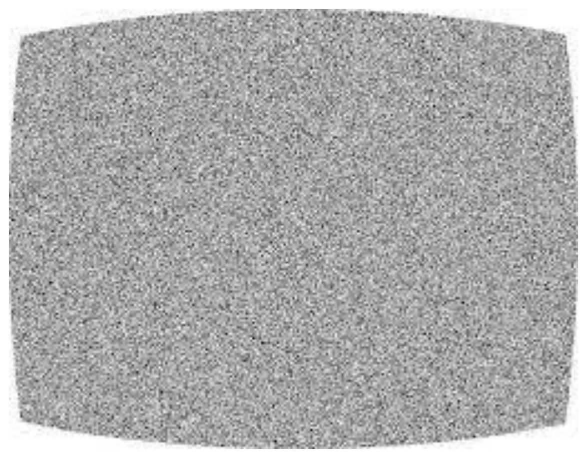
Noises

Jaanus Jaggo



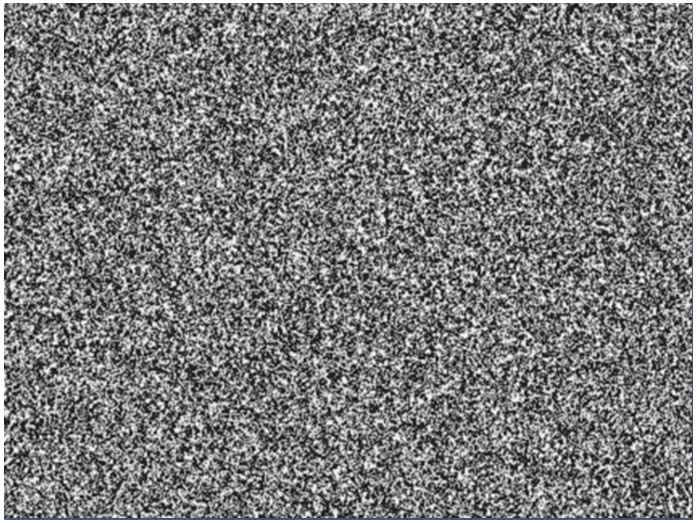
Noise

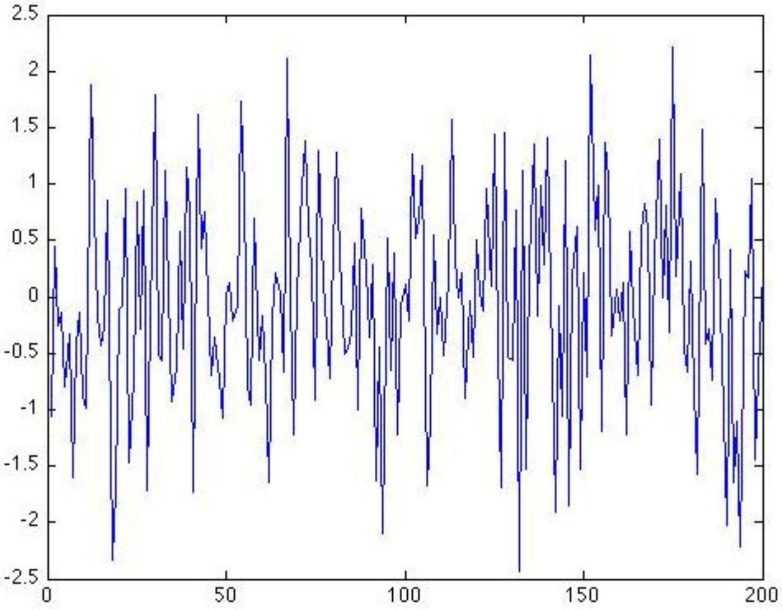
Noise is a function:

noise(coordinate) -> value

Pseudo-random: gives the appearance of randomness
Determinism: same input gives the same result every time

White noise

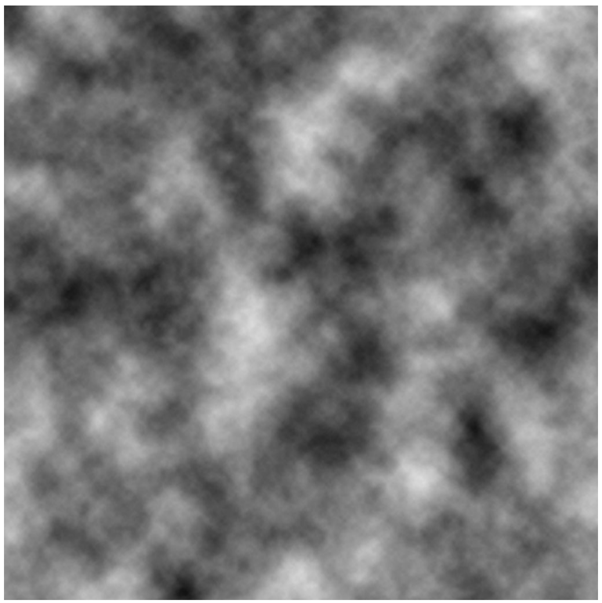




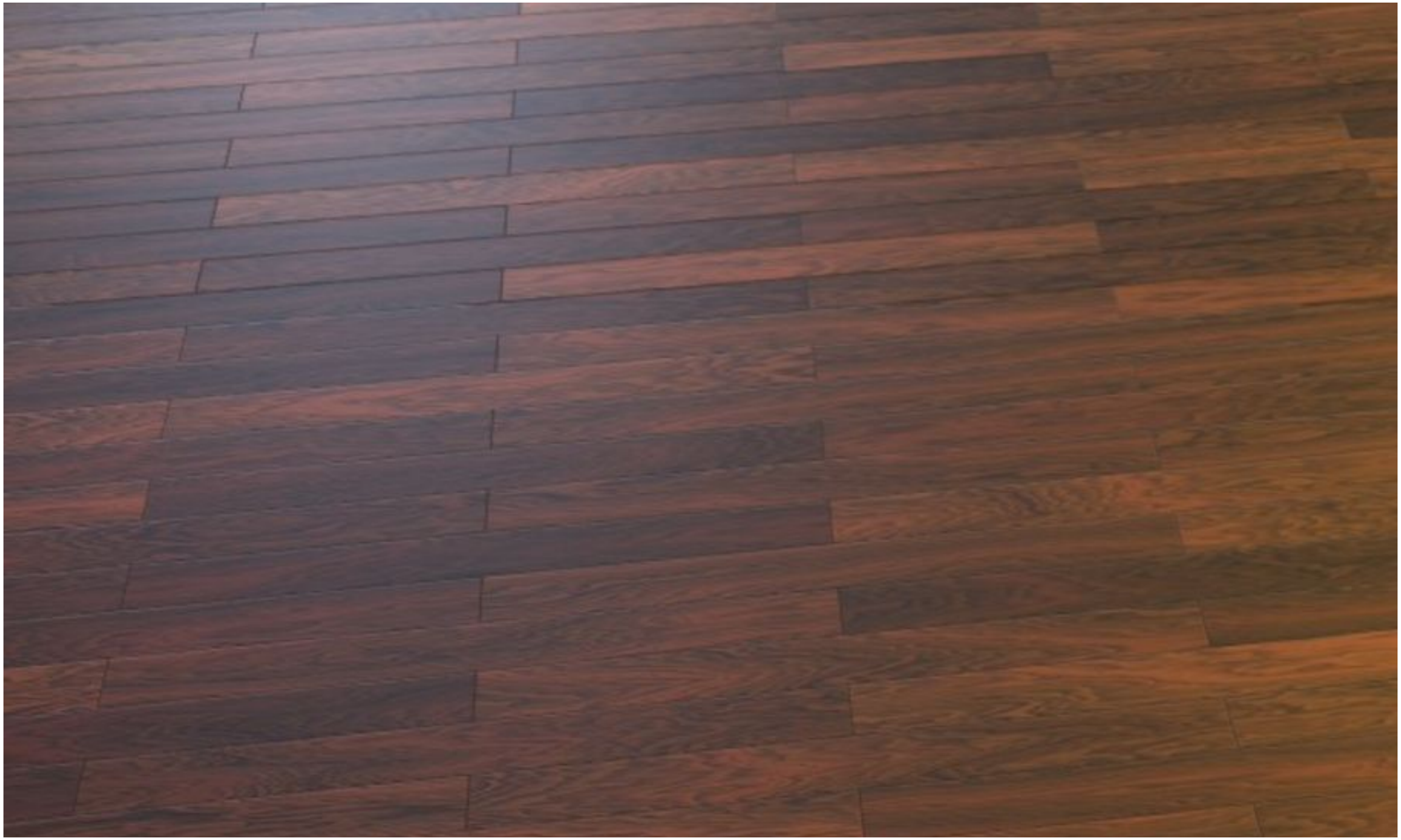
? Dimensions

? Dimensions

Better noise



Combination of noises

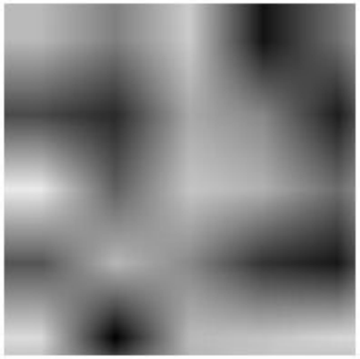


http://www.blendswap.com/blends/view/80871

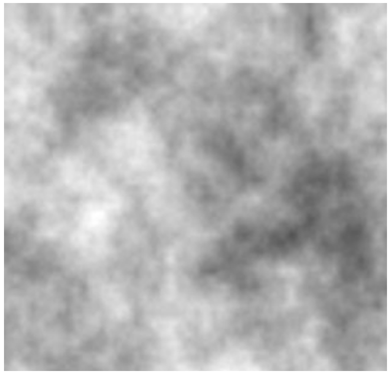
Value noise

x3

x9

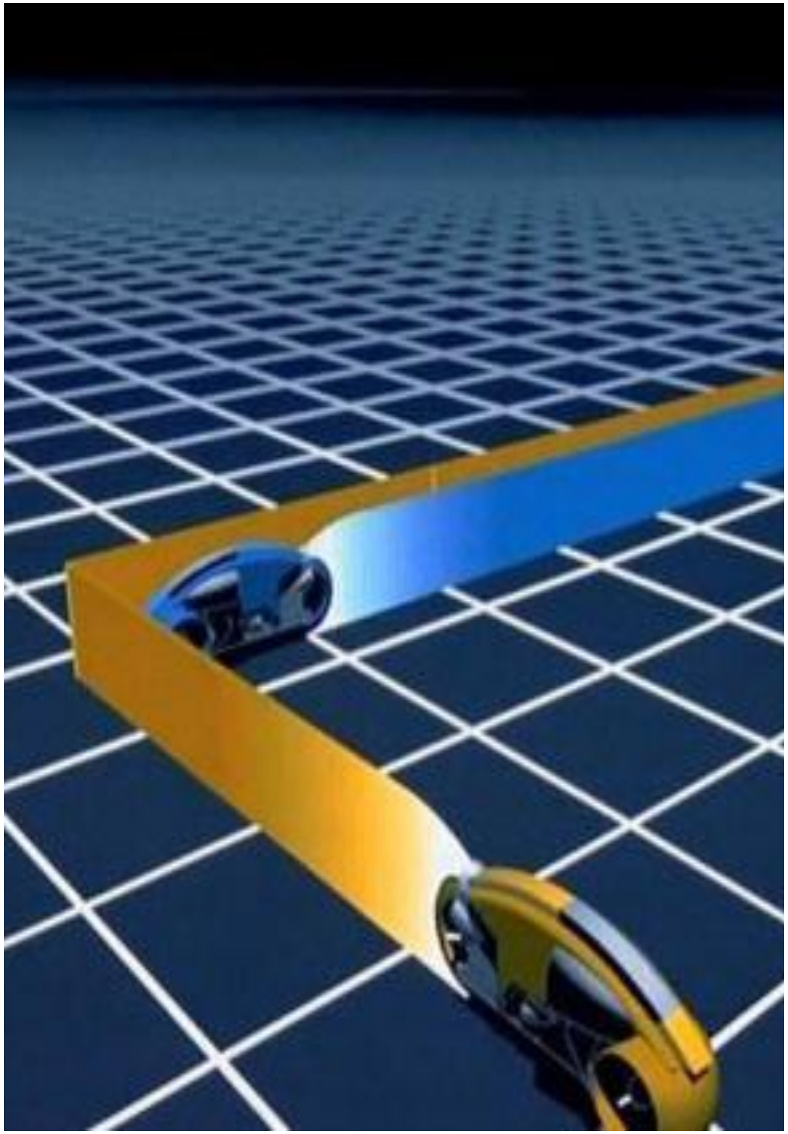


x27



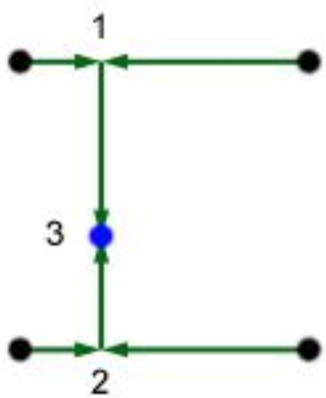
Perlin noise

- **Author:** Ken Perlin
- **Idea:** 1-st Tron movie
- **Complexity:** $O(2^n)$

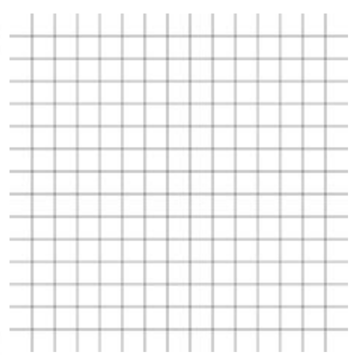


Perlin Implementation

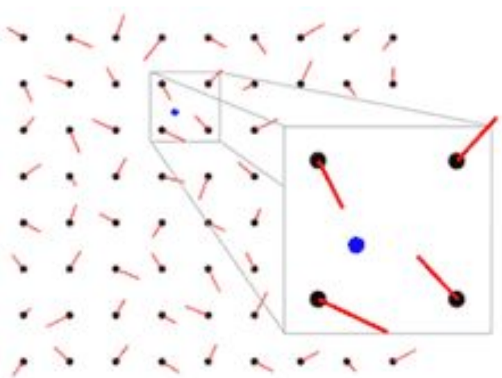
1. Define n-dimensional grid
2. Assign a gradient vector to each grid coordinate
    - Lookup table / texture / hash function
3. Find dot product between the **gradient vector** and **distance vector** (2D - 4 products, 3D - 8 products)
4. Interpolate between the dot product values



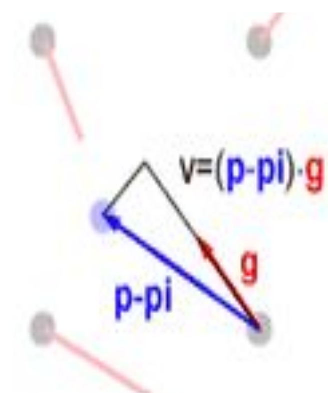
A



B



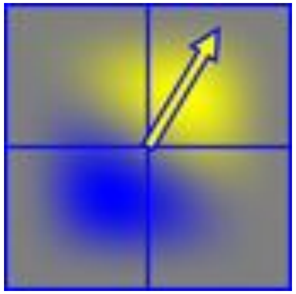
C

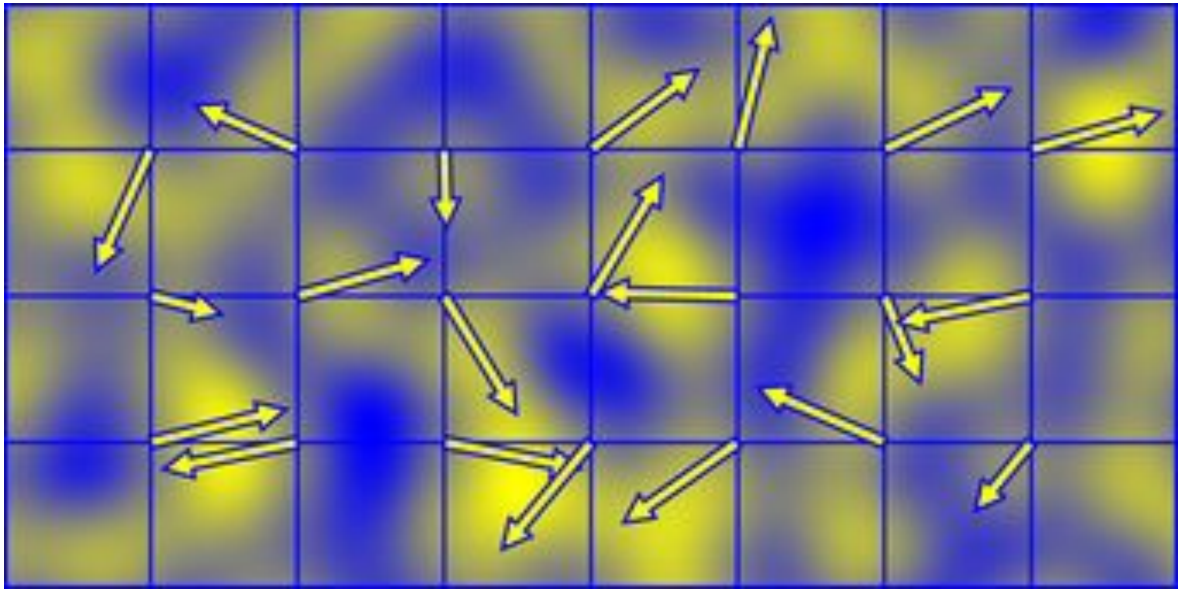


D

Perlin Implementation

yellow - positive
blue - negative





Pseudocode

```
float perlin(float x, float y) {
    // Determine grid cell coordinates
    int x0 = int(x);
    int x1 = x0 + 1;
    int y0 = int(y);
    int y1 = y0 + 1;

    // Determine interpolation weights
    // Could also use higher order polynomial/s-curve
    float sx = x - (float)x0;
    float sy = y - (float)y0;

    // Interpolate between grid point gradients
    float n0, n1, ix0, ix1, value;
    n0 = dotGridGradient(x0, y0, x, y);
    n1 = dotGridGradient(x1, y0, x, y);
    ix0 = lerp(n0, n1, sx);
    n0 = dotGridGradient(x0, y1, x, y);
    n1 = dotGridGradient(x1, y1, x, y);
    ix1 = lerp(n0, n1, sx);
    value = lerp(ix0, ix1, sy);

    return value;
}
```

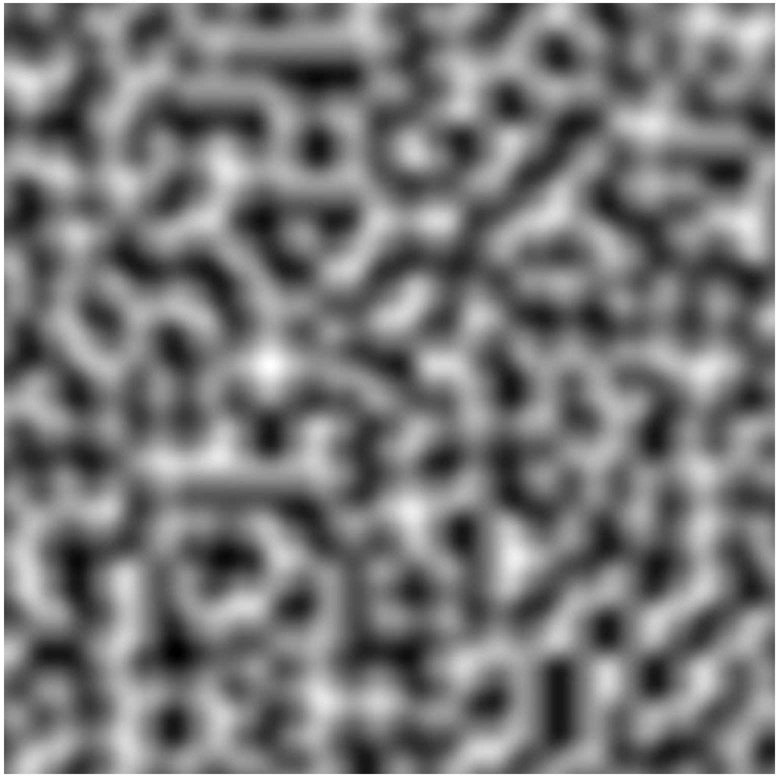
```
float dotGridGradient(int ix, int iy, float x, float y) {
    // Precomputed (or otherwise) gradient vectors at each grid node
    extern float Gradient[IYMAX][IXMAX][2];

    // Compute the distance vector
    float dx = x - (float)ix;
    float dy = y - (float)iy;

    // Compute the dot-product
    return (dx*Gradient[iy][ix][0] + dy*Gradient[iy][ix][1]);
}
```

Simplex noise

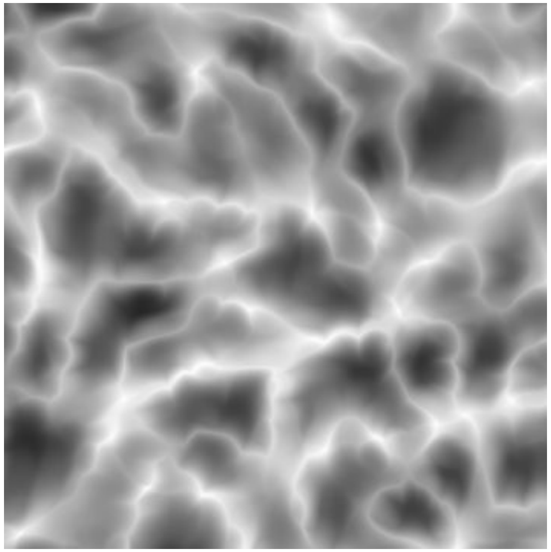
- **Author:** Ken Perlin
- **Complexity:** $O(n^2)$
  - Scales well on high dimensions.

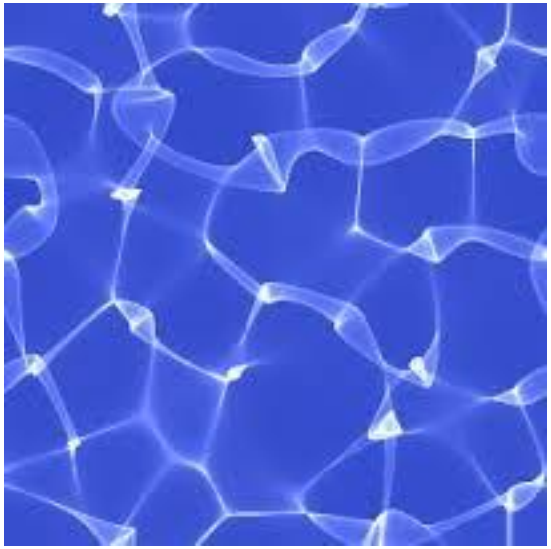


Uses simplicial grid

(**triangles** instead of squares, **tetrahedron** instead of cubes)

Applications - textures

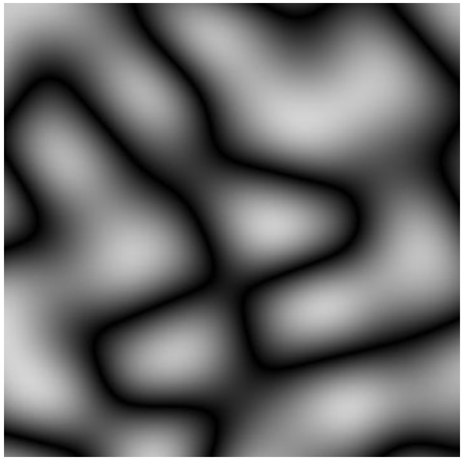




Creating textures

simplex(p)

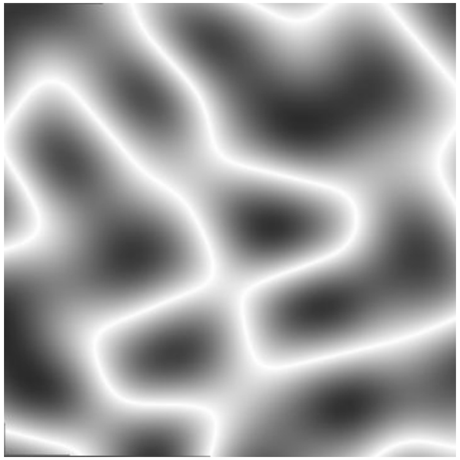
abs(simplex(p))



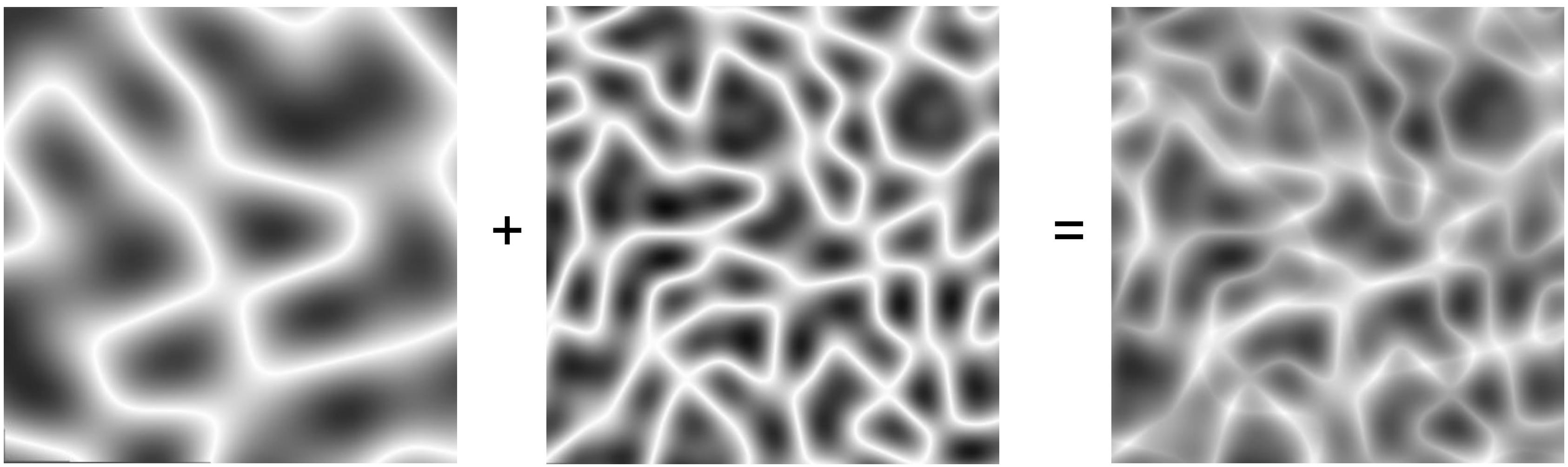
billow

1 - (abs(simplex(p)))

ridged

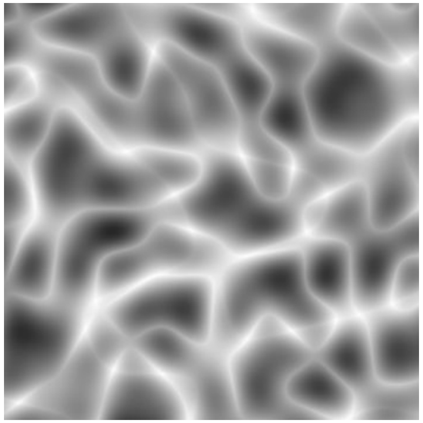


Creating textures

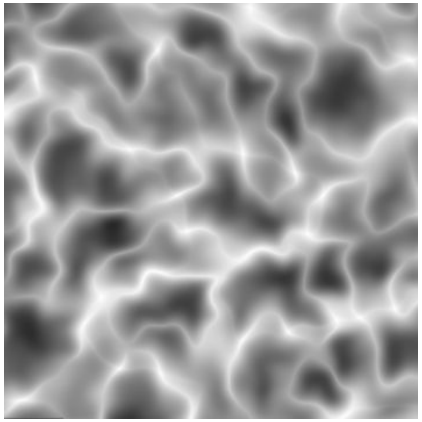


Creating textures

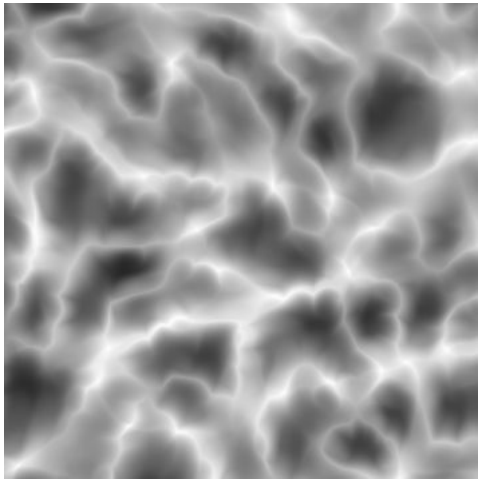
Another simplex noise for distortion



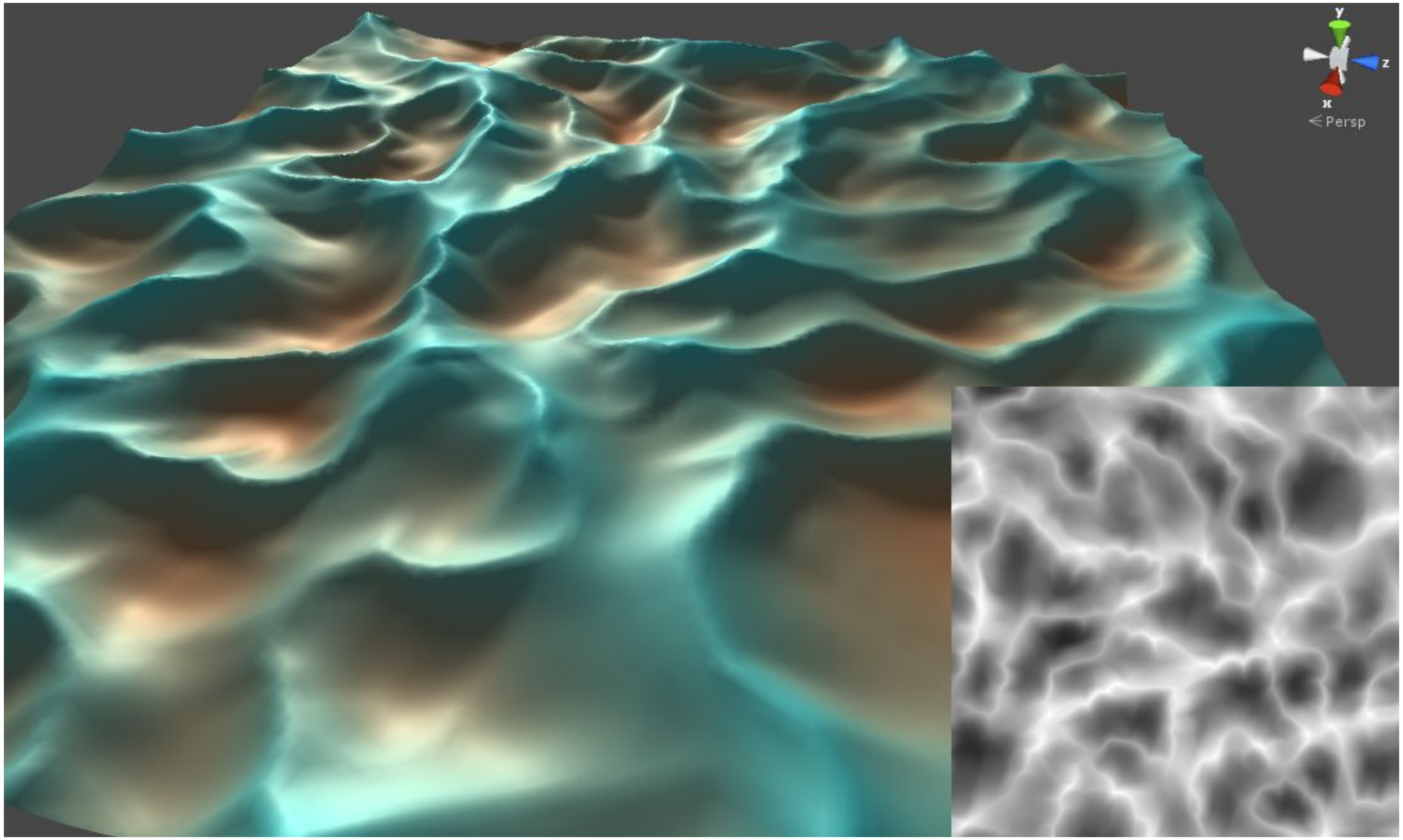
->



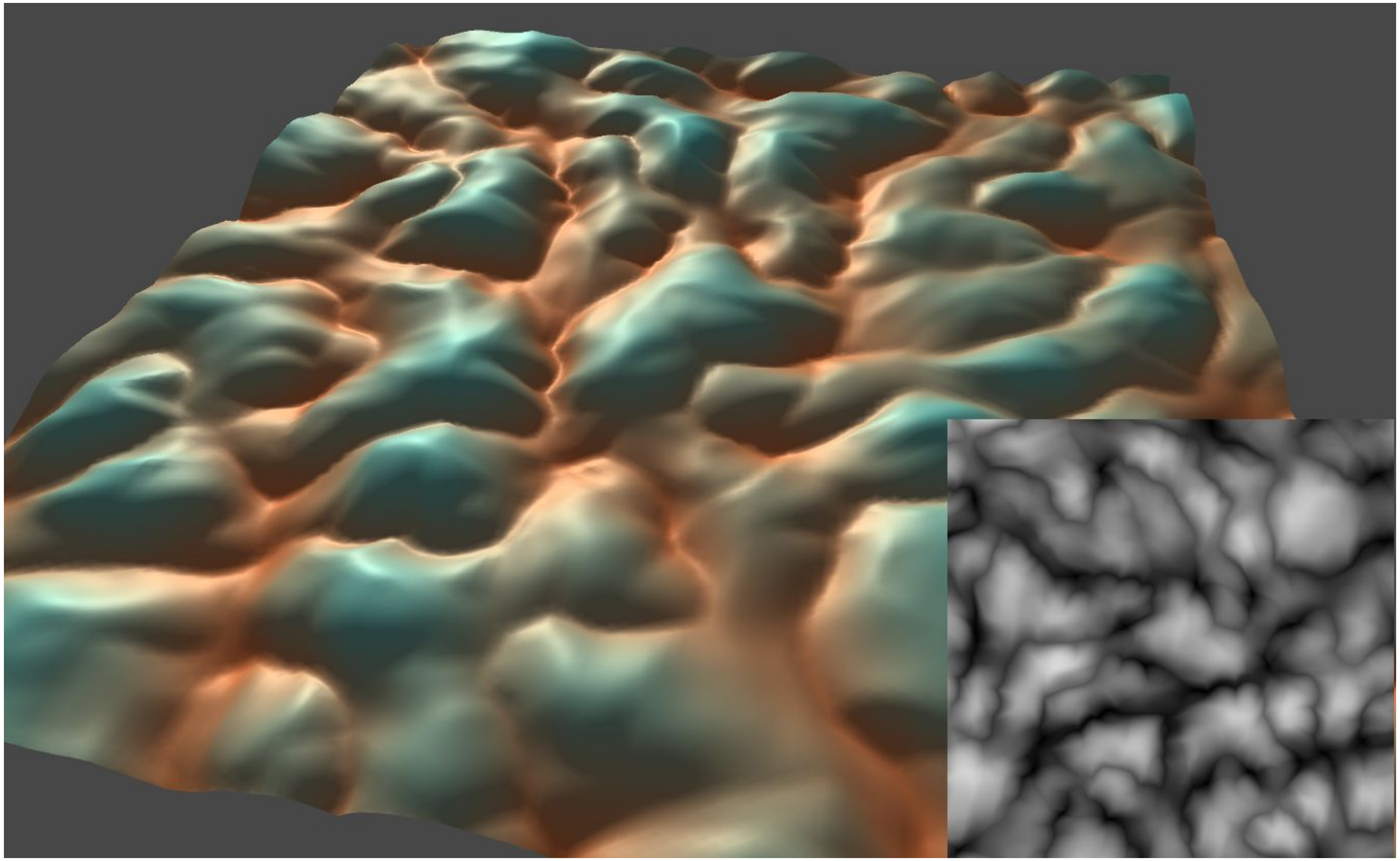
Or use ridged noise instead



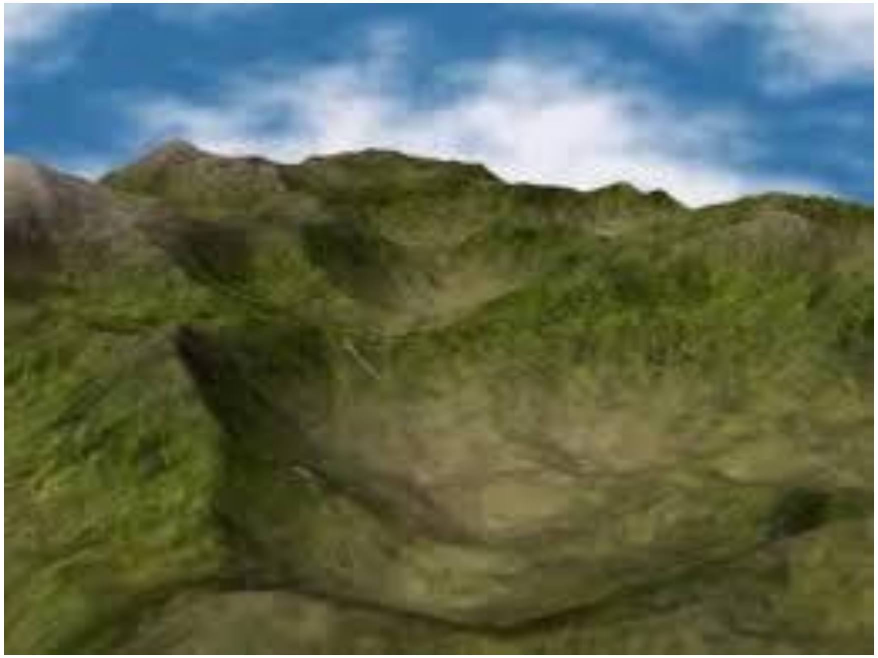
Result



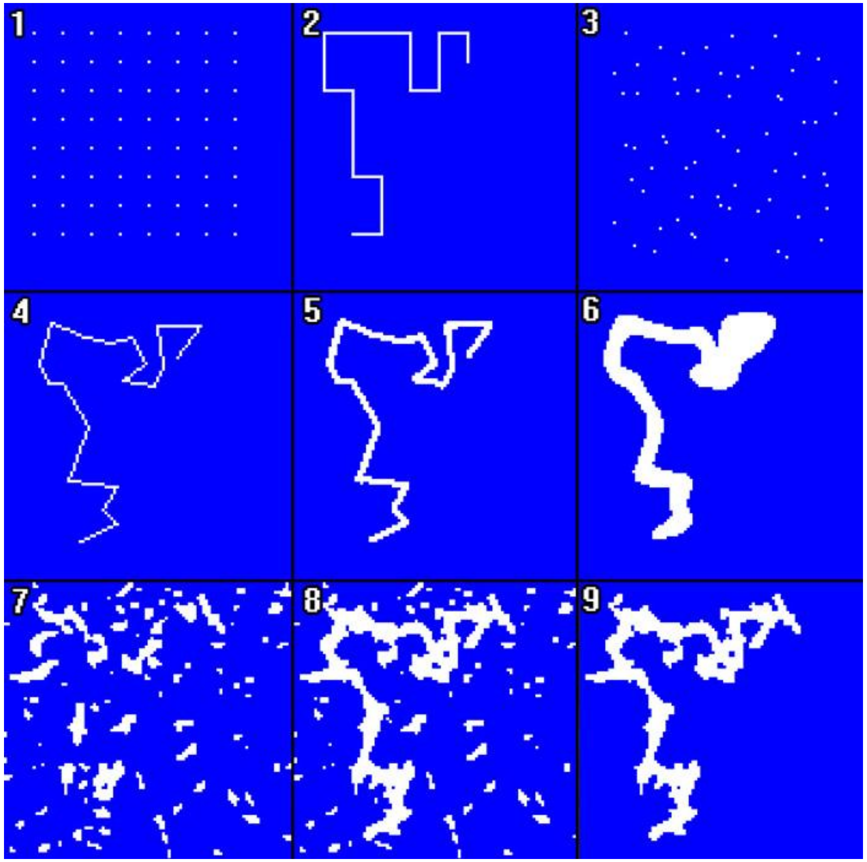
Result

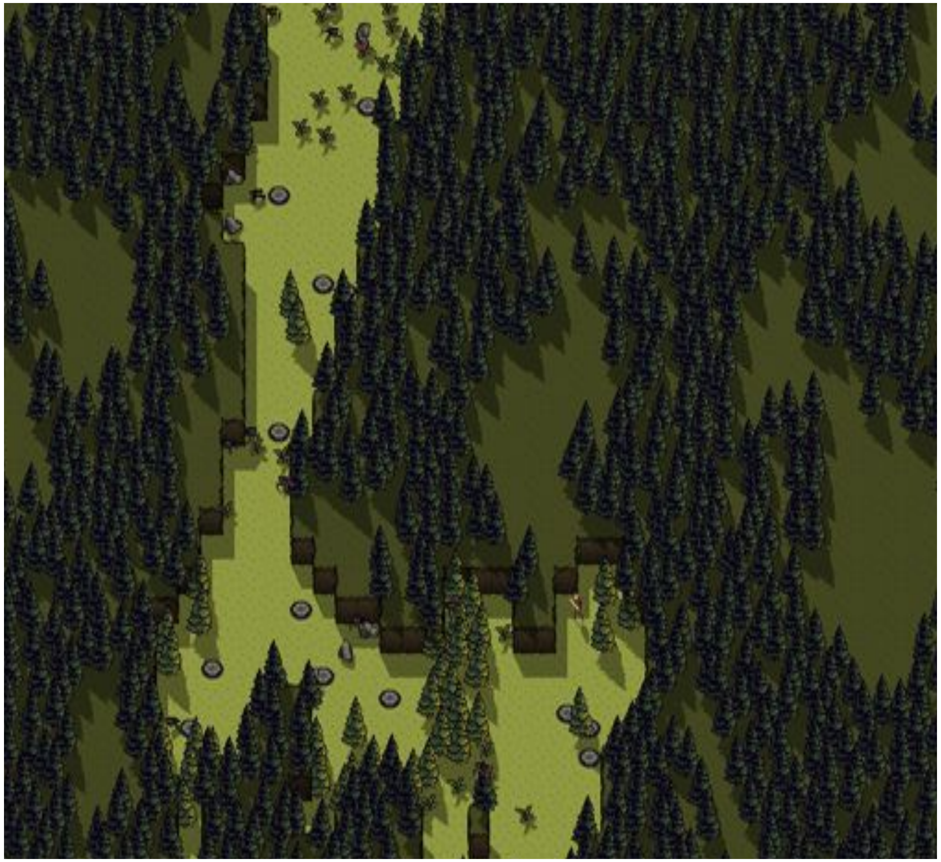


Terrain



Level





Animations

3D animated noise:
https://www.youtube.com/watch?v=4KOJiQ4jZhY

3D clouds:
https://www.shadertoy.com/view/XslGRr

Advanced noise optimization in Factorio:
https://www.factorio.com/blog/post/fff-112