Textures and noise functions

© 1998-2017 Josef Pelikán

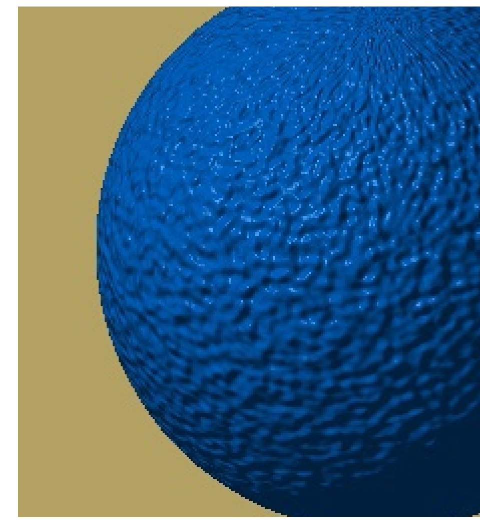
CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

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

Effect of a texture

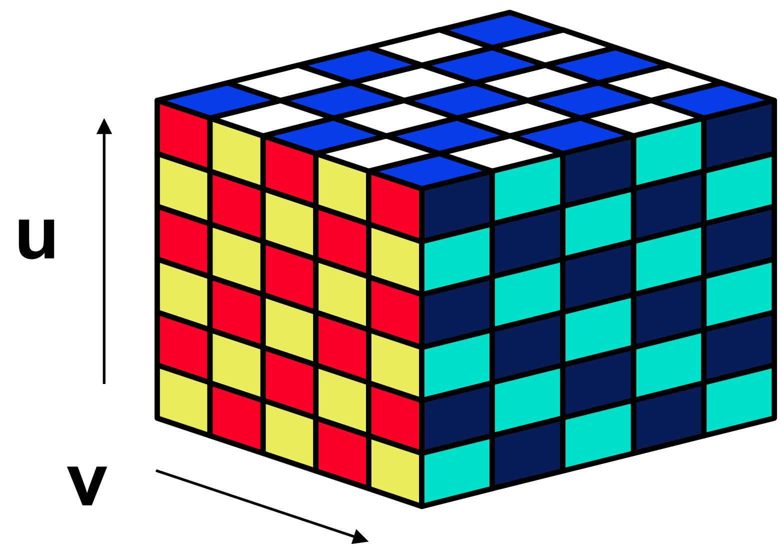
- surface **color**
- parameters of a **reflectance model**
  - Phong: k_D, k_S, h, ..
- **normal vector**
  - "bump-map", normal map
  - replacement for complicated geometry

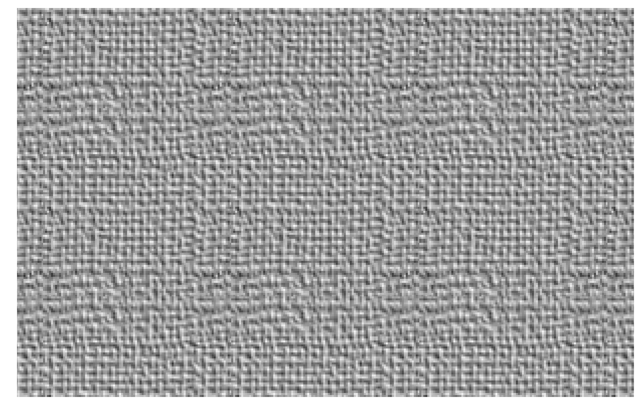


- simulation of complex **natural phenomena**
  - internal structure of a material
  - random textures (noise synthesis)
  - fractal textures (deterministic, stochastic)

© Josef Pelikán, http://cgg.mff.cuni.cz/~pepca

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, ...)

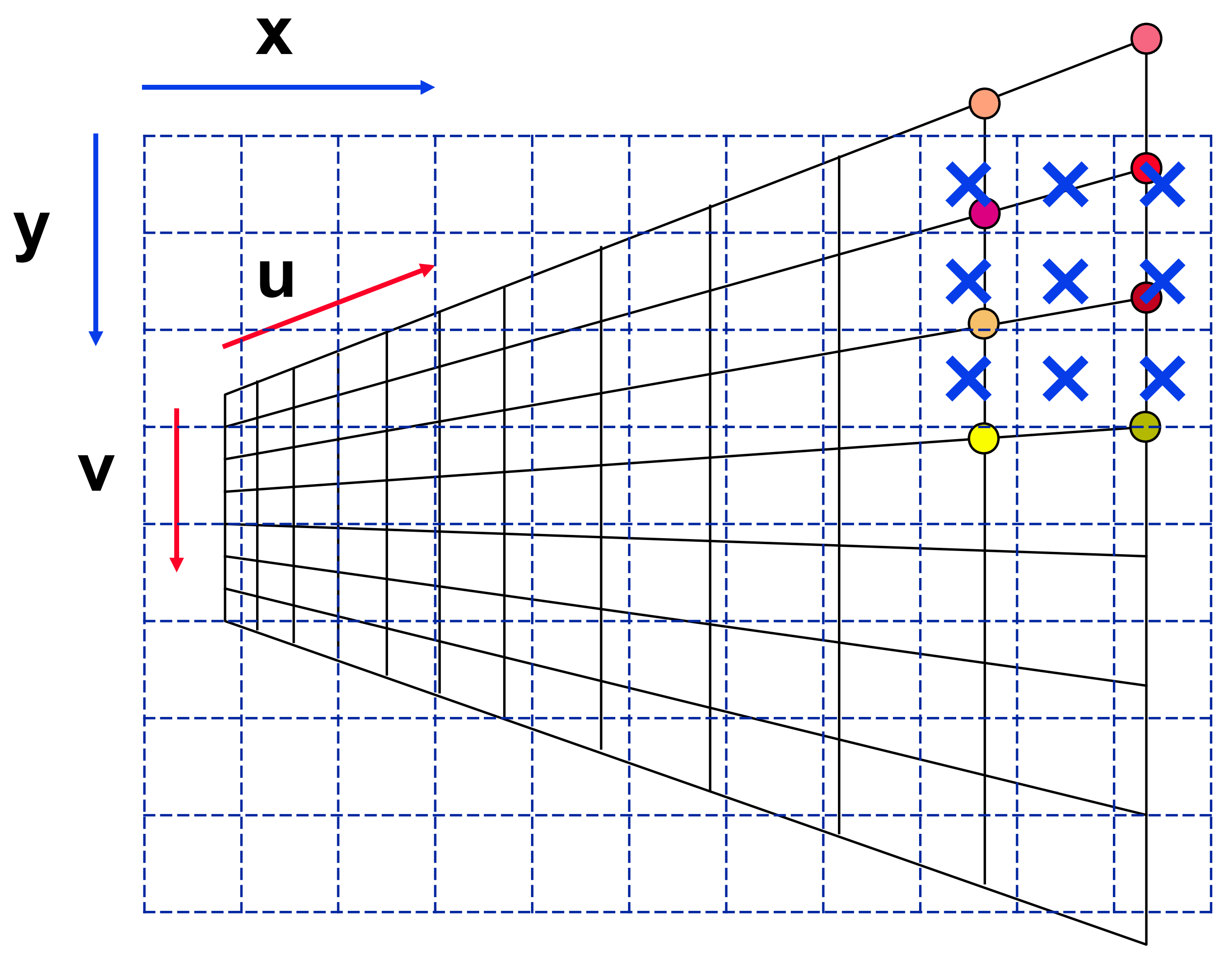
- no need of **inverse mapping**
- **3D texture**: [**x**,**y**,**z**] → **color** (reflectance, etc.)

- 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, ..)

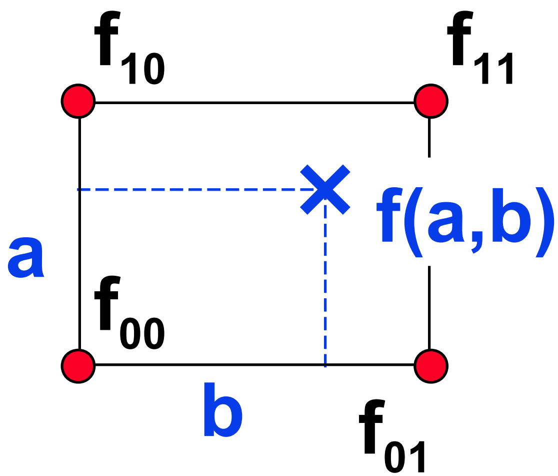
Table-defined texture



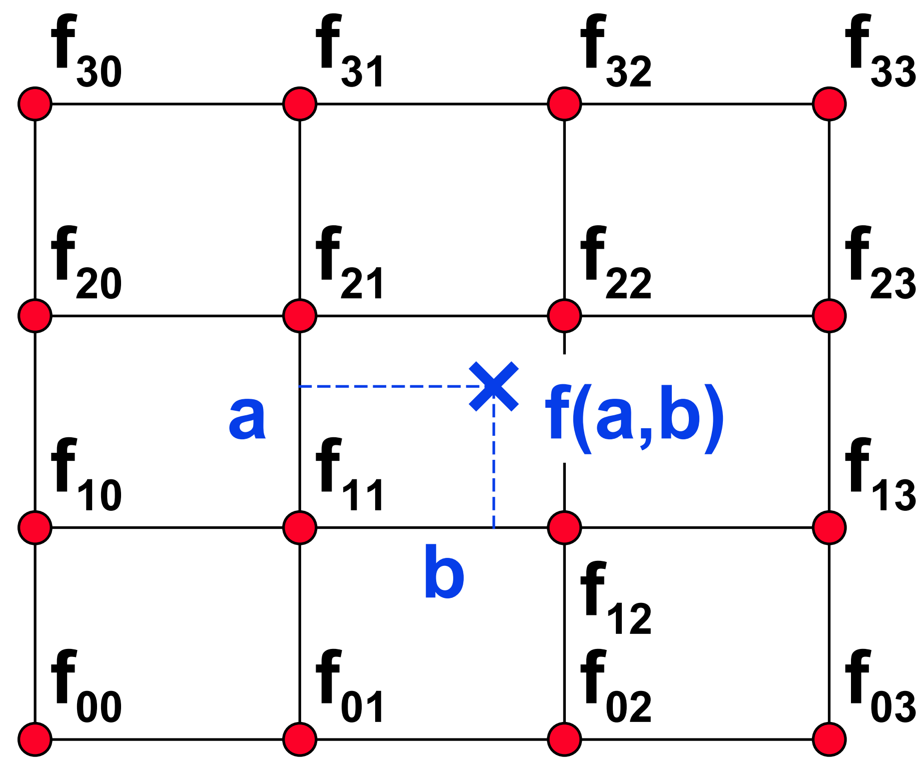
Interpolation types

- **w/o 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:

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

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

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

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

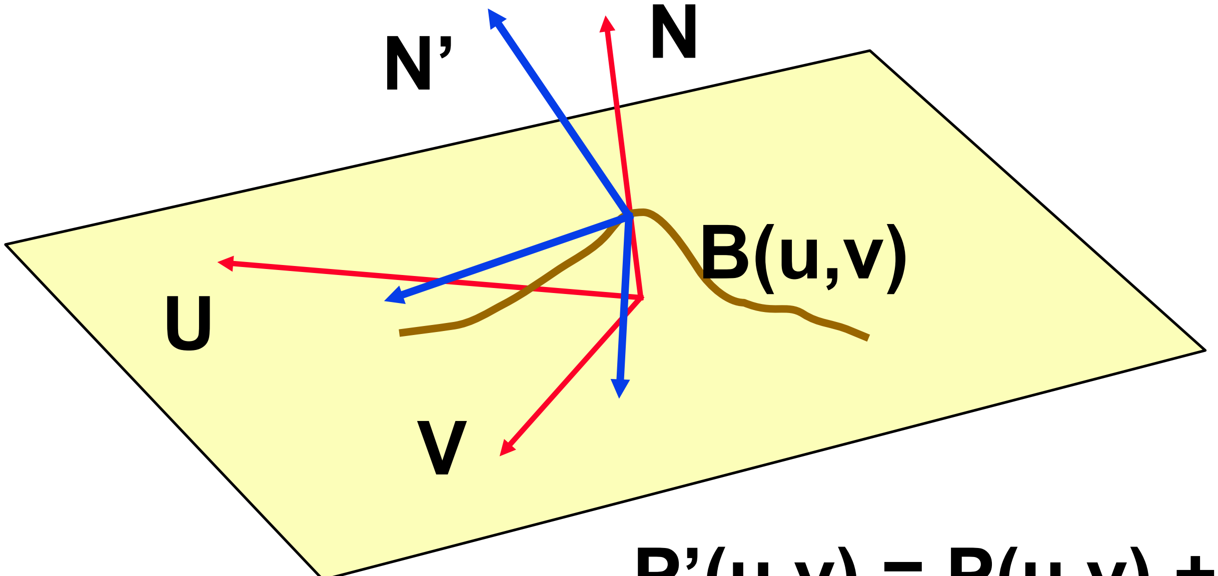
$$\sum_{i=0}^{3} C_i(t) = 1$$

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

Procedural and mixed textures

- simple **geometric shapes**, patterns
  - checkerboard, regular stripes, stars, ..
- imitation of **natural processes**
  - often **pseudo-random methods** are used (noise synthesis)
  - fractals, turbulence (clouds, dirt, ..)
  - reaction-diffusion (animal skin and fur patterns)
  - 3D random perturbation textures (wood, marble, ..)

Normal modulation (“bump map”)

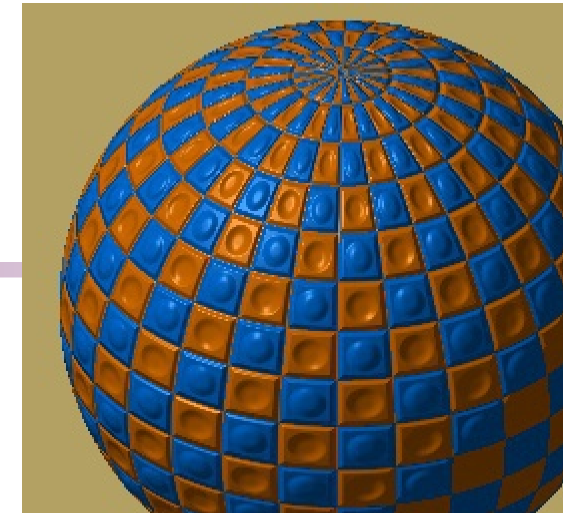


$$\mathbf{N} = \mathbf{U} \times \mathbf{V}$$

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

- imitation of **object surface roughness**/bumpiness
- **B(u,v)** – local surface displacement function: + outside, – inside

Normal modulation



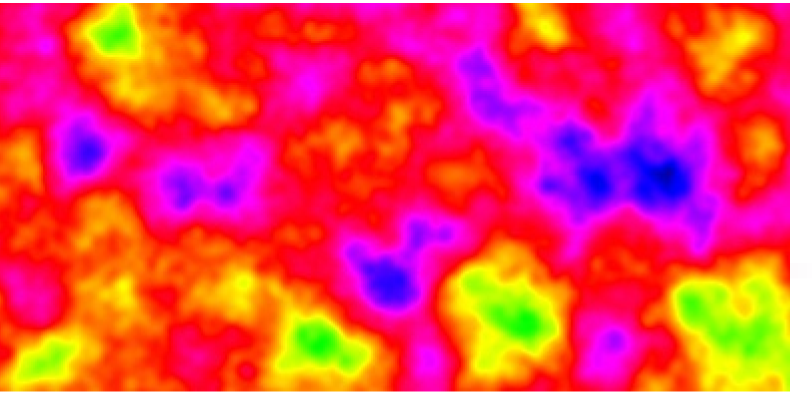
original normal: $\mathbf{N} = \mathbf{U} \times \mathbf{V}$

moved point: $\mathbf{P}'(\mathbf{u}, \mathbf{v}) = \mathbf{P}(\mathbf{u}, \mathbf{v}) + \frac{\mathbf{B}(\mathbf{u}, \mathbf{v}) \cdot \mathbf{N}}{|\mathbf{N}|}$

approximation of **modified normal vector**:

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

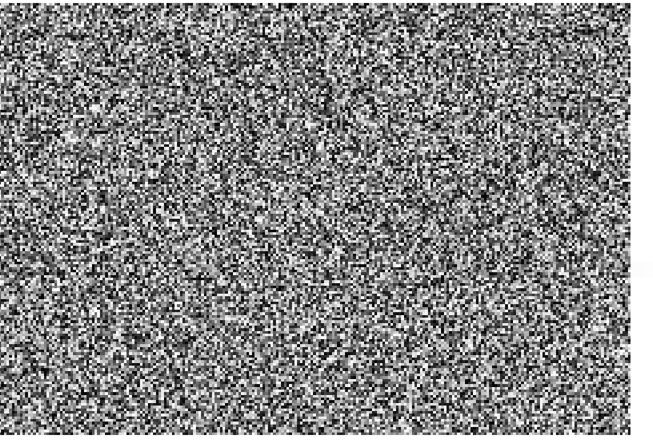
Noise synthesis



- **subjectively plausible** appearance / shape
  - imitation of complex natural phenomena
  - chaotic system results, random diffusions, 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 (opt.)
  - uncorrelated (white) noise, frequency-limited noise, ..

© Josef Pelikán, http://cgg.mff.cuni.cz/~pepca

White noise



- noise with **unlimited spectrum**
  – no correlation of any two result values

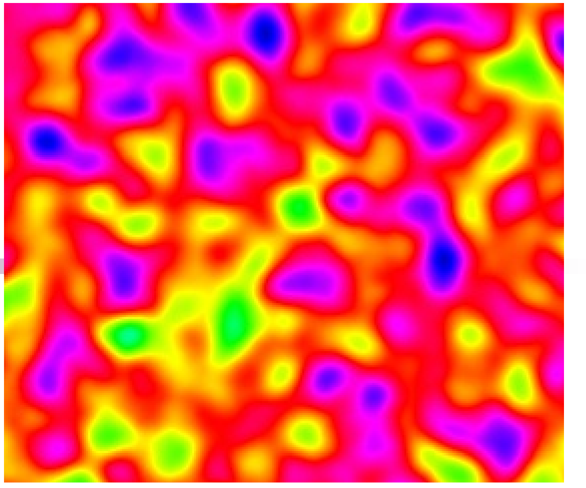
➡ example of deterministic white-noise generator:

```
double RandomTab[RANDOMTABLEN];       // 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 ] );
}
```

uses **k** lowest mantissa bits **LOW_BITS**, hash function **HASH**
RandomTab, **Indx**, **Indy** are 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 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 coord. components)
  – usually quadratic or cubic blending polynomials
  – **2D**: 4 to 16 points, **3D**: 8 to 64 points

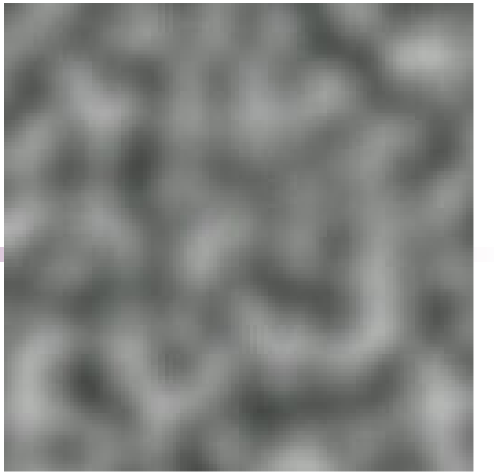
© Josef Pelikán, http://cgg.mff.cuni.cz/~pepca

Ken Perlin's noise (3D noise)

- spectrum is limited (one octave = **f ÷ 2f**)
- efficient implementation

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

Perlin's noise



❷ **grid values**:

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

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

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

Sparse convolution (Lewis)

- controlled spectral characteristics
- efficient (scalable) implementation

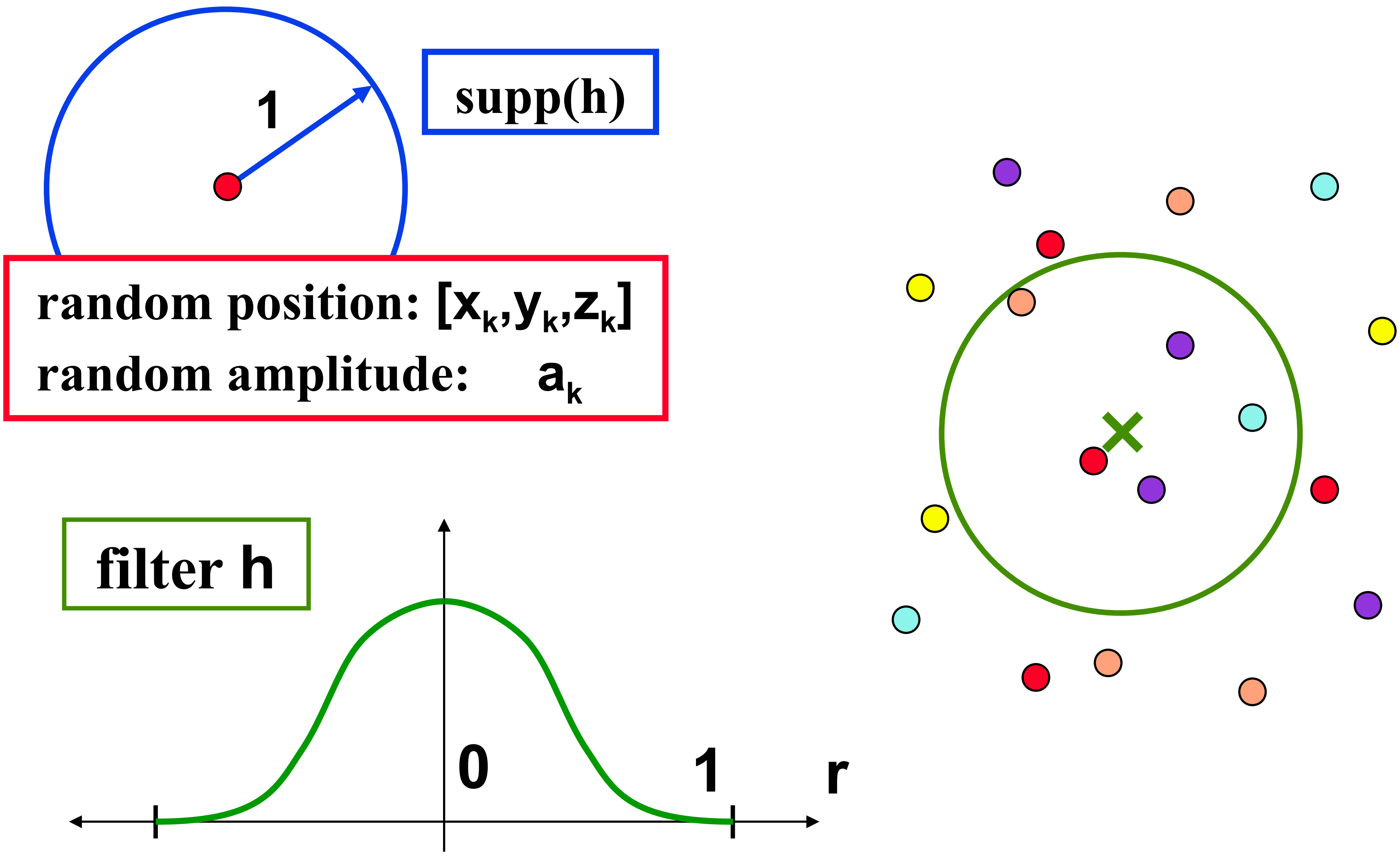
➡ **convolution** of 3D filter **h(x,y,z)** with **Poisson noise** γ:

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

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

© Josef Pelikán, http://cgg.mff.cuni.cz/~pepca

Poisson noise convolution



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 indistingui-shable from an interpolation noise
  - sparse convolution can have higher efficiency for normal quality

Efficient implementation

- space division scheme – grid with **cell size = r** (filter **supp(h)** radius – usually **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 a **hash function** can be used: `HASH(x,y,z)`

Efficient implementation

- 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(r²)** values into a table
  - no more square roots in convolution calculations

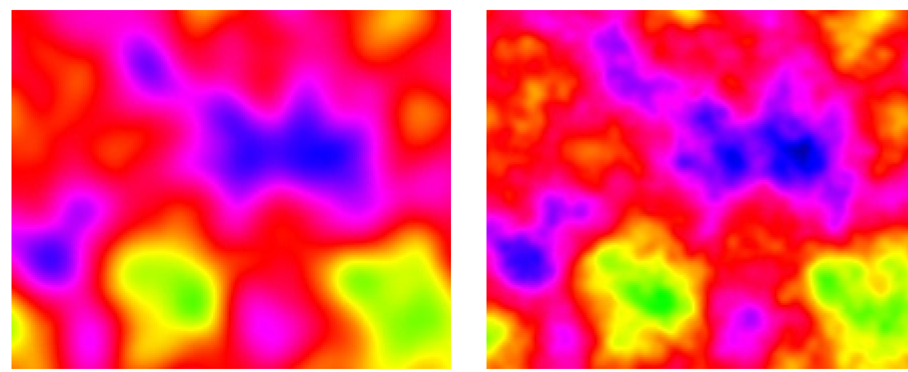
Noise function combination

- **general combination** of noise functions with frequencies $\mathbf{f_i}$, amplitudes $\mathbf{a_i}$ using drift vectors $[\mathbf{x_i}, \mathbf{y_i}, \mathbf{z_i}]$:

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

- **turbulence** simulation:

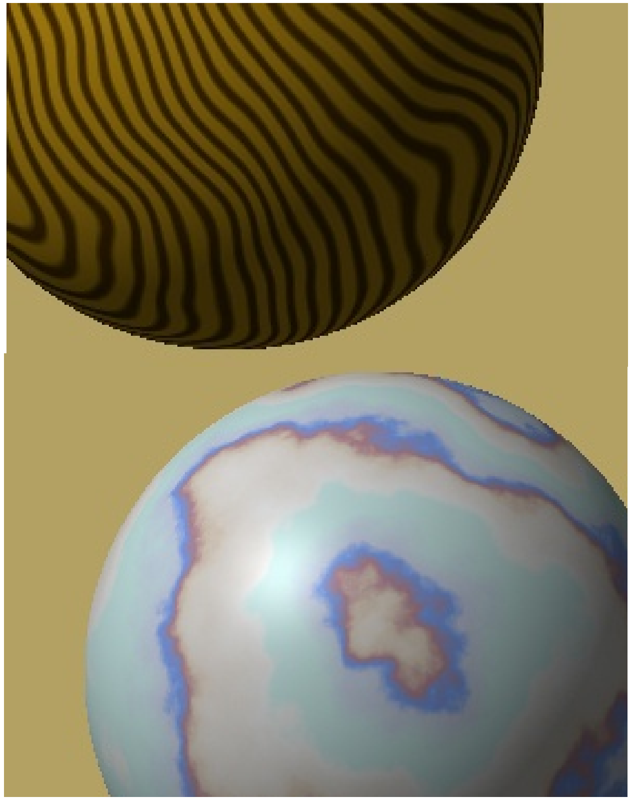
  $f_i = F^i, \quad a_i = A^{-i}$



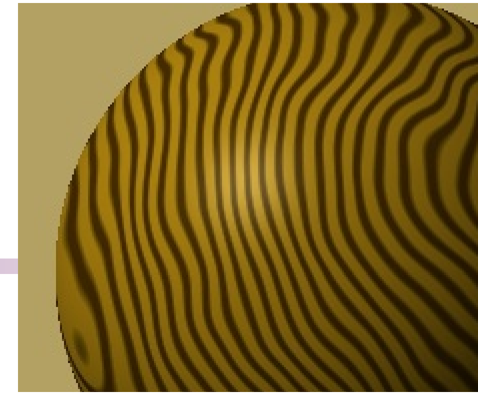
$$\sum_i \frac{1}{A^i} \cdot \mathrm{Noise}\left[F^i \cdot \left(x + x_i \right), F^i \cdot \left(y + y_i \right), F^i \cdot \left(z + z_i \right) \right]$$

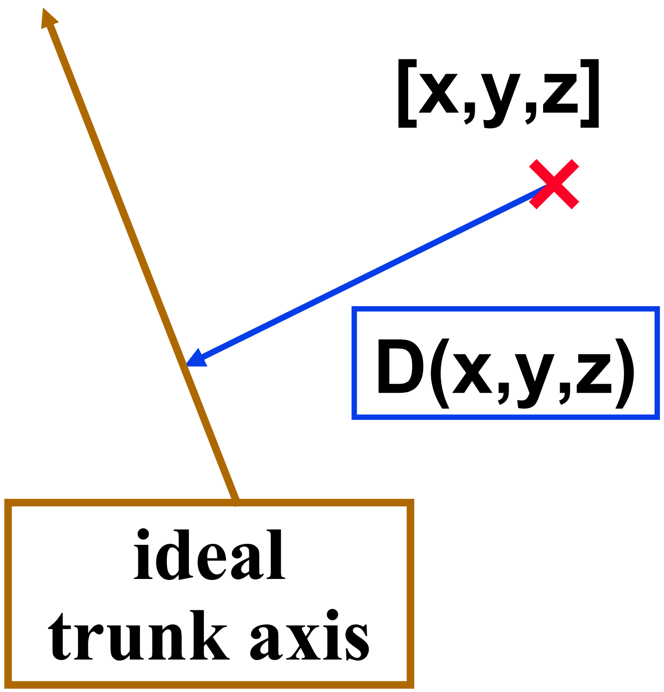
Applications

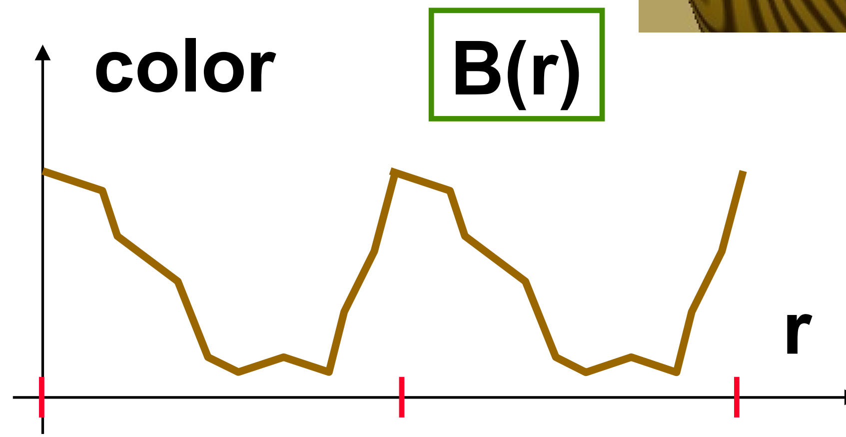
- random **normal modulations** (“bump maps”)
  - illusion of randomly wrinkled object surface
  - ”citrus peel”
- **turbulence**
  - fog, clouds, many other modeling techniques
- **3D textures**
  - inner material structure
  - wood, marble, ..



Wood imitation





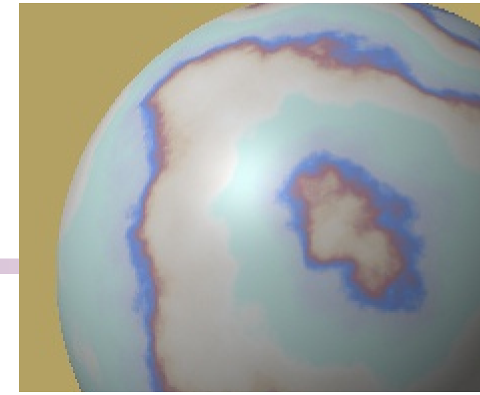


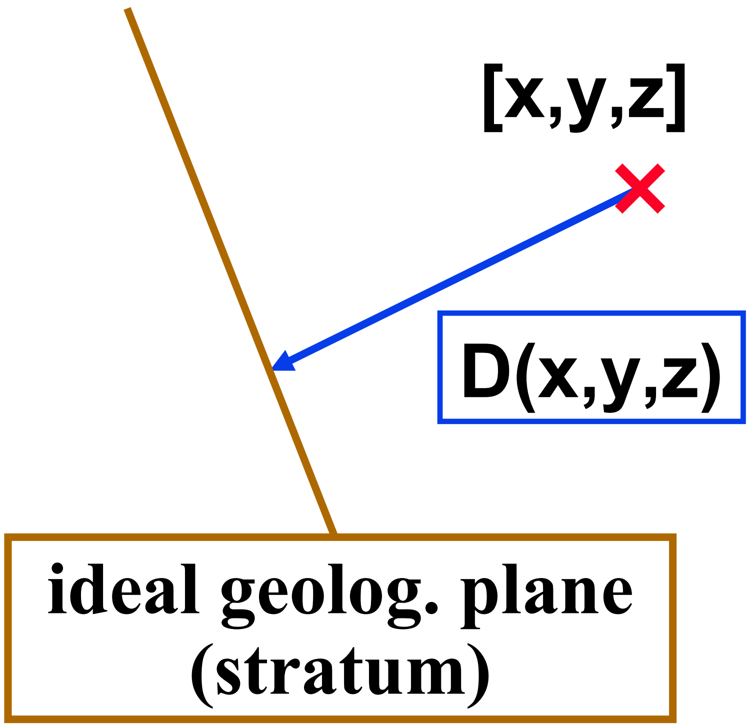
$$B\left[D(x, y, z) + Noise(x, y, z) \right]$$

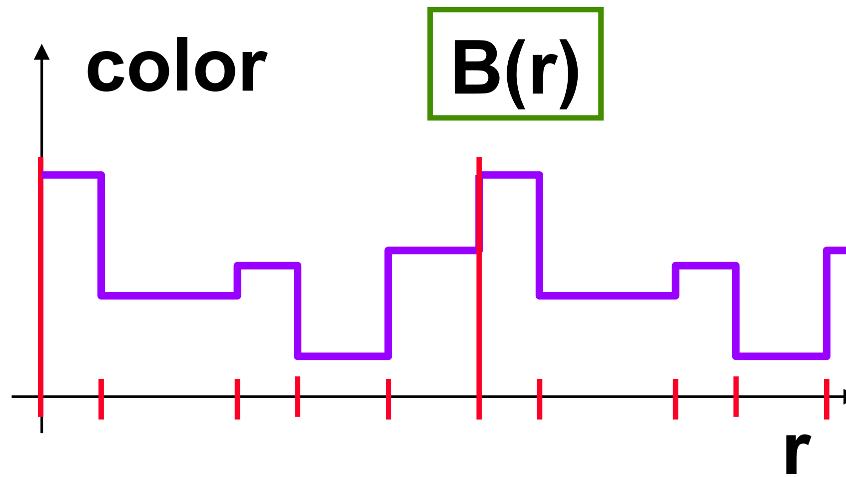
$$B\left[D(x, y, z) \cdot \left(1 + Noise_1(x, y, z) \right) + Noise_2(x, y, z) \right]$$

© Josef Pelikán, http://cgg.mff.cuni.cz/~pepca

Marble imitation







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

References

- **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

 © Josef Pelikán, http://cgg.mff.cuni.cz/~pepca