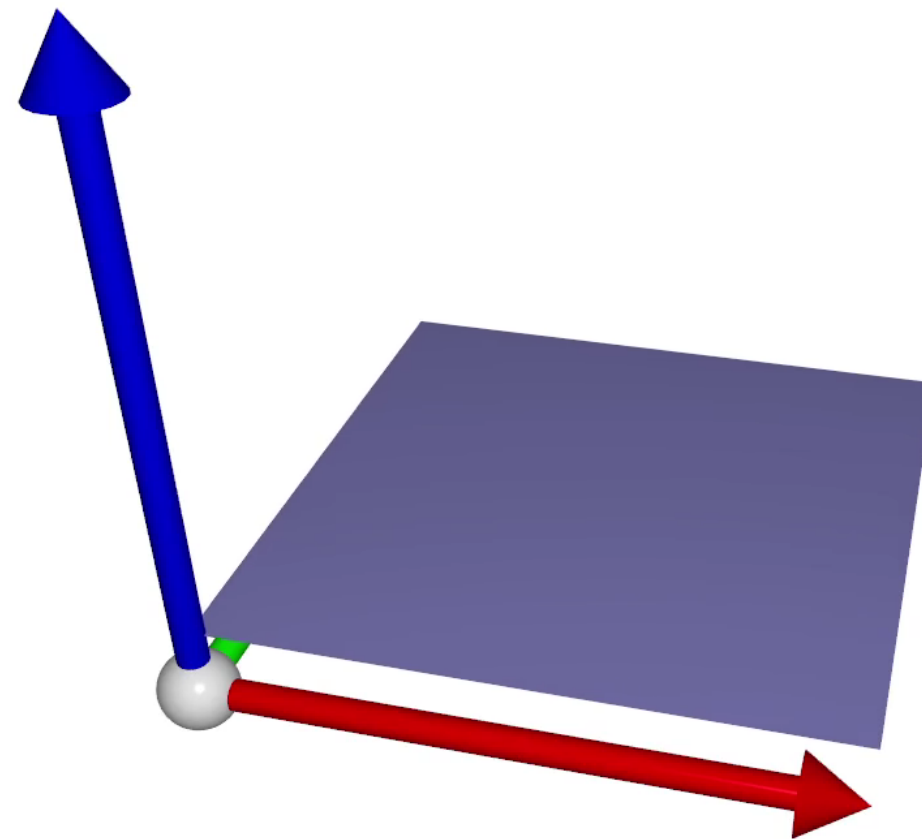
Q. Which parameters correspond to (a,b,c,d,e,f) terrains?

Perlin Noise - Animation



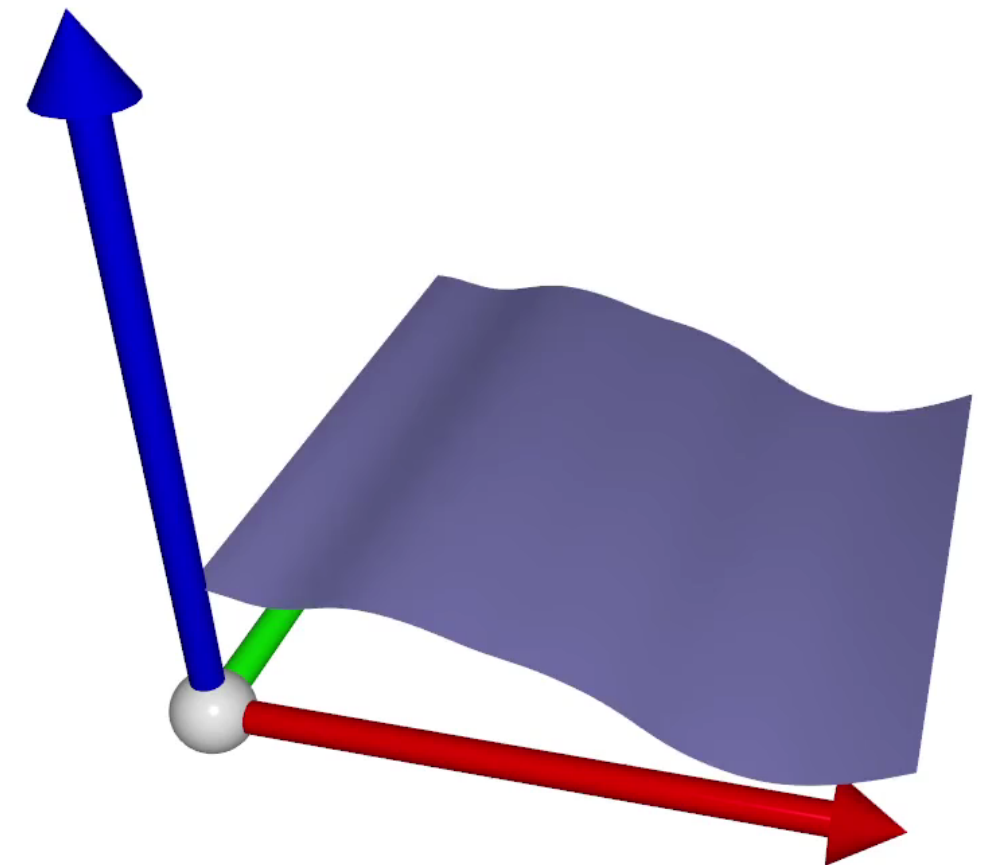
Reference surface function

$$(u, v) \in [0, 1]^2, S(u, v) = (u, v, 0)$$



Perlin noise can depends on time

$$S(u, v, t) = (u, v, P(t))$$



Depends on space and time

$$S(u, v, t) = (u, v, P(u, t))$$

Perlin Noise - Animation

Reference surface function: $(u, v) \in [0, 1]^2$, $f(u, v) = (u, v, 0)$

Q. How generate the following animations? $S(u, v, t) = \dots$

