matfyz

Computer Graphics Charles University

Textures and noise functions

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

Effect of a texture

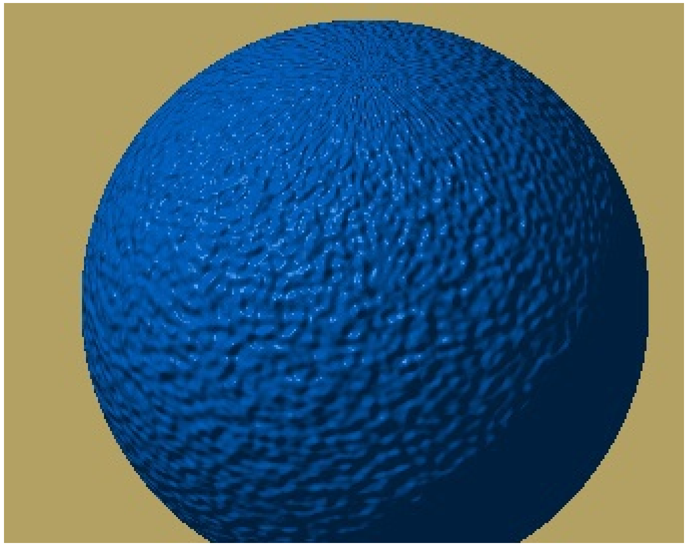
Surface **color**

Parameters of a **reflectance model**

- Phong: $\mathbf{k_D}$, $\mathbf{k_S}$, $\mathbf{h}$…

Normal vector

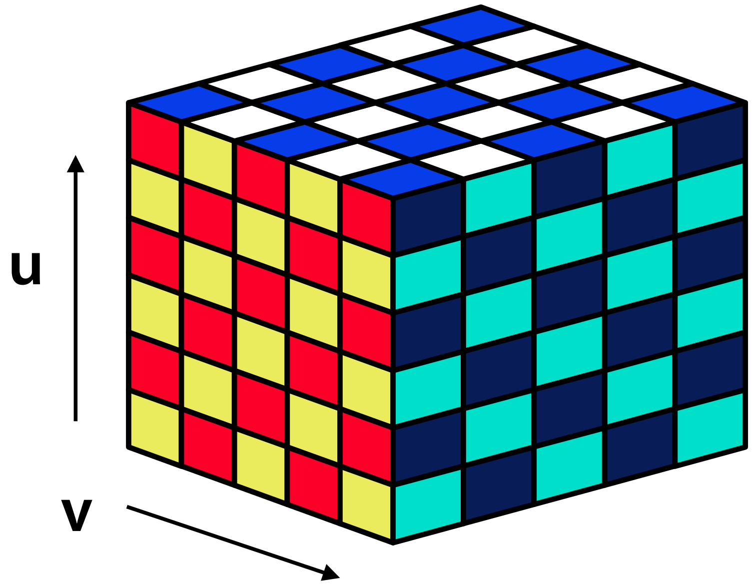
- "bump-map", normal map
- replacement for complicated geometry

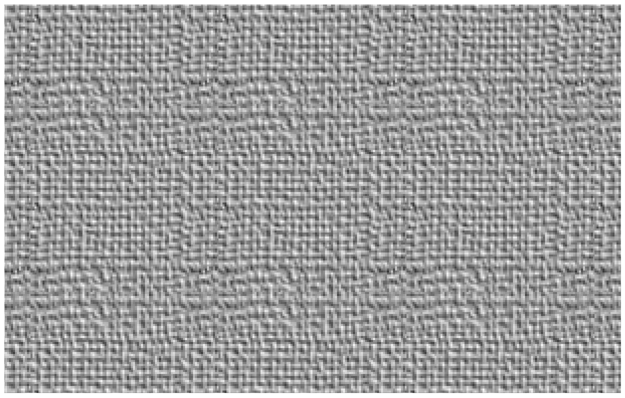


Simulation of complex **natural phenomena**

- internal structure of a material
- random textures (noise synthesis)
- fractal textures (deterministic, stochastic)

2D texture





Covers **object surface** (wallpaper)

Texture mapping: [**x, y, z**] → [**u, v**]

– "inverse mapping" function

2D texture itself: [**u, v**] → **color** (normal…)

3D texture

Represents/simulates **internal object quantities**

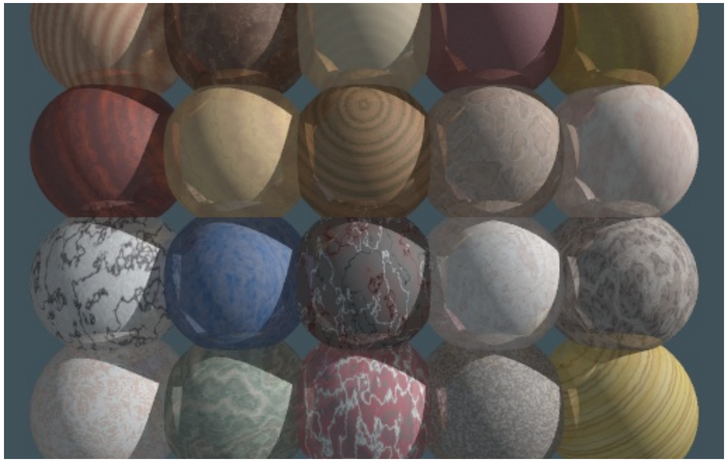
Imitates **internal material structure** (wood, marble…)

No need of **inverse mapping**

3D texture: [**x**, **y**, **z**] $\rightarrow$ **color** (reflectance, etc.)

For imitating natural materials or phenomena **noise functions** are often used

- pseudo-random continuous "folding"



Implementation types

Precomputed **data array** (table, raster image)

- often for 2D textures
- actual (natural) data, images, stickers…
- interpolation for better quality (continuity)

Algorithm-based textures (procedural)

- simple geometric shapes (checkerboard, stripes…)
- fractals, stochastic functions (noise, turbulence)

Mixed approaches (precomputed table, caching)

- computationally-intensive simulations (reaction-diffusion systems…)

Table-defined texture – near

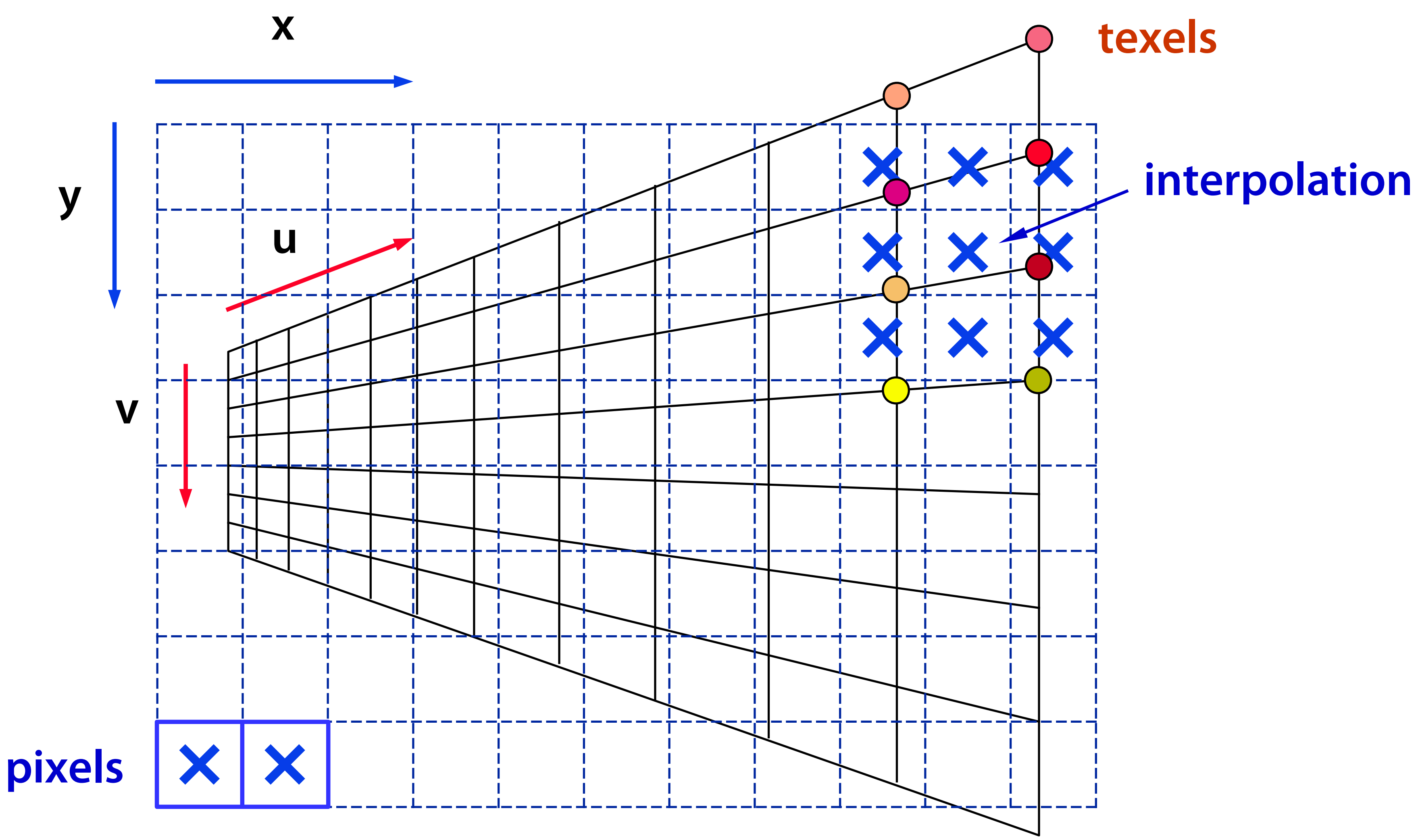
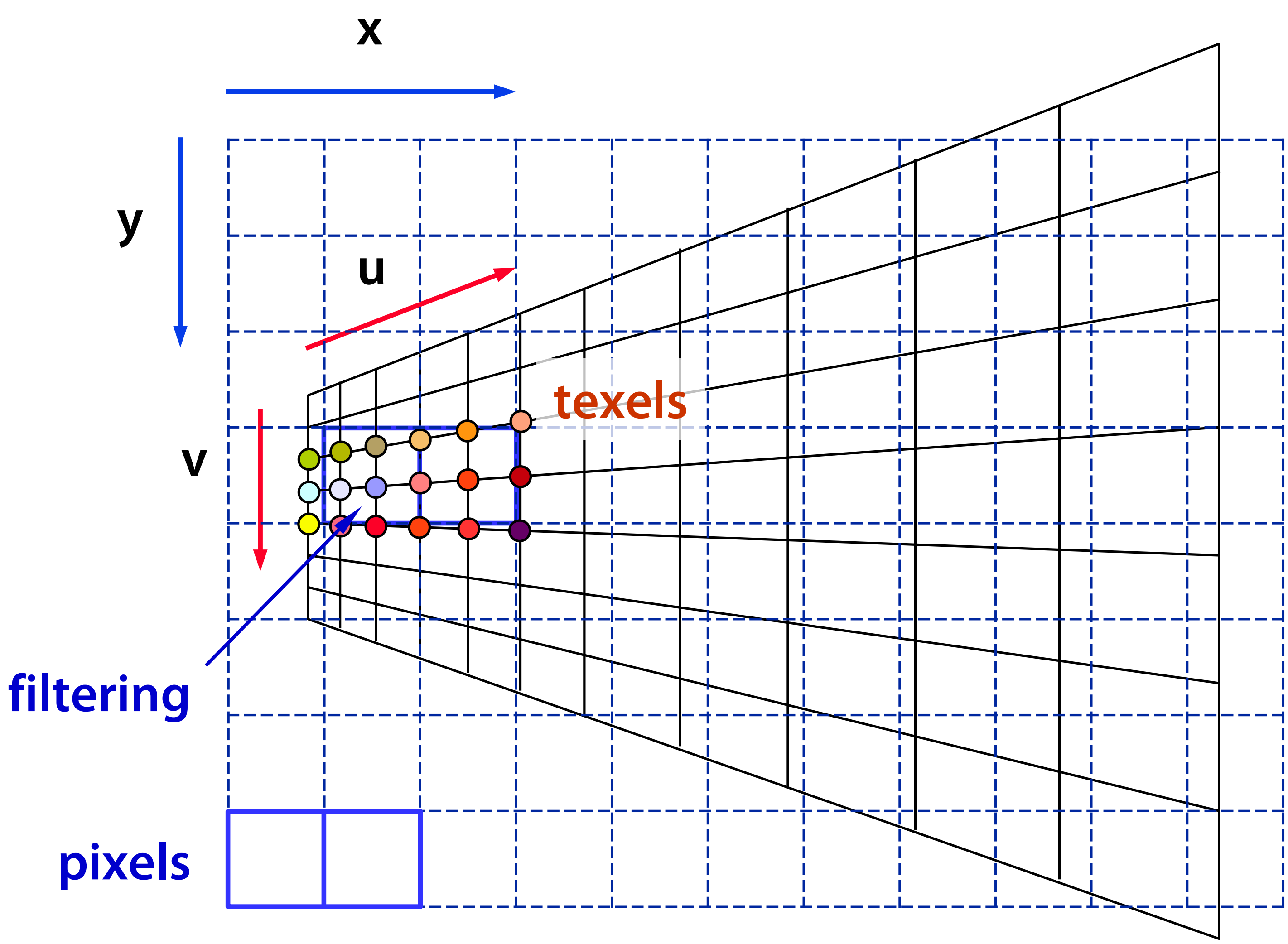


Table-defined texture – far



Interpolation types

No interpolation (rounding)

- fast and simple
- interference, pixellation artifacts (Wolfenstein 3D)

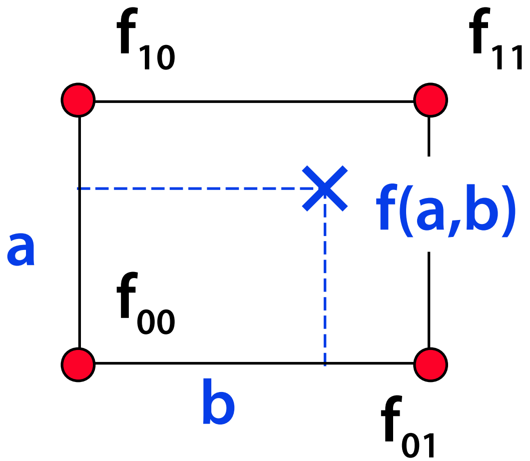
Bilinear interpolation

- **continuity** of the image function (C^0)

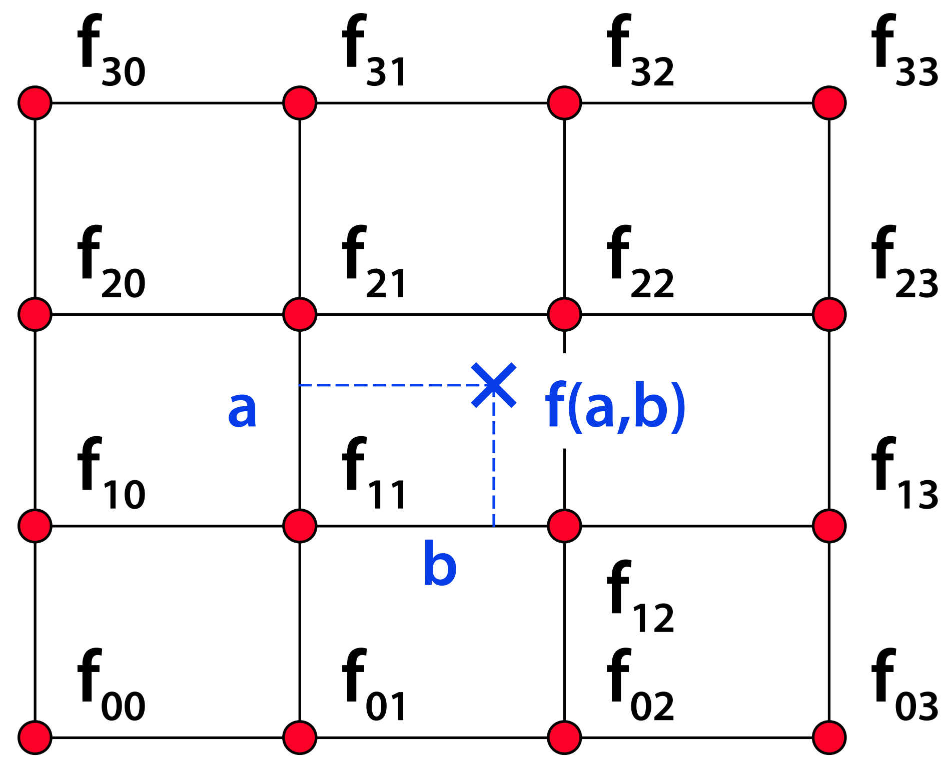
Polynomial interpolation (e.g. using spline function)

- **higher level continuity** (C^2 for bi-cubic spline)
- computing-intensive (2D case: 9-16 values has to be combined)

Bi-linear and bi-cubic interpolation



$$f(a,b) = a\cdot\left[b\cdot f_{11} + (1-b)\cdot f_{10}\right] + \\ + (1-a)\cdot\left[b\cdot f_{01} + (1-b)\cdot f_{00}\right]$$



$$f(a,b) = \sum_{i,j=0}^{3} C_i(a)\, C_j(b)\, f_{ij}$$

$C_i(t)$ … cubic polynomials

Cubic B-spline interpolation

$$f(a,b) = \sum_{i=0}^{3} \sum_{j=0}^{3} C_i(a)\, C_j(b)\, f_{ij}$$

B-spline blending functions

$$C_0(t) = \tfrac{1}{6}(1-t)^3$$

$$C_1(t) = \tfrac{1}{6}\left(3t^3 - 6t^2 + 4\right)$$

$$C_2(t) = \tfrac{1}{6}\left(-3t^3 + 3t^2 + 3t + 1\right)$$

$$C_3(t) = \tfrac{1}{6}t^3$$

Partition of unity condition

$$\sum_{i=0}^{3} C_i(t) = 1$$

$$0 \le C_i(t) \le 1 \quad \text{for} \quad 0 \le t \le 1$$

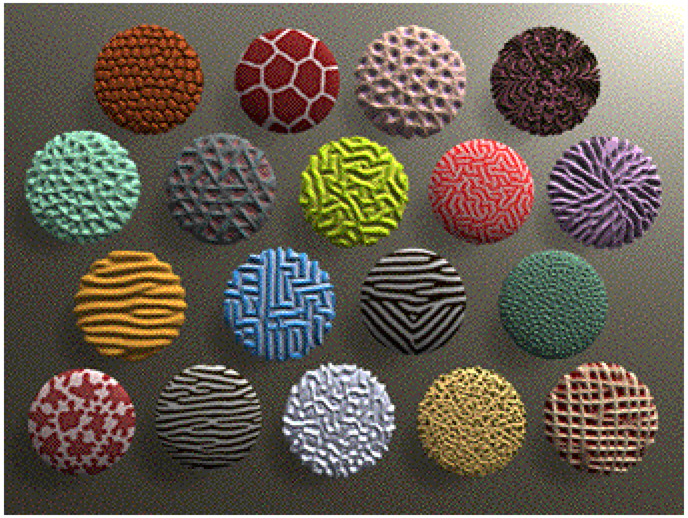
Procedural and mixed textures

Simple **geometric shapes**, patterns

- checkerboard, regular stripes, stars…

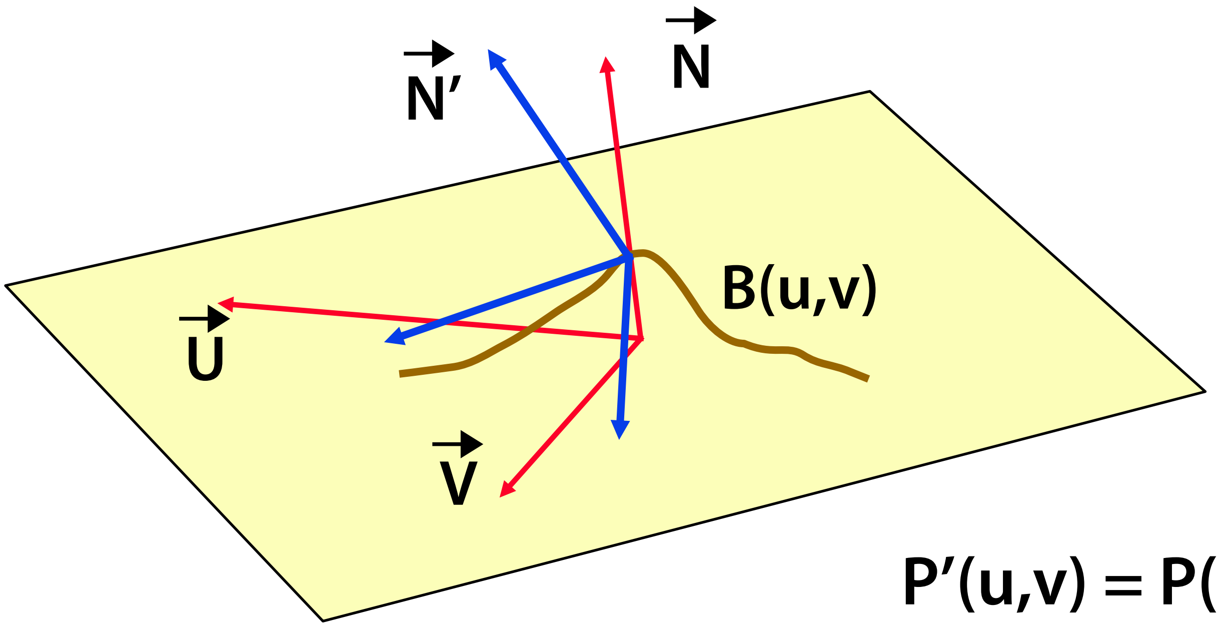
Imitation of a **natural processes**

- **pseudo-random methods** are often used (noise synthesis)
- fractals, turbulence (clouds, dirt…)
- reaction-diffusion (animal skin and fur patterns)
- 3D random perturbation textures (wood, marble…)



Reaction-diffusion
© Andy Witkin

Normal modulation (“bump map”)



$$\vec{N} = \vec{U} \times \vec{V}$$

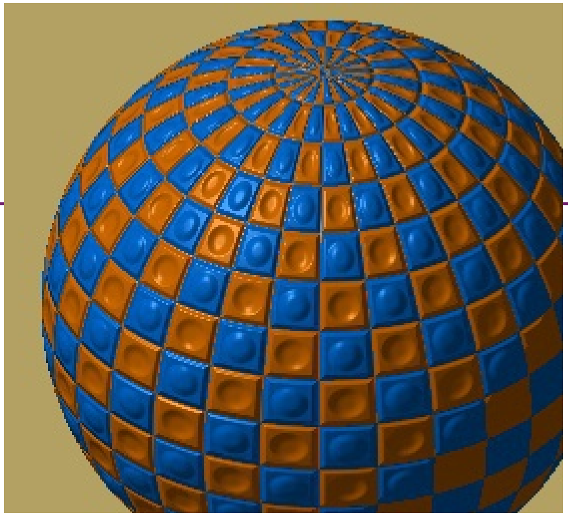
$$P'(u,v) = P(u,v) + B(u,v) \cdot \vec{N} \,/\, \|\vec{N}\|$$

Imitation of **object surface roughness**/bumpiness

B(u,v) – local surface displacement function

+ outside, – inside

Normal modulation



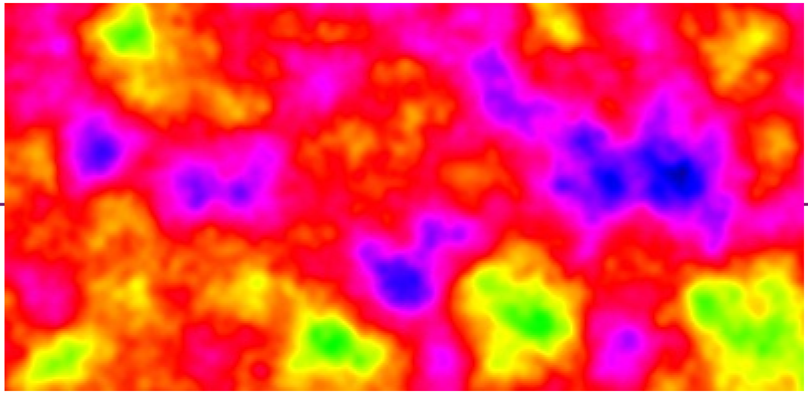
Original normal $$\vec{N} = \vec{U} \times \vec{V}$$

Moved point $$P'(u,v) = P(u,v) + \frac{B(u,v) \cdot \vec{N}}{|N|}$$

Approximation of a **modified normal vector**

$$\vec{N}' = \vec{N} + \frac{\frac{\partial B}{\partial u}(u,v) \cdot (\vec{N} \times \vec{V}) - \frac{\partial B}{\partial v}(u,v) \cdot (\vec{N} \times \vec{U})}{|N|}$$

Noise synthesis



Subjectively plausible appearance / shape

- imitation of complex natural phenomena
- chaotic system results, random diffusion, systems with [partial] feedback…

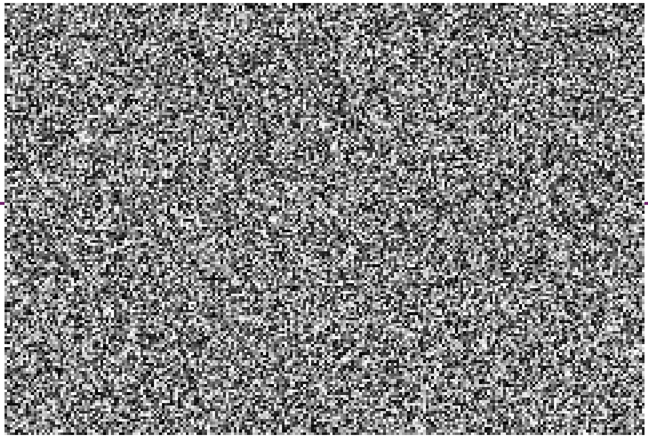
Computing noise function value in a specific point has to be **deterministic** (repeatable)

- distributed computing, super-sampling…

Required **spectral characteristics** of a noise (optional)

- uncorrelated (white) noise, frequency-limited noise…

White noise



Noise with **unlimited spectrum**

– no correlation of result values

Example of **deterministic white-noise generator**

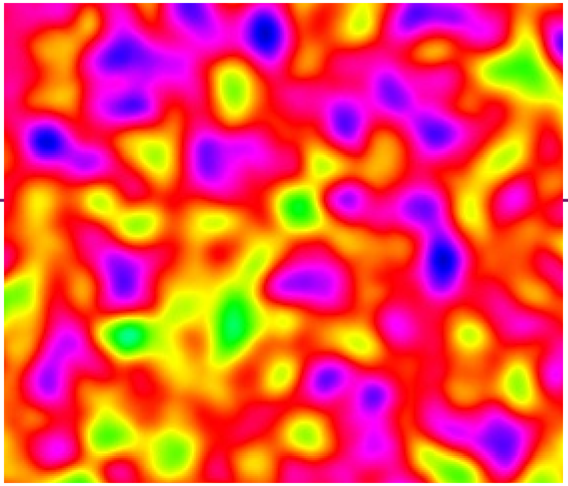
```
double RandomTab[RANDOM_TAB_LEN];  // random values
int Indx[ILEN], Indy[ILEN];        // random permutations

double white_noise_2D (double x, double y)
{
  int i = HASH(Indx[LOW_BITS(x)],Indy[LOW_BITS(y)]);
  return RandomTab[i % RANDOM_TAB_LEN];
}
```

`LOW_BITS` … extracts **k** lowest mantissa bits

`HASH` … hash function

`RandomTab`,`Indx`,`Indy` … precomputed tables

Continuous noise



Continuous function with limited spectrum

- stationary, isotropic (translation- & rotation- invariant)
- too short period could be a problem

Fourier synthesis

- tight control of frequency characteristics

Interpolation of random grid values

- classics – B-splines
- Hermite interpolation – gradients (Perlin)
- stochastic sample set – sparse convolution (Lewis)

Regular grid interpolation

- ❶ Pre-processing – a regularly distributed system of **pseudorandom values** (vectors, tangents, Jacobians)
  - required target probability densities
  - 1D, 2D or 3D topology
  - multi-dimensional case – memory saver using a hash function, see `HASH(x,y,z)`
- ❷ **Interpolation** in all other points
  - separable methods (independent coordinate components)
  - quadratic or cubic blending polynomials
  - **2D**: 4 to 16 points, **3D**: 8 to 64 points

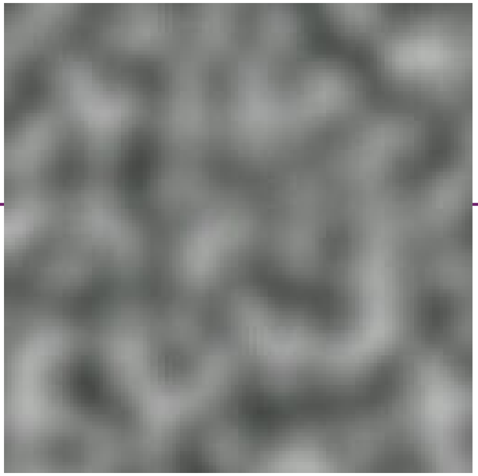
Ken Perlin's noise (3D noise)

Spectrum is limited (one octave = **f ÷ 2f**)

- efficient implementation

- Precomputed **grid of pseudo-random gradient vectors $[a,b,c,d]_{ijk}$**
  - $[a,b,c]_{ijk}$ is random **unit direction** (rejection sampling of the unit sphere)
  - d_{ijk} is noise value of the grid point $[x_i,y_j,z_k]$
  - support value $d'_{ijk} = d_{ijk} - a_{ijk} \cdot x_i - b_{ijk} \cdot y_j - c_{ijk} \cdot z_k$

Perlin's noise



❷ **Grid values**

$K_{ijk}(x,y,z) = d'_{ijk} + a_{ijk} \cdot x + b_{ijk} \cdot y + c_{ijk} \cdot z$

❸ **Interpolation** cubic splines

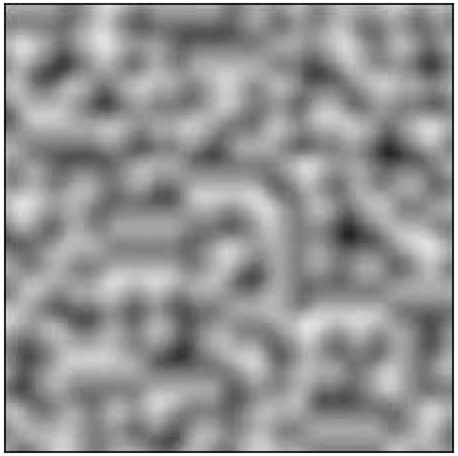
$w(t) = 2|t|^3 - 3t^2 + 1$ for $|t| < 1$

$w(t) = 0$ else

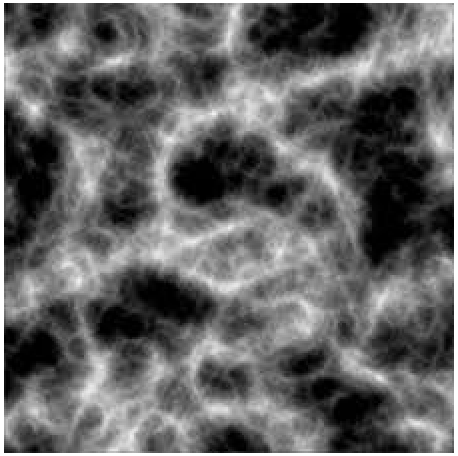
– support radius = **1** ⇒ I need only 2^D grid points

$$a(x, y, z) = \sum_{i=\lfloor x \rfloor}^{\lfloor x \rfloor + 1} w(x - i) \sum_{j=\lfloor y \rfloor}^{\lfloor y \rfloor + 1} w(y - j) \sum_{k=\lfloor z \rfloor}^{\lfloor z \rfloor + 1} w(z - k) \cdot a_{ijk}$$

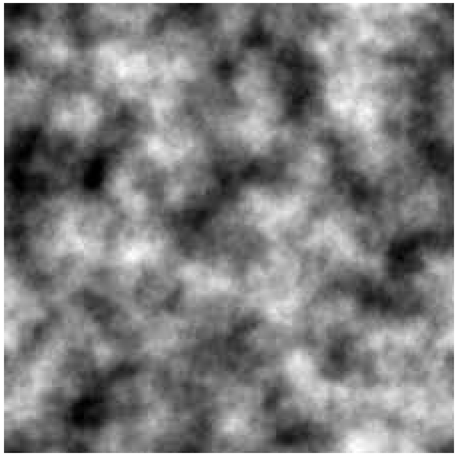
Perlin noise examples



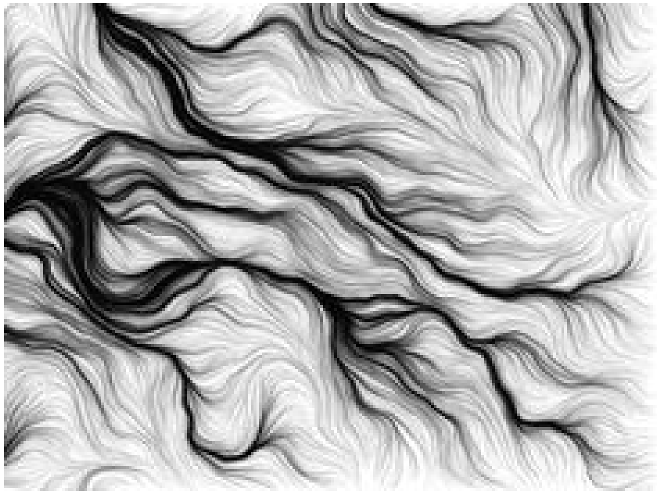
basic noise



sum(2) ?



turbulence



noise
streams

Sparse convolution (Lewis)

Controlled spectral characteristics

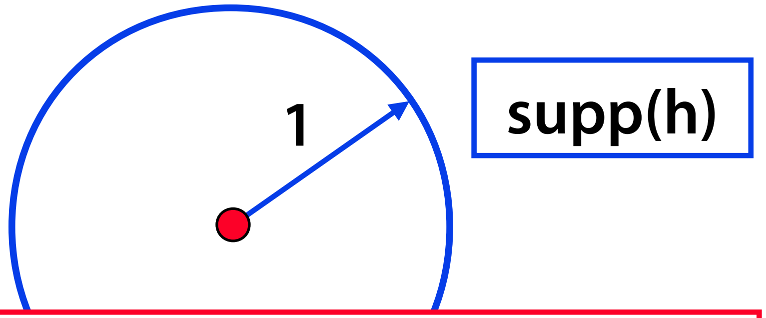
- efficient (scalable) implementation

Convolution of 3D filter **h(x,y,z)** with **Poisson noise** γ

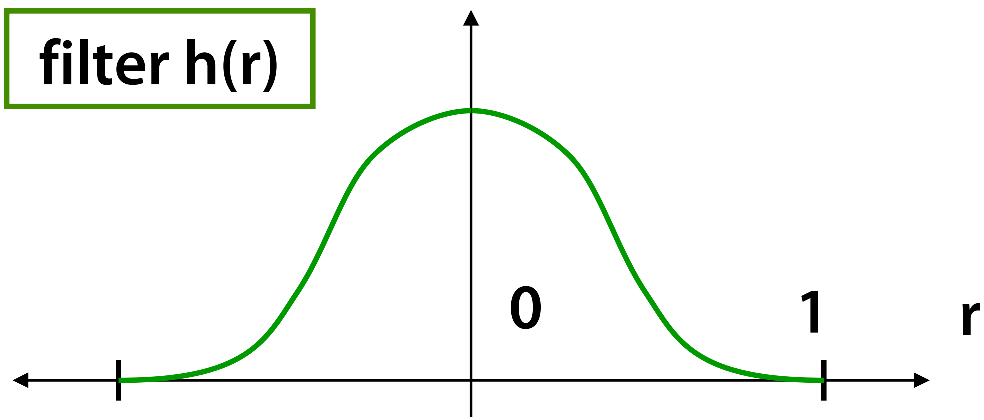
$$n(x,y,z) = \int_{\Re^3} \gamma(u,v,w) \cdot \underline{h(x-u,y-v,z-w)} \, du\, dv\, dw$$

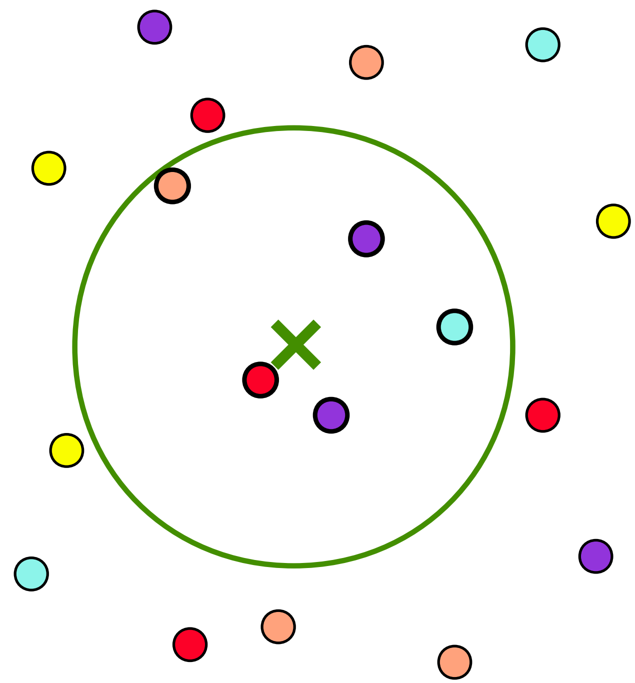
$$\gamma(x,y,z) = \sum_k \underline{a_k} \cdot \delta(x-\underline{x_k}, y-\underline{y_k}, z-\underline{z_k})$$

Poisson noise convolution



random position: $[x_k, y_k, z_k]$
random amplitude: a_k





h(r) … Radial Basis Function

Sparse convolution

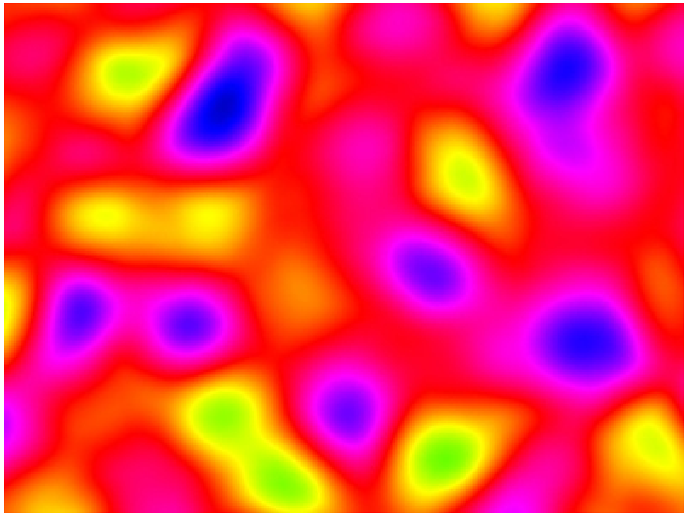
Thanks to discrete nature of a Poisson noise

$$n(x, y, z) = \sum_k \underline{a_k} \cdot h(x - \underline{x_k}, y - \underline{y_k}, z - \underline{z_k})$$

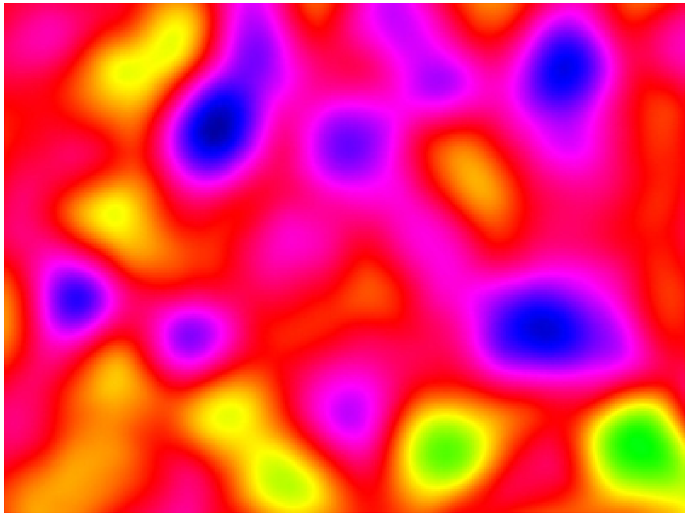
Sample density $[x_k, y_k, z_k]$ controls the result quality of the noise

- for **10+** samples per **supp(h)** the quality is indistinguishable from an interpolation noise
- sparse convolution can have higher efficiency for normal quality

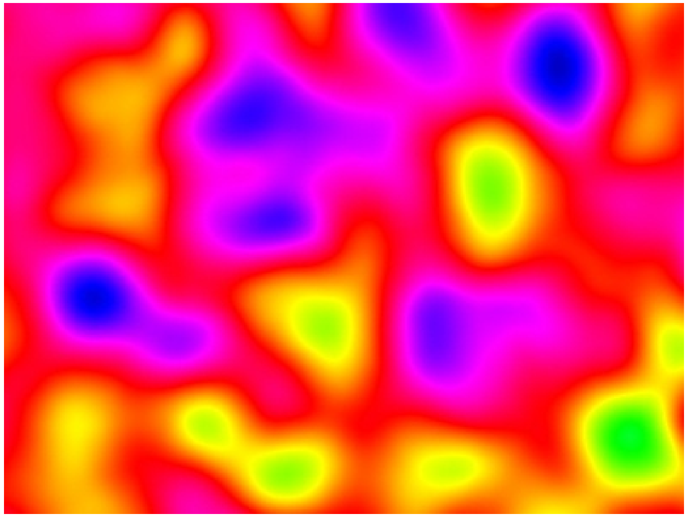
Different sample densities



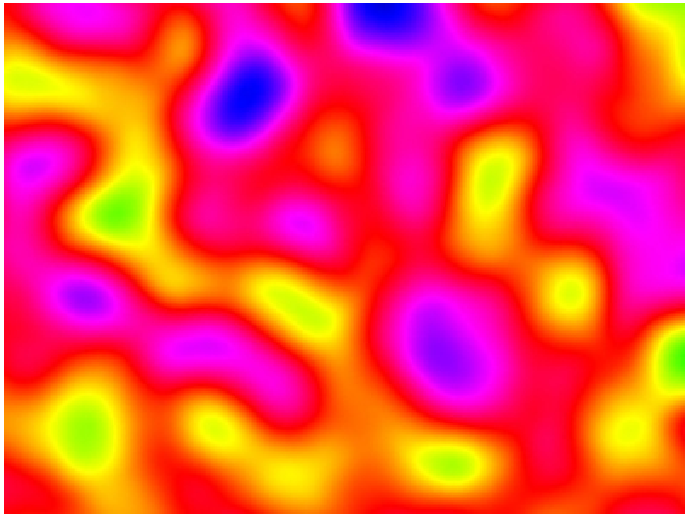
2 samples per cell



3 samples per cell



6 samples per cell



10 samples per cell

Efficient implementation

Space division scheme – grid with **cell size = r**

- filter **supp(h)** radius – usually **r = 1**

Each grid cell generates its samples independently, using **pseudo-random generator** initialized to **$Seed_{ijk}$**

- **$Seed_{ijk}$** values are prepared in advance by a differrent random source
- or some **hash function** can be used: `HASH(x,y,z)`

Efficient implementation trick

For the result **Noise(x, y, z)** we only need to process a limited number of **neighbour cells**

- 2D: **4 ÷ 9** cells
- 3D: **8 ÷ 27** cells

For an **isotropic noise** (symetrical filter function **h**) we can precompute **h**($\mathbf{r^2}$) values into a table

- no more square roots in convolution calculations

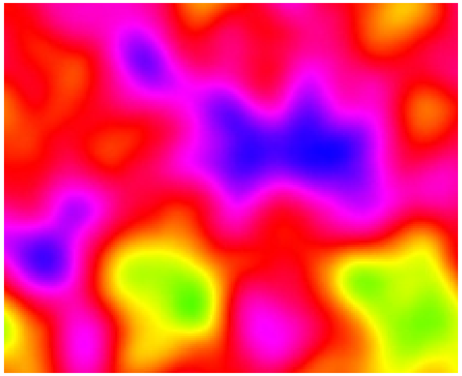
Noise function combination

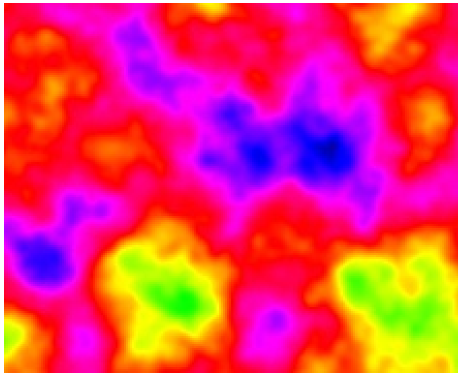
General combination of noise functions with fre-quencies $\mathbf{f_i}$, amplitudes $\mathbf{a_i}$ using drift vectors $[\mathbf{x_i}, \mathbf{y_i}, \mathbf{z_i}]$

$$\sum_i a_i \cdot \mathrm{Noise}\left[f_i \cdot (x + x_i), f_i \cdot (y + y_i), f_i \cdot (z + z_i) \right]$$

Turbulence simulation

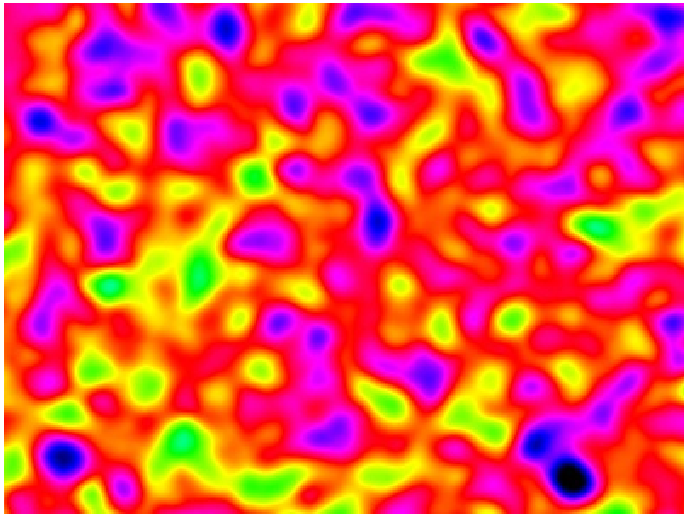
$$f_i = F^i, \; a_i = A^{-i}$$



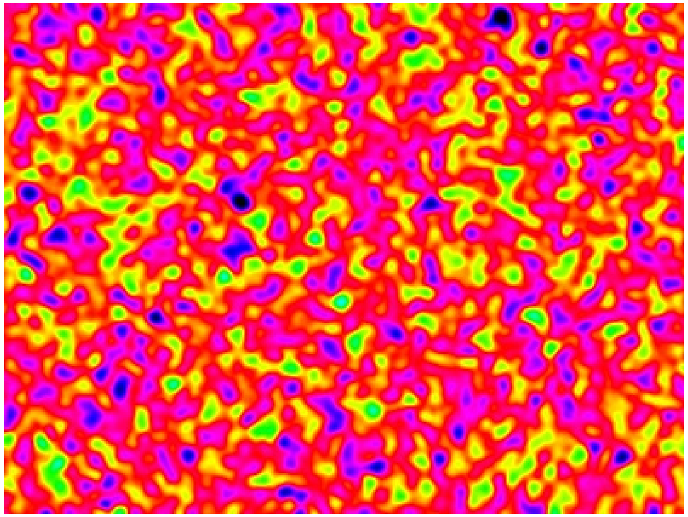


$$\sum_i \frac{1}{A^i} \cdot \mathrm{Noise}\left[F^i \cdot (x + x_i), F^i \cdot (y + y_i), F^i \cdot (z + z_i) \right]$$

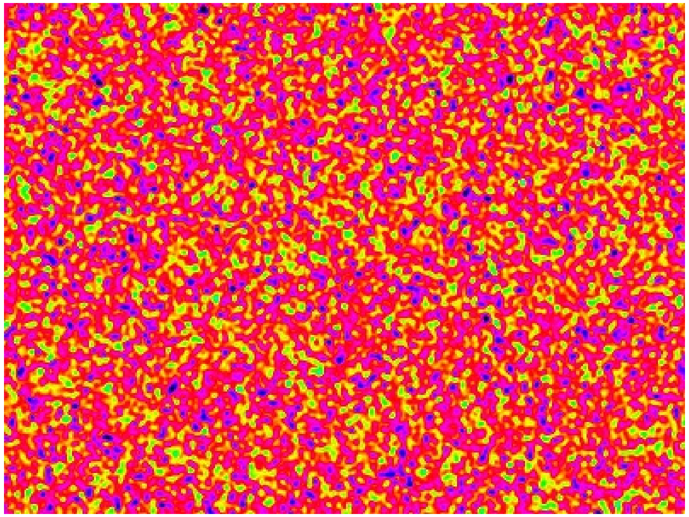
Noise frequency



cell-size = 20

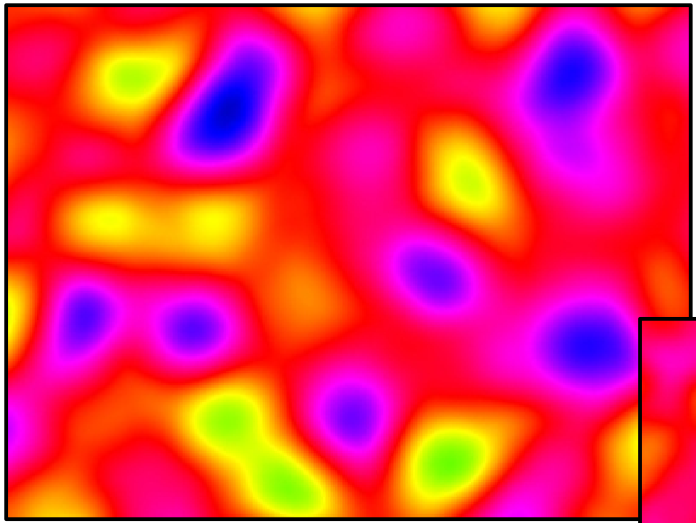


cell-size = 8.4



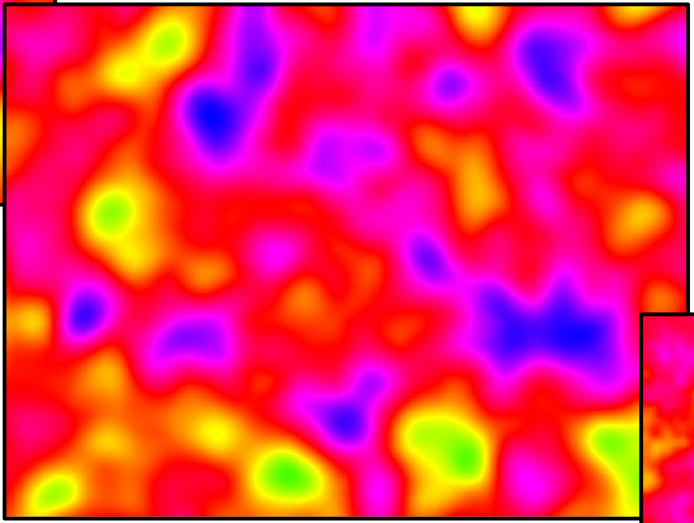
cell-size = 3.3

Turbulence

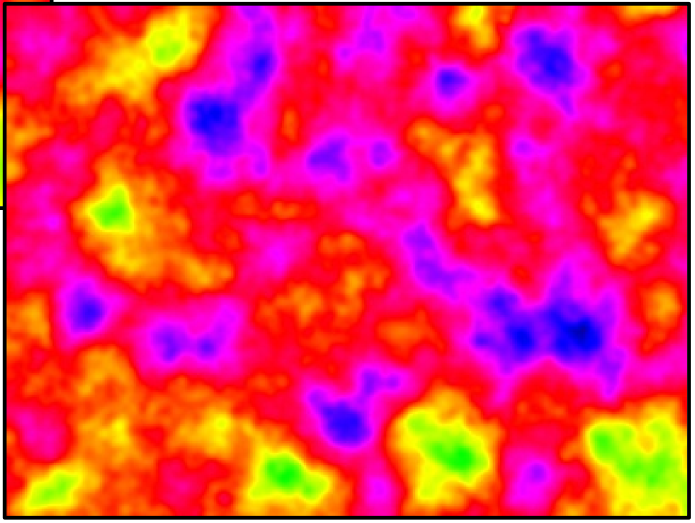


1 component
(plain noise)

2 components



4 components

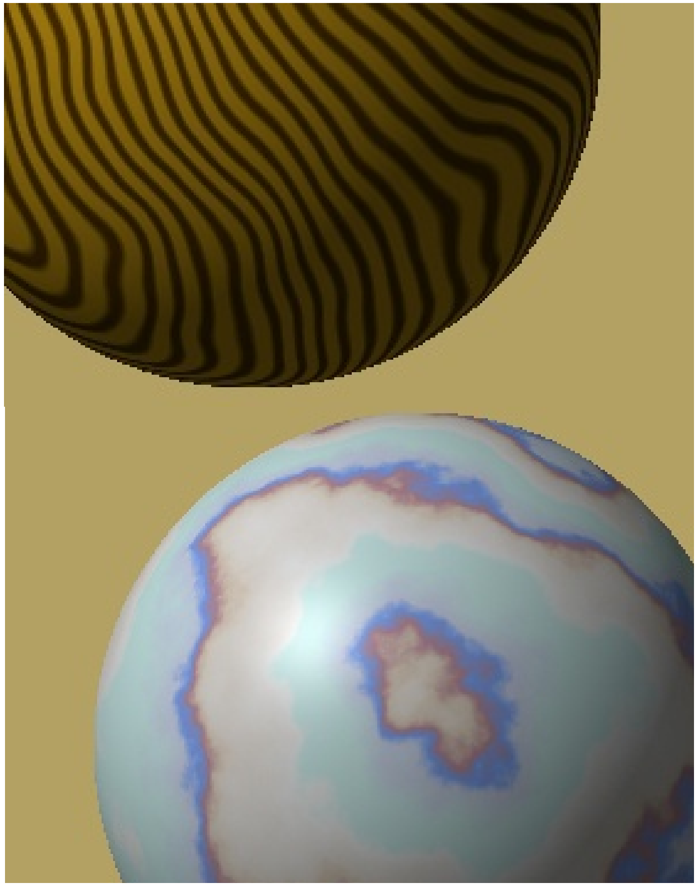


Applications

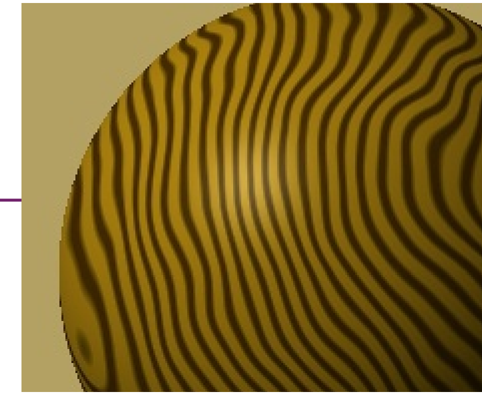
Random **normal modulations** (“bump maps”)
- illusion of randomly wrinkled object surface
- ″citrus peel”

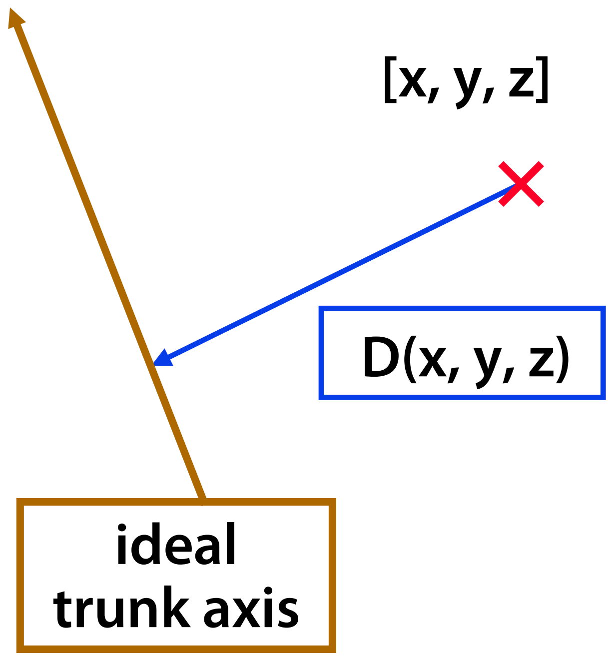
Turbulence
- fog, clouds, many other modeling

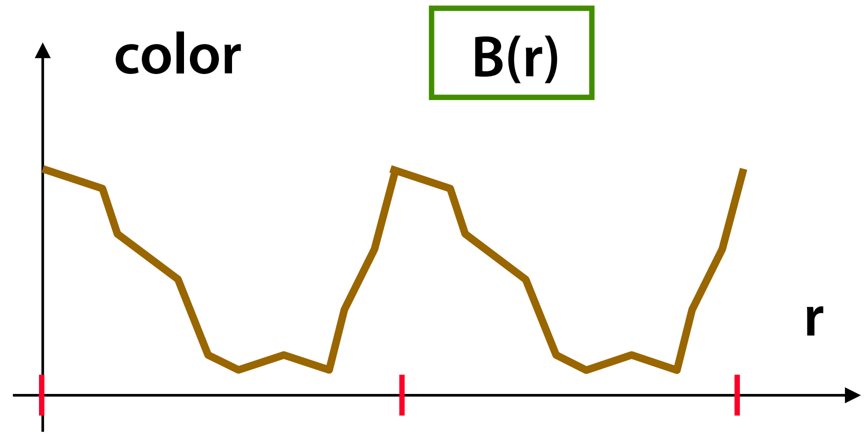
3D textures
- inner material structure
- wood, marble…



Wood imitation



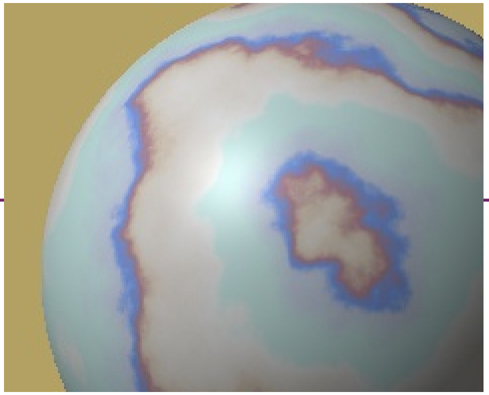


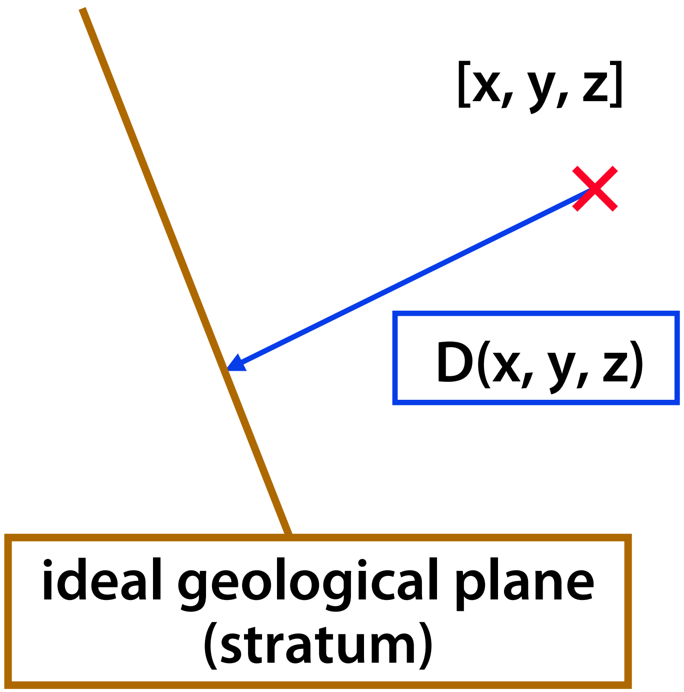


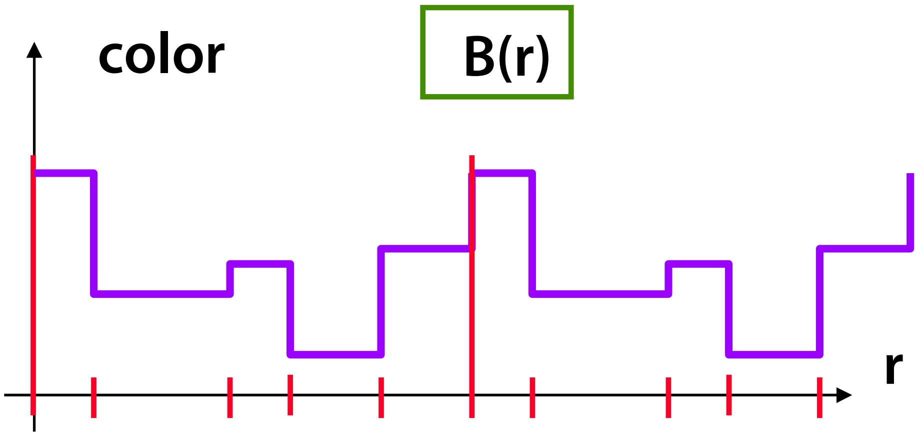
$$\mathbf{B}\left[\mathbf{D}(x, y, z) + \mathbf{Noise}(x, y, z)\right]$$

$$\mathbf{B}\left[\mathbf{D}(x, y, z) \cdot \left(1 + \mathbf{Noise}_1(x, y, z)\right) + \mathbf{Noise}_2(x, y, z)\right]$$

Marble imitation

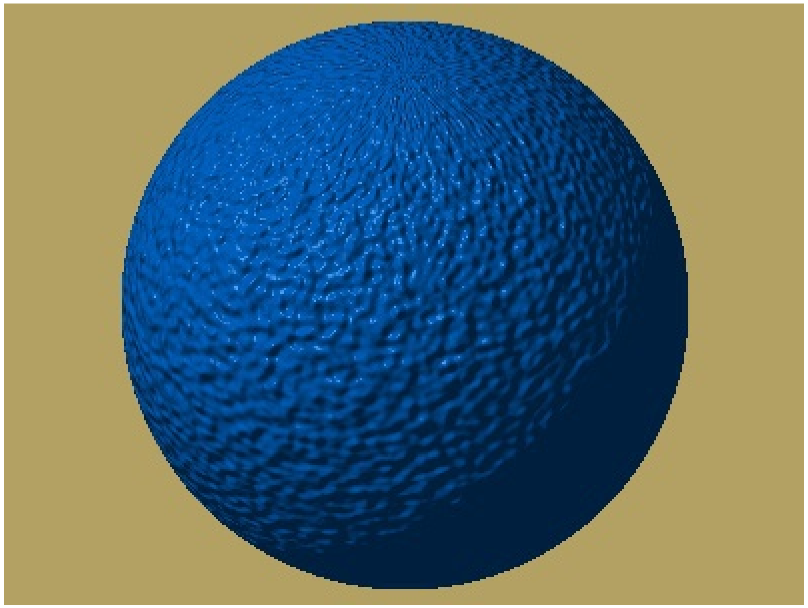






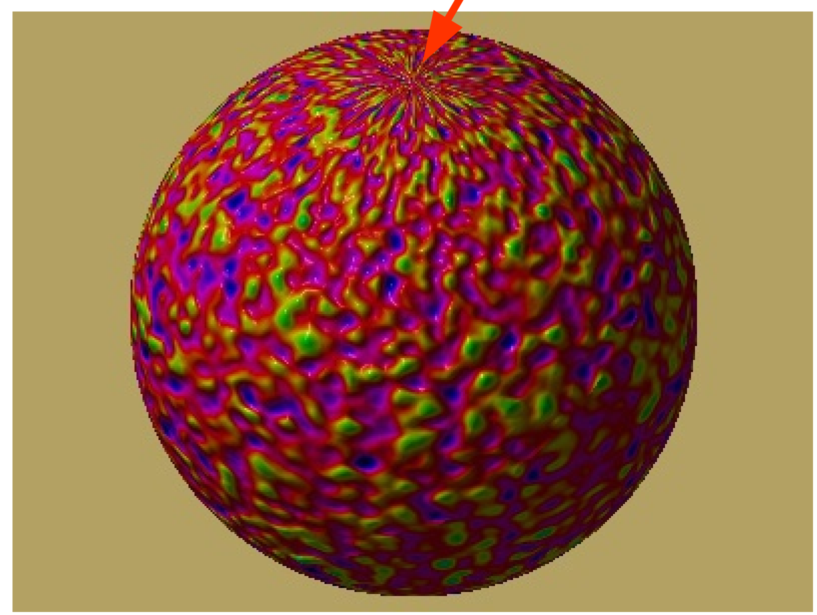
$$B\left[D(x,y,z) + \mathrm{Turb}\,(x,y,z)\right]$$

Bump noise examples



2D bump noise
(polar mapping)

pole = singularity

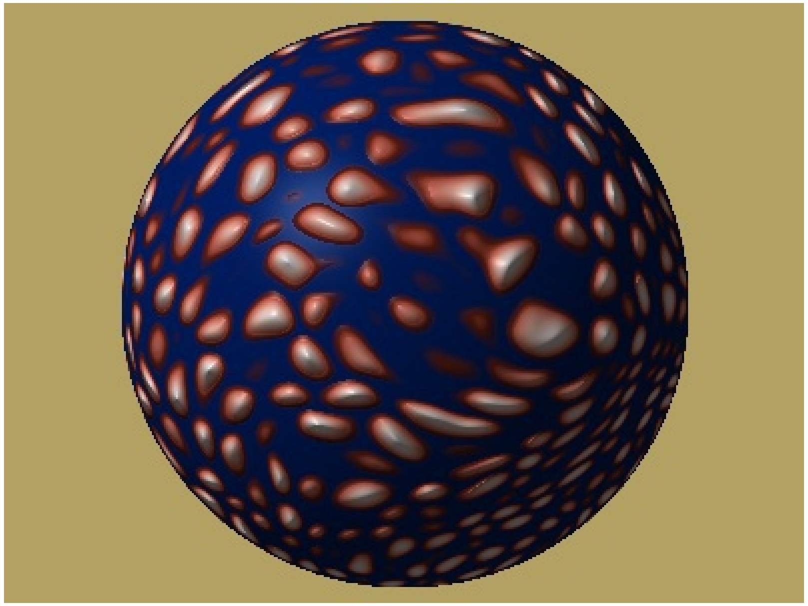


2D bump & color
(polar mapping)

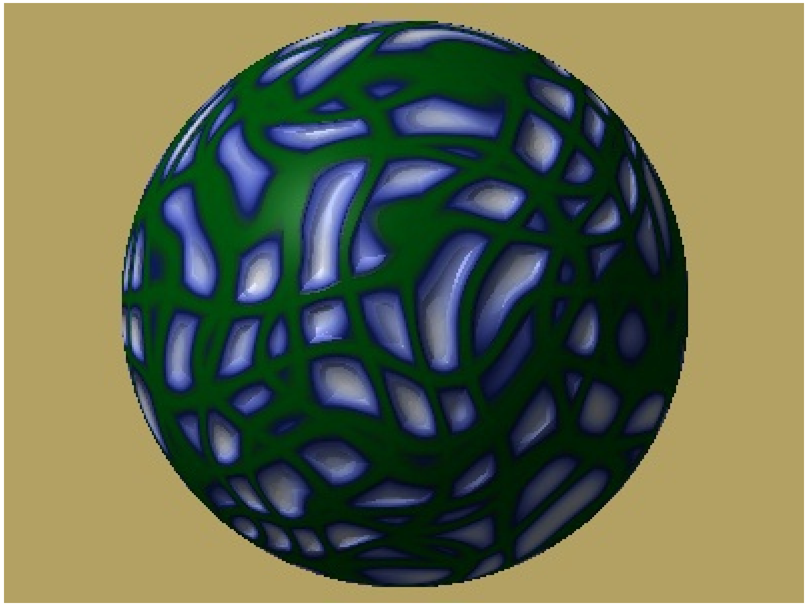
Voronoy/Worley noise

3D tesselation

- basis = cubic/tetrahedral cells
- **space warp** by 3D noise

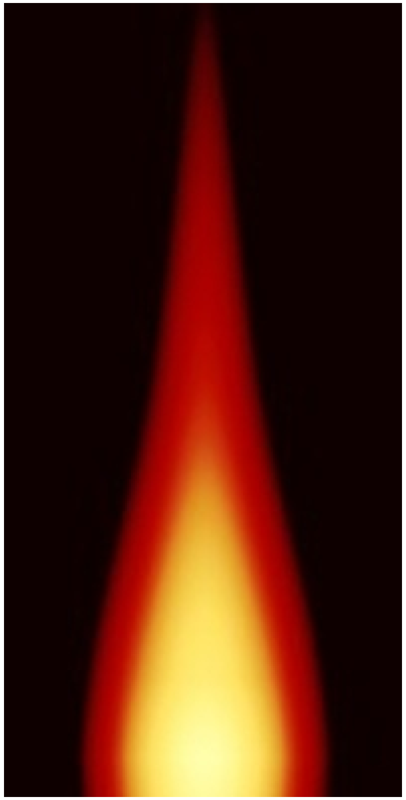


tetrahedrons



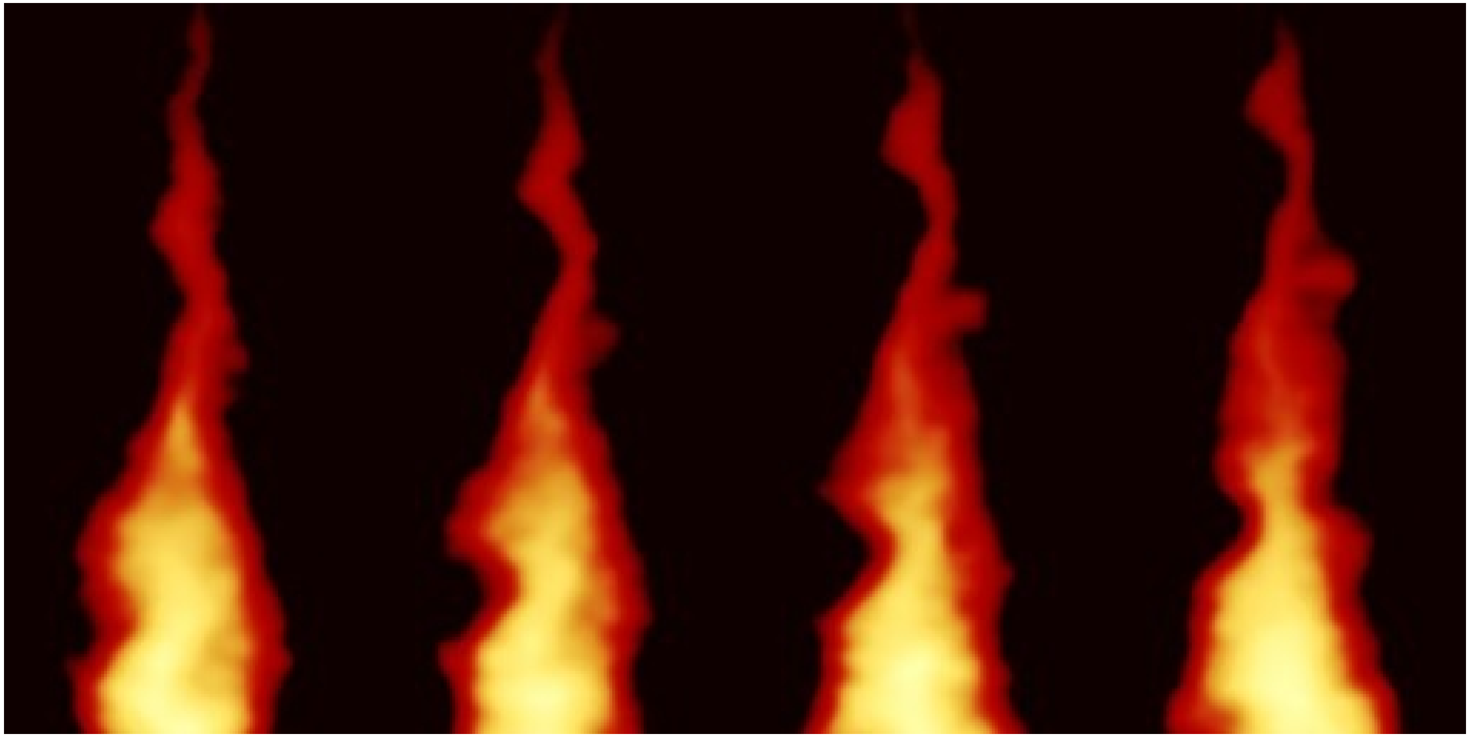
cubes

2D flames using 3D turbulence

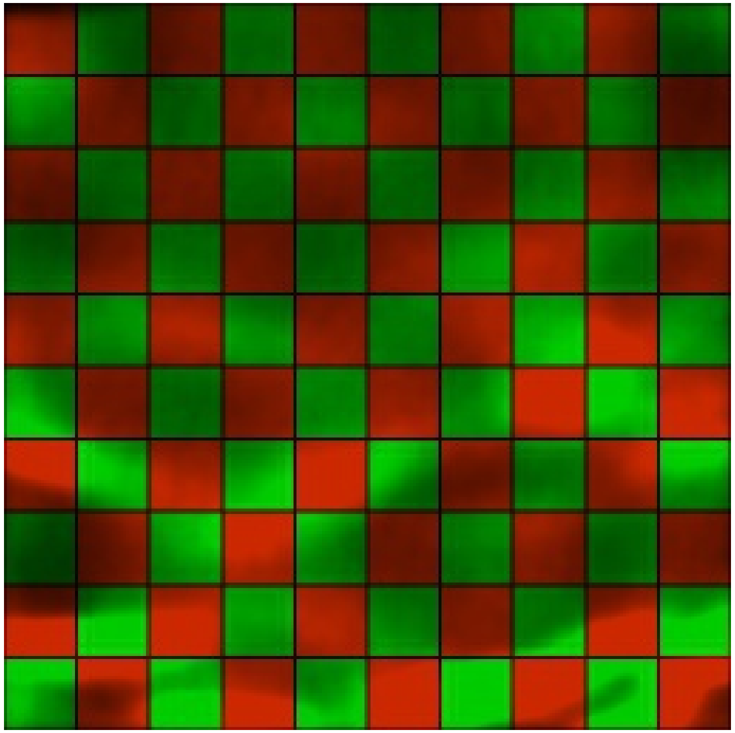


base image

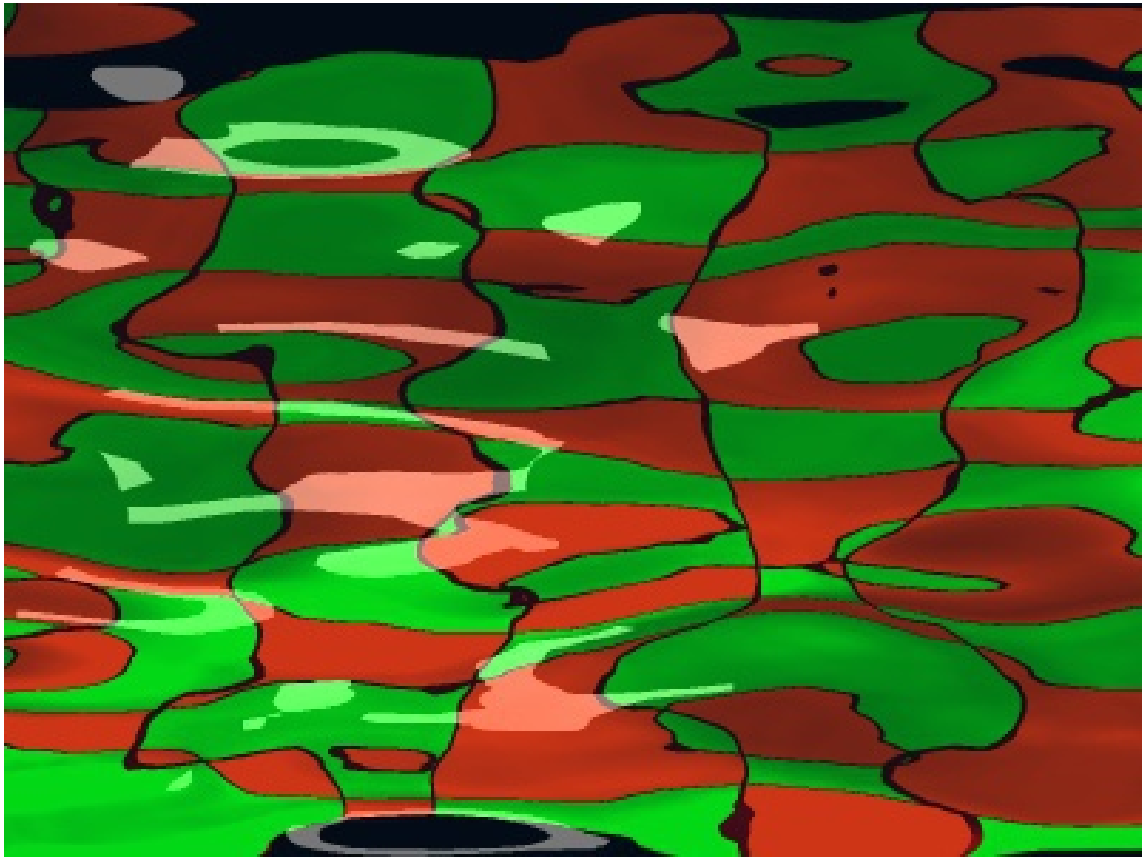
3D noise is drifting up (y axis) and far (z axis)



Caustics and waves



pre-processed caustics
(11.8M rays, 512x512px light-map)



final image
(Ray-tracing)

Literature

K. Perlin: ***An Image Synthesizer***, Computer Graphics, Vol. 19, #3, July 1985, 287-296

K. Perlin, E. M. Hoffert: ***Hypertexture***, Computer Graphics, Vol. 23, #3, July 1989, 253-262

J. P. Lewis: ***Algorithms for Solid Noise Synthesis***, Computer Graphics, Vol. 23, #3, July 1989, 263-270

J. Foley, A. van Dam, S. Feiner, J. Hughes: ***Computer Graphics, Principles and Practice***, 741-745, 1015-1018, 1043-1047