Image synthesis

Romain Vergne
A 3D world
Spatial variation

- Allow BRDF parameters to vary over space
  - Diffuse / specular colors, roughness, normal
Spatial variation

- Textures are used for representing all elements
  - Material (BRDFs, BTFs)
  - Geometry (normal / bump / displacement maps)
  - Lighting (light / environment maps)
The bad idea

- Store BRDF properties on each vertex
  - Refine / tesselate if necessary
- On a given point, interpolate properties (barycentric coords)

- But:
  - Details mean (much) more geometry/memory
  - Only works on meshes
The good idea

- Use textures

For more info on the computer artwork of Jeremy Birn see [http://www.3drender.com/jbirn/productions.html](http://www.3drender.com/jbirn/productions.html)
Textures for what?

- BRDF variations
Textures for what?

- Geometry variations

original mesh
4M triangles

simplified mesh
500 triangles

simplified mesh
and normal mapping
500 triangles
Textures for what?

- Geometry variations
Textures for what?

- Visibility (ambient occlusion)

\[ A = \frac{1}{\pi} \int V(\omega)(n \cdot \omega) d\omega \]
Textures for what?

- Lighting environments
Textures for what?

- Lighting environments
2 approaches

- From data
  - Colors, coefs, normals stored in 2D images
2 approaches

- From data
  - Colors, coefs, normals stored in 2D images

- Procedural shader
  - Little program that compute info at a given position
Procedural textures

- **pros**
  - Easy to implement (ray tracing)
  - Compact
  - Infinite resolution

- **Cons**
  - Non-intuitive
  - Difficult to match existing textures
Simple math functions

- Combination of simple functions
  - Mod
  - Clamp
  - Mix
  - Sin / cos / tan
  - Pow
  - Exp
  - Etc…
Cellular textures

- Generate a bunch of random points
- For each pixel
  - Find the nearest distance to the nearest couple points
  - Use these values to determine a color
- Voronoi-like

https://www.google.fr/search?q=cellular+texture&tbm=isch&tbo=u&source=univ&sa=X&ei=FADJVKG8McXwUpfRg6AK&ved=0CCIQsAQ&biw=1920&bih=969
Minimizing functions

- Genetic algorithms
- Reaction-diffusion
- ...

Noisy initial conditions at $t = 0$.  
State of the system at $t = 10$.  
Almost converged state at $t = 100$.  
L-systems

- **Formal grammar**

  - Alphabet: $V = \{A, B\}$
  - Constantes: $S = \{}$
  - Axiome de départ: $w = A$
  - Règles: $(A \rightarrow AB) \land (B \rightarrow A)$

Notation:

```
Algue
{
  Axiom A
  A=AB
  B=A
}
```
L-systems

- **Formal grammar**

  - Alphabet : $V = \{A, B\}$
  - Constantes : $S = \{}$
  - Axiome de départ : $w = A$
  - Règles : $(A \rightarrow AB) \land (B \rightarrow A)$

  Notation :

  ```
  Algue
  {
  Axiom A
  A=AB
  B=A
  }
  ```

- $n=0, A$
- $n=1, AB$
- $n=2, AB A$
- $n=3, AB A AB$
- $n=4, AB A AB AB A$
- $n=5, AB A AB AB A AB A AB A$
- $n=6, AB A AB A AB A AB A AB A AB A AB A$

[Link to L-systems page](http://www.kevs3d.co.uk/dev/lsystems/)

Perlin-like textures

- Requirements
  - Pseudo random
  - Arbitrary dimension
  - Smooth
  - Band pass (one scale)
  - Little memory usage
  - Implicit evaluation
Perlin-like textures

Value noise 1D case

- Distribute random values at particular locations (a grid)…
Perlin-like textures

Value noise 1D case

- Distribute random values at particular locations (a grid)…
- … and interpolate
Perlin-like textures

Value noise 1D case

- Evaluation algorithm for a given point $x$
Perlin-like textures

Value noise 1D case

- Evaluation algorithm for a given point \( x \)
- Get the associated 2 random values?
Perlin-like textures

Value noise 1D case

- Evaluation algorithm for a given point x
- Get the associated 2 random values?
  - Pseudo random function
  - Precomputed in an array
Perlin-like textures

Value noise 1D case

- Evaluation algorithm for a given point \( x \)
- Get the associated 2 random values?
  - Pseudo random function
  - Precomputed in an array
- Get relative position of \( x \) (between 0 and 1)
- Mix
Perlin-like textures

Value noise 1D case

- Evaluation algorithm for a given point x
- Get the associated 2 random values?
  - Pseudo random function
  - Precomputed in an array
- Get relative position of x (between 0 and 1)
- Mix

S-shaped function \( t = t^2(3 - 2t) \)
Perlin-like textures

Value noise 1D case

- Evaluation algorithm for a given point x
- Get the associated 2 random values?
  - Pseudo random function
  - Precomputed in an array
- Get relative position of x (between 0 and 1)
- Mix
Perlin-like textures

Value noise 1D case

- Evaluation algorithm for a given point $x$
- Get the associated 2 random values?
  - Pseudo random function
    - Precomputed in an array
  - Get relative position of $x$ (between 0 and 1)
- Mix

Pros/cons?
Perlin-like textures

Value noise 1D case

- Evaluation algorithm for a given point $x$
- Get the associated 2 random values?
  - Pseudo random function
  - Precomputed in an array
- Get relative position of $x$ (between 0 and 1)
- Mix

Pros/cons?
Perlin-like textures

Value noise 1D case

- Controls
  - Frequency: evalNoise( x * freq )
Perlin-like textures

Value noise 1D case

- Controls
  - Frequency: evalNoise( x * freq )
  - Amplitude: evalNoise( x ) * amplitude
Perlin-like textures

Value noise 1D case

- Controls
  - Frequency: evalNoise( x * freq )
  - Amplitude: evalNoise( x ) * amplitude
  - Offsetting: evalNoise( x + offset )
Perlin-like textures

Value noise 2D case

- Same principle, using a 2D grid
Perlin-like textures

Value noise 2D case

- Same principle, using a 2D grid
- Need 3 interpolations instead of 1

https://www.shadertoy.com/view/lsf3WH
Perlin-like textures

Value noise 3D case

- Same principle, using a 3D grid
- Need 7 interpolations
Perlin-like textures

Value noise 3D case

- Same principle, using a 3D grid
- Need 7 interpolations

https://www.shadertoy.com/view/4sfGzS
Perlin-like textures

Gradient noise (1D case)

- Instead of distributing random positions:
  - Consider positions at 0
  - Distribute random gradients
  - Interpolate
Perlin-like textures

Gradient noise (1D case)

- Instead of distributing random positions:
  - Consider positions at 0
  - Distribute random gradients
  - Interpolate

Available: G1, G2 and dx
Perlin-like textures

Gradient noise (1D case)

- Instead of distributing random positions:
  - Consider positions at 0
  - Distribute random gradients
  - Interpolate

Available: G1, G2 and dx

\[ n1 = dx \times G1 \]
\[ n2 = (dx-1) \times G2 \]
Perlin-like textures

Gradient noise (1D case)

- Instead of distributing random positions:
  - Consider positions at 0
  - Distribute random gradients
  - Interpolate

Available: G1, G2 and dx

\[ n1 = dx \cdot G1 \]
\[ n2 = (dx-1) \cdot G2 \]

\[ P = w1 \cdot G1 \cdot dx + w2 \cdot G2 \cdot (dx-1) \]
Perlin-like textures

Gradient noise (2D case)

- Same principle
  - 2D gradients on each point
Perlin-like textures

Gradient noise (2D case)

- Same principle
  - 2D gradients on each point
  - Compute positional differences
Perlin-like textures

Gradient noise (2D case)

- Same principle
  - 2D gradients on each point
  - Compute positional differences
  - Compute corner values

\[
\begin{align*}
  s &= g(x_0, y_0) \cdot ((x, y) - (x_0, y_0)) , \\
  t &= g(x_1, y_0) \cdot ((x, y) - (x_1, y_0)) , \\
  u &= g(x_0, y_1) \cdot ((x, y) - (x_0, y_1)) , \\
  v &= g(x_1, y_1) \cdot ((x, y) - (x_1, y_1)) .
\end{align*}
\]
Gradient noise (2D case)

- Same principle
  - 2D gradients on each point
  - Compute positional differences
  - Compute corner values

\[
\begin{align*}
    s &= g(x_0, y_0) \cdot ((x, y) - (x_0, y_0)), \\
    t &= g(x_1, y_0) \cdot ((x, y) - (x_1, y_0)), \\
    u &= g(x_0, y_1) \cdot ((x, y) - (x_0, y_1)), \\
    v &= g(x_1, y_1) \cdot ((x, y) - (x_1, y_1)).
\end{align*}
\]

- And interpolate, as before

https://www.shadertoy.com/view/XdXGW8
Perlin-like textures

Gradient noise (3D case)

- Same principle… again
  - 3D gradients on each point
  - Compute positional differences
  - Compute corner values
  - And interpolate, as before
Perlin-like textures

Gradient noise (3D case)

- Same principle… again
  - 3D gradients on each point
  - Compute positional differences
  - Compute corner values
  - And interpolate, as before

- Speeding it up (precompute gradients)
  - Precompute (1D) table of $n$ gradients $G[n]$
  - Precompute (1D) permutation $P[n]$
  - For 3D grid point $i, j, k$
    \[ G(i,j,k) = G[ ( i + P[ (j + P[k]) \mod n ] ) \mod n ] \]
Perlin-like textures

Gradient noise (3D case)

- Simplex noise
  - Use triangles instead of quads
  - Sum contributions instead of interpolating

http://webstaff.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf
Perlin-like textures

Gradient noise (3D case)

- Simplex noise
  - Use triangles instead of quads
  - Sum contributions instead of interpolating

http://webstaff.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf
Fractal textures

- Noise at one scale = 1 octave
Fractal textures

- Noise at one scale = 1 octave
- Multiple octave usually used
  - Frequency multiplied by 2 each time
  - Hence the name octave
  - Different amplitudes too
Fractal textures

- Noise at one scale = 1 octave
- Multiple octave usually used
  - Frequency multiplied by 2 each time
  - Hence the name octave
  - Different amplitudes too
- Sum of all noises =
  - Fractal noise

© www.scratchapixel.com
Fractal textures

- Computing the \( i \)th noise texture:

\[
\begin{align*}
\text{frequency} &= 2^i \\
\text{amplitude} &= \text{persistence}^i
\end{align*}
\]
Fractal textures

- Computing the \( i \)th noise texture:

\[
\begin{align*}
\text{frequency} & = 2^i \\
\text{amplitude} & = \text{persistence}^i
\end{align*}
\]

- Persistence controls the relation between frequency and amplitude.

<table>
<thead>
<tr>
<th>Frequency</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Persistence = 1/4</td>
<td><img src="image" alt="Graph" /></td>
<td><img src="image" alt="Graph" /></td>
<td><img src="image" alt="Graph" /></td>
<td><img src="image" alt="Graph" /></td>
<td><img src="image" alt="Graph" /></td>
<td><img src="image" alt="Graph" /></td>
</tr>
<tr>
<td>Amplitude:</td>
<td>1</td>
<td>1/4</td>
<td>1/16</td>
<td>1/64</td>
<td>1/256</td>
<td>1/1024</td>
</tr>
<tr>
<td>Persistence = 1/2</td>
<td><img src="image" alt="Graph" /></td>
<td><img src="image" alt="Graph" /></td>
<td><img src="image" alt="Graph" /></td>
<td><img src="image" alt="Graph" /></td>
<td><img src="image" alt="Graph" /></td>
<td><img src="image" alt="Graph" /></td>
</tr>
<tr>
<td>Amplitude:</td>
<td>1</td>
<td>1/2</td>
<td>1/4</td>
<td>1/8</td>
<td>1/16</td>
<td>1/32</td>
</tr>
<tr>
<td>Persistence = 1/\sqrt{2}</td>
<td><img src="image" alt="Graph" /></td>
<td><img src="image" alt="Graph" /></td>
<td><img src="image" alt="Graph" /></td>
<td><img src="image" alt="Graph" /></td>
<td><img src="image" alt="Graph" /></td>
<td><img src="image" alt="Graph" /></td>
</tr>
<tr>
<td>Amplitude:</td>
<td>1</td>
<td>1/\sqrt{2}</td>
<td>1/\sqrt{2}</td>
<td>1/\sqrt{2}</td>
<td>1/\sqrt{2}</td>
<td>1/\sqrt{2}</td>
</tr>
<tr>
<td>Persistence = 1</td>
<td><img src="image" alt="Graph" /></td>
<td><img src="image" alt="Graph" /></td>
<td><img src="image" alt="Graph" /></td>
<td><img src="image" alt="Graph" /></td>
<td><img src="image" alt="Graph" /></td>
<td><img src="image" alt="Graph" /></td>
</tr>
<tr>
<td>Amplitude:</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Fractal textures

- 2D example

Freq=16
Scale=0.0625
Freq=8
Scale=0.125
Freq=4
Scale=0.25
Freq=2
Scale=0.5
Freq=1
Scale=1

© www.scratchapixel.com
Fractal textures

- 2D example

Each octave \( f \) has weight \( 1/f \)
Fractal textures

- 3D example  Sum 1/f(noise)
Fractal textures

- Taking the absolute value of the noise $\text{Sum } 1/f(|\text{noise}|)$

Creates $C_0$ discontinuities
Fractal textures

- Taking the absolute value of the noise $\text{Sum } 1/f(|\text{noise}|)$

Create $C^0$ discontinuities

1D $|\text{noise}|$

2D $|\text{noise}|$
Fractal textures

- Taking the absolute value of the noise \( \sum 1/f(|\text{noise}|) \)

 Creates C0 discontinuities

© www.scratchapixel.com

1D |noise|

2D |noise|

3D |noise|
Fractal textures

- Even better: $\sin (x + \text{Sum } 1/f(\text{noise}) )$
Fractal textures

- $\text{noise}$

$$\sin(x + \sum \frac{1}{f(|\text{noise}|)})$$

$\sum \frac{1}{f(\text{noise})}$  $\sum \frac{1}{f(|\text{noise}|)}$
Fractal textures

- Marble

\[
\sin (x + \text{Sum } 1/f(\text{noise}) ) = \text{Colormap}(\sin (x + \text{turbulence}))
\]

- Wood

\[
\text{Colormap}(\sin (\text{radius } + \text{turbulence}))
\]
Fractal textures

- Texture
- Terrain
- Clouds
- Fire

- But also,
- Density for distributing elements
  - Forest
  - Plants
  - Etc
Render to texture

- Allows to render in a texture instead of the output framebuffer
- Usefull to
  - Create textures using the GPU…
    https://www.shadertoy.com/view/4sfGzS
  - Manipulate images/renderings (post-effects)
    http://evanw.github.io/webgl-filter/
  - Deferred shading

![Deferred shading examples](image_url)
References

- MIT:

- Standford:
  - http://candela.stanford.edu/cs348b-14/doku.php

- Siggraph:

- Image synthesis & OpenGL:
  - http://romain.vergne.free.fr/blog/?page_id=97

- Path tracing and global illum:

- GLSL / Shadertoy:
  - https://www.opengl.org/documentation/glsl/
  - https://www.shadertoy.com/
  - http://www.iquilezles.org/