Rendering Competition Theme

- Four rendering competition themes

- Vote for your favorite before **Sunday 23:59 December 3rd**

- Think about the scene you want to render and how it could fit together with the theme you vote for.
Vote for your favorite!

Journey to the Unknown

Whispers of Tomorrow

Chaos in Harmony

Lost in Translation
Computer Graphics

- Texturing -

Philippe Weier
Alexander Rath
Philipp Slusallek
Overview

• Texture
• Image Textures
• Procedural Textures
• Texture Mapping
TEXTURE
Texture

- Textures modify the input for shading computations
  - Either via (painted) images textures or procedural functions
- Example texture maps for
  - Reflectance, normals, shadows, reflections, essentially anything, …

Polyhaven: https://polyhaven.com/
Texture

- Two BSDFs only, all the variations come from textures
Definition: Textures

- **Textures map texture coordinates to shading values**
  - Input: 1D/2D/3D/4D texture coordinates
    - Explicitly given or derived via other data (e.g., position, direction, …)
  - Output: Scalar or vector value

- **Modified values in shading computations**
  - Reflectance
    - Changes the diffuse or specular reflection coefficient \( (k_d, k_s) \)
  - Geometry and Normal (important for lighting)
    - Displacement mapping \( P' = P + \Delta P \)
    - Normal mapping \( N' = N + \Delta N \)
    - Bump mapping \( N' = N(P + tN) \)
  - Opacity
    - Modulating transparency (e.g., for fences in games)
  - Illumination
    - Light maps, environment mapping, reflection mapping
  - Anything else …
IMAGE TEXTURES
Image Textures

• **Image textures**
  – Return the color of the image at a given point
  – Point defined by mapping the texture coordinates $[0,1]^n$ to the entire image
  – Images may be 1D (line of pixels), 2D, and 3D (stacks of images)
  – Coordinates outside of $[0,1]^2$ can be mapped in different modes
Wrap Mode

- **Texture Coordinates**
  - $(u, v)$ in $[0, 1] \times [0, 1]$

- **What if?**
  - $(u, v)$ not in unit square?
Wrap Mode

- **Repeat**

- **Fractional Coordinates**
  - $tu = u - \lfloor u \rfloor$
  - $tv = v - \lfloor v \rfloor$
Wrap Mode

- **Mirror**

- **Fractional Coordinates**
  - $tu = u - \lfloor u \rfloor$;
  - $tv = v - \lfloor v \rfloor$;

- **Lattice Coordinates**
  - $lu = \lfloor u \rfloor$;
  - $lv = \lfloor v \rfloor$;

- **Mirror if Odd**
  - if ($lu \% 2 == 1$) $tu = 1 - tu$;
  - if ($lv \% 2 == 1$) $tv = 1 - tv$;
Wrap Mode

- **Clamp**

- **Clamp u to [0, 1]**
  - if \((u < 0)\) \(tu = 0\);
  - else if \((u > 1)\) \(tu = 1\);
  - else \(tu = u\);

- **Clamp v to [0, 1]**
  - if \((v < 0)\) \(tv = 0\);
  - else if \((v > 1)\) \(tv = 1\);
  - else \(tv = v\);
Wrap Mode

• **Border**

• **Check Bounds**
  – if \((u < 0 \lor u > 1)\)
    • \((v < 0 \lor v > 1)\)
    • return backgroundColor;
  – else
    • \(tu = u;\)
    • \(tv = v;\)
Wrap Mode

• Comparison

GL_REPEAT

GL_MIRRORED_REPEAT

GL_CLAMP_TO_EDGE

GL_CLAMP_TO_BORDER
Reconstruction Filter

• **Image Texture**
  - Discrete set of sample values

• **In Practice**
  - Hit point generally does not exactly hit a texture sample

• **Reconstruct Continuous Function**
  - Use reconstruction filter to find color for hit point
Nearest Neighbor

- **Local Coordinates**
  - Assuming cell-centered samples
  - $u = tu \times \text{resU}$;
  - $v = tv \times \text{resV}$;

- **Lattice Coordinates**
  - $lu = \min(\lceil u \rceil, \text{resU} - 1)$;
  - $lv = \min(\lceil v \rceil, \text{resV} - 1)$;

- **Texture Value**
  - return $\text{image}[lu, lv]$;
Nearest Neighbor

- In Lightwave with Repeat Wrap Mode
Bilinear Interpolation

• **Local Coordinates**
  - Assuming node-centered samples
  - \( u = tu \times (\text{resU} - 1); \)
  - \( v = tv \times (\text{resV} - 1); \)

• **Fractional Coordinates**
  - \( fu = u - \lfloor u \rfloor; \)
  - \( fv = v - \lfloor v \rfloor; \)

• **Texture Value**
  - \( \text{return} \ (1-fu) \ (1-fv) \ \text{image}[\lfloor u \rfloor, \lfloor v \rfloor] \)
    \[+ \ (1-fu) \ (fv) \ \text{image}[\lfloor u \rfloor, \lfloor v \rfloor + 1] \]
    \[+ \ (fu) \ (1-fv) \ \text{image}[\lfloor u \rfloor + 1, \lfloor v \rfloor] \]
    \[+ \ (fu) \ (fv) \ \text{image}[\lfloor u \rfloor + 1, \lfloor v \rfloor + 1] \]
Bilinear Interpolation

- **Successive Linear Interpolations**
  - \( u_0 = (1-fv) \cdot \text{image}[u, v] \) 
    \[ + (fv) \cdot \text{image}[u+1, v]\]
  
  - \( u_1 = (1-fv) \cdot \text{image}[u+1, v] \) 
    \[ + (fv) \cdot \text{image}[u+1, v+1]\]

  - return \((1-fu) \cdot u_0\) 
    \[ + (fu) \cdot u_1;\]
Bilinear Interpolation

- In Lightwave with Repeat Wrap Mode
Nearest vs. Bilinear Interpolation
Bicubic Interpolation

- **Properties**
  - Assuming node-centered samples
  - Essentially based on cubic splines (see later)

- **Pros**
  - Even smoother

- **Cons**
  - More complex & expensive (4x4 kernel)
  - Overshoot
Discussion: Image Textures

• Pros
  – Simple generation
    • Painted, simulation, ...
  – Simple acquisition
    • Photos, videos

• Cons
  – Illumination “frozen” during acquisition (e.g. photo)
  – Limited resolution
  – Susceptible to aliasing
  – High memory requirements (often HUGE for films, 100s of GB)
  – Issues when mapping 2D image onto 3D object
PROCEDURAL TEXTURES
Discussion

- **Cons**
  - Possibly non-trivial programming

- **Pros**
  - Flexibility & parametric control
  - Unlimited resolution
  - Anti-aliasing possible
  - Low memory requirements
  - May be directly defined as 3D “image” mapped to 3D geometry
  - Low-cost visual complexity
2D Checkerboard

- **Lattice Coordinates**
  - \( lu = \lfloor u \rfloor \)
  - \( lv = \lfloor v \rfloor \)

- **Compute Parity**
  - \( \text{parity} = (lu + lv) \mod 2; \)

- **Return Color**
  - \( \text{if} \ (\text{parity} == 1) \)
    - return \( \text{color1} \);
  - \( \text{else} \)
    - return \( \text{color0} \);
3D Checkerboard - Solid Texture

- **Lattice Coordinates**
  - $lu = \lceil u \rceil$
  - $lv = \lceil v \rceil$
  - $lw = \lceil w \rceil$

- **Compute Parity**
  - $parity = (lu + lv + lw) \mod 2$;

- **Return Color**
  - if ($parity == 1$)
    - return color1;
  - else
    - return color0;
2D Checkerboard – Nearest-Node

- **Fractional Coordinates**
  - \( fu = u - \lfloor u \rfloor \)
  - \( fv = v - \lfloor v \rfloor \)

- **Compute Booleans**
  - \( bu = fu < 1/2; \)
  - \( bv = fv < 1/2; \)

- **Return Color**
  - if \( (bu \land bv) \)
    - return color1;
  - else
    - return color0;
Tile

- **Fractional Coordinates**
  - $fu = u - \lfloor u \rfloor$
  - $fv = v - \lfloor v \rfloor$

- **Compute Booleans**
  - $bu = fu < \text{groutWidth}$;
  - $bv = fv < \text{groutWidth}$;

- **Return Color**
  - if ($bu || bv$)
    - return groutColor;
  - else
    - return tileColor;
Brick

• **Shift Column for Odd Rows**
  - parity = ⌊ \( v \) ⌋ % 2;
  - \( u = \text{parity} \times 0.5; \)

• **Fractional Coordinates**
  - \( fu = u - \lfloor u \rfloor \)
  - \( fv = v - \lfloor v \rfloor \)

• **Compute Booleans**
  - \( bu = fu < \text{mortarWidth}; \)
  - \( bv = fv < \text{mortarWidth}; \)

• **Return Color**
  - if (\( bu \) || \( bv \))
    - return mortarColor;
  - else
    - return brickColor;
More Variation

(a) Simple bond
(b) Scottish bond
(c) Flemish bond
(d) Sussex bond
(e) Monk bond
Other Patterns

- Circular Tiles
- Octagonal Tiles
- Use your imagination!
**Perlin Noise**

- **Natural Patterns**
  - Similarity between patches at different locations
    - Repetitiveness, coherence (e.g. skin of a tiger or zebra)
  - Similarity on different resolution scales
    - Self-similarity
  - But never completely identical
    - Additional disturbances, turbulence, noise

- **Mimic Statistical Properties**
  - Purely empirical approach
  - Looks convincing, but has nothing to do with material’s physics
Perlin Noise

- Natural Fractals
Noise Function

• **Noise**(x, y, z)
  – Statistical invariance under rotation
  – Statistical invariance under translation
  – Roughly fixed frequency of ~1 Hz

• **Integer Lattice** (i, j, k)
  – **Value noise**
    • Random value at lattice points
  – **Gradient noise**
    • Random gradient vector at lattice point
    • Compute lattice values as dot products
  – **Interpolation**
    • Bi-/tri-linear or cubic (Hermite spline)
  – **Hash function**
    • Randomized look up
    • Virtually infinite extent
Noise vs. Noise

- **Value Noise vs. Gradient Noise**
  - Gradient noise has lower regularity artifacts
  - More high frequencies in noise spectrum

- **Random Values vs. Perlin Noise**
  - Stochastic vs. deterministic
Turbulence Function

- **Noise Function**
  - Single spike in frequency spectrum

- **Natural Textures**
  - Decreasing amplitude for high frequencies

- **Turbulence from Noise**
  - \(\text{Turbulence}(x) = \sum_{i=0}^{k} |a_i \cdot \text{noise}(f_i x)|\)
    - Frequency: \(f_i = 2^i\)
    - Amplitude: \(a_i = 1 / p^i\)
    - Persistence: \(p\) typically \(p=2\)
    - Power spectrum: \(a_i = 1 / f_i\)
    - Brownian motion: \(a_i = 1 / f_i^2\)
  - Summation truncation
    - 1st term: \(\text{noise}(x)\)
    - 2nd term: \(\text{noise}(2x)/2\)
    - ...
    - Until period \((1/f_k) < 2\) pixel-size (band limit)
Synthesis of Turbulence (1-D)

Amplitude: 128, frequency: 4

Amplitude: 64, frequency: 8

Amplitude: 32, frequency: 16

Amplitude: 16, frequency: 32

Amplitude: 8, frequency: 64

Sum of Noise Functions = (Perlin Noise)
Synthesis of Turbulence (2-D)
Example: Marble

- **Overall Structure**
  - Alternating layers of white and colored marble
  - $f_{\text{marble}}(x,y,z) := \text{marble\_color}(\sin(x))$
  - $\text{marble\_color} : \text{transfer function (see lower left)}$

- **Realistic Appearance**
  - Simulated turbulence
  - $f_{\text{marble}}(x,y,z) := \text{marble\_color}(\sin(x + \text{turbulence}(x, y, z)))$
Solid Noise

• 3D Noise Texture
  – Wood
  – Bump map
  – Marble
Others Applications

• **Bark**
  – Turbulated saw-tooth function

• **Clouds**
  – White blobs
  – Turbulated transparency along edge

• **Animation**
  – Vary procedural texture function’s parameters over time

• **See** [Rainforest ShaderToy](#)
TEXTURE MAPPING
Textures Coordinates

• **Solid Textures**
  – 3D world/object \((x,y,z)\) coords → 3D \((u,v,w)\) texture coordinates
  – Similar to carving object out of material block

• **2D Textures**
  – 3D Cartesian \((x,y,z)\) coordinates → 2D \((u,v)\) texture coordinates?
Parametric Surfaces

• **Definition**
  - Surface defined by parametric function
    - \((x, y, z) = p(u, v)\)
  - Input
    - Parametric coordinates: \((u, v)\)
  - Output
    - Cartesian coordinates: \((x, y, z)\)

• **Texture Coordinates**
  - Directly derived from surface parameterization
  - Invert parametric function
    - From world coordinates to parametric coordinates
Parametric Surfaces

- **Triangle**
  - Use barycentric coordinates directly
  - $p(u, v) = (1 - u - v)p_0 + up_1 + vp_2$
Parametric Surfaces

- **Triangle Mesh**
  - Associate a predefined texture coordinate to each triangle vertex
    - Interpolate texture coordinates using barycentric coordinates
      - \( u = \lambda_0 p_{0u} + \lambda_1 p_{1u} + \lambda_2 p_{2u} \)
      - \( v = \lambda_0 p_{0v} + \lambda_1 p_{1v} + \lambda_2 p_{2v} \)
  - Texture mapped onto manifold
    - Single texture shared by many triangles
Parametric Surfaces

- **Polar Coordinates**
  - \((x, y, 0) = \text{Polar2Cartesian}(r, \phi)\)

- **Disc**
  - \(p(u, v) = \text{Polar2Cartesian}(R v, 2\pi u) \; // \; \text{disc radius } R\)
Parametric Surfaces

- **Cylindrical Coordinates**
  - \((x, y, z) = \text{Cylindrical2Cartesian}(r, \varphi, z)\)

- **Cylinder**
  - \(p(u, v) = \text{Cylindrical2Cartesian}(r, 2\pi u, H v)\) // cylinder height \(H\)
Parametric Surfaces

• **Spherical Coordinates**
  – $(x, y, z) = \text{Spherical2Cartesian}(r, \theta, \phi)$

• **Sphere**
  – $p(u, v) = \text{Spherical2Cartesian}(r, \pi v, 2 \pi u)$
Surface Parameterization

• **Other Surfaces**
  – No intrinsic parameterization??
Intermediate Mapping

• **Coordinate System Transform**
  – Express Cartesian coordinates into a given coordinate system

• **3D to 2D Projection**
  – Drop one coordinate
  – Compute u and v from remaining 2 coordinates
Intermediate Mapping

- **Planar Mapping**
  - Map to different Cartesian coordinate system
  - \((x', y', z') = \text{AffineTransformation}(x, y, z)\)
    - Orthogonal basis: translation + row-vector rotation matrix
    - Non-orthogonal basis: translation + inverse column-vector matrix
  - Drop \(z'\), map \(u = x'\), map \(v = y'\)
  - E.g.: Issues when surface normal orthogonal to projection axis
Intermediate Mapping

- **Cylindrical Mapping**
  - Map to cylindrical coordinates (possibly after translation/rotation)
  - \((r, \phi, z) = \text{Cartesian2Cylindrical}(x, y, z)\)
  - Drop \(r\), map \(u = \phi / 2 \pi\), map \(v = z / H\)
  - Extension: add scaling factors: \(u = \alpha \phi / 2 \pi\)
  - E.g.: Similar topology gives reasonable mapping
Intermediate Mapping

- **Spherical Mapping**
  - Map to spherical coordinates (possibly after translation/rotation)
  - \((r, \theta, \phi) = \text{Cartesian2Spherical}(x, y, z)\)
  - Drop \(r\), map \(u = \phi / 2 \pi\), map \(v = \theta / \pi\)
  - Extension: add scaling factors to both \(u\) and \(v\)
  - E.g.: Issues in concave regions
Slope-Based Mapping

• Definition
  – Depends on surface normal and predefined vector

• Example
  – $\alpha = n \cdot \omega$
  – return $\alpha$ flatColor + (1 - $\alpha$) slopeColor;
Environment Map

- **Spherical Map**
  - Photo of a reflective sphere (gazing ball)
  - Photos with a fish-eye camera
Environment Map

• **Latitude-Longitude Map**
  – Remapping 2 images of reflective sphere
  – Photo with an environment camera

• **Algorithm**
  – Hit infinitely far (parallax-free) background if no intersection found
  – Cartesian coords of ray dir. → spherical coords → uv tex coords
Environment Map

- **Cube Map**
  - Remapping 2 images of reflective sphere
  - Photos with a perspective camera

- **Algorithm**
  - Find main axis (-x, +x, -y, +y, -z, +z) of ray direction
  - Use other 2 coordinates to access corresponding face texture
    - Akin to a 90° projective light
Light Maps

- **Light maps (e.g. in Quake)**
  - Pre-calculated illumination (local irradiance)
    - Often very low resolution: smoothly varying
  - Multiplication of irradiance with base texture
    - Diffuse reflectance only
  - Provides surface radiosity
    - View-independent out-going radiance
  - Animated light maps
    - Animated shadows, moving light spots, etc…

\[
B(x) = \rho(x) E(x) = \pi L_o(x)
\]

Reflectance  Irradiance  Radiosity

Representing radiosity in a mesh or texture
Billboards / Transparency Map

- **Single textured polygons**
  - Often with opacity texture
  - Rotates, always facing viewer
  - Used for rendering distant objects
  - Best results if approximately radially or spherically symmetric

- **Multiple textured polygons**
  - Azimuthal orientation: different orientations
  - Complex distribution: trunk, branches, …

Opacity texture