More Texture Mapping
Defining a Coordinate System

- Assume that $x$, $y$, and $z$ are linear functions of $u$ and $v$ in each triangle:
  \[ x = a_0u + b_0v + c_0 \]
  \[ y = a_1u + b_1v + c_1 \]
  \[ z = a_2u + b_2v + c_2 \]
- For each vertex, plug known values of $x$, $y$, $z$, $u$, $v$ into the above equations
- 9 equations (3 sets of 3 equations)
- 3x3 system of equations for $a_0$, $b_0$, $c_0$
- 3x3 system of equations for $a_1$, $b_1$, $c_1$
- 3x3 system of equations for $a_2$, $b_2$, $c_2$
- After solving:
  - Set $\vec{T} = \left( \frac{\partial x}{\partial u}, \frac{\partial y}{\partial u}, \frac{\partial z}{\partial u} \right) = (a_0, a_1, a_2)$
  - Set $\vec{B} = \left( \frac{\partial x}{\partial v}, \frac{\partial y}{\partial v}, \frac{\partial z}{\partial v} \right) = (b_0, b_1, b_2)$
  - Set $\vec{N} = \vec{T} \times \vec{B}$
Texture Mapping

Perturbing Normals
Perturbing the Normal

- Normal is \( (0, 0, 1) \) in the triangle’s local coordinate system \((\vec{T}, \vec{B}, \vec{N})\)
  - i.e. \( 0\vec{T} + 0\vec{B} + 1\vec{N} = \vec{N} \)
- Instead, use a new normal \((n_T, n_B, n_N)\) stored in or computed from a “texture”

Aside on the light direction:
- Can be computed on the fly at each point on the triangle corresponding to a pixel
- Alternatively, computed at triangle vertices, and interpolated to the interior with (perspective correct) barycentric interpolation

- Caution: Before computing dot products between light direction and normal direction, transform light direction into the triangle’s local coordinate system \((\vec{T}, \vec{B}, \vec{N})\)
Perturbing Normals

- Fetch a new perturbed normal vector (from a texture)
- Flat triangles appear to have more geometric details
Bump Maps

- Single-channel (grey-scale) height map \( h(u, v) \)
- Points on the 3D surface are given by \((u, v, h(u, v))\)
- Tangent plane at point \((u_0, v_0, h(u_0, v_0))\) is
  \[
  \frac{\partial h(u, v)}{\partial u}
  (u - u_0) - \frac{\partial h(u, v)}{\partial v}
  (v - v_0) + (h - h(u_0, v_0)) = 0
  \]
- Partial derivatives can be computed using finite differences, e.g.,
  \[
  \frac{\partial h(u,v)}{\partial u}
  |_{(u_0,v_0)} = \frac{h(u_{right},v_{center})-h(u_{left},v_{center})}{u_{right}-u_{left}}
  \]
  \[
  \frac{\partial h(u,v)}{\partial v}
  |_{(u_0,v_0)} = \frac{h(u_{center},v_{top})-h(u_{center},v_{bottom})}{v_{top}-v_{bottom}}
  \]
- The outward (non-unit) normal to this tangent plane is
  \[
  \left(-\frac{\partial h(u,v)}{\partial u}|_{(u_0,v_0)}, -\frac{\partial h(u,v)}{\partial v}|_{(u_0,v_0)}, 1 \right)
  \]
- Normalize to get a unit normal \((n_T, n_B, n_N)\)
Bump Maps

geometry

bump mapping
Normal Maps

- Stores “color–codes” of normal vectors
- \((n_T, n_B, n_N)\) is normalized, so each value is in \([-1, 1]\)
- Convert to a range of \([0, 255]\) and store as a color:

\[
(R, G, B) = 255 \times \frac{(n_T, n_B, n_N) + (1, 1, 1)}{2}
\]

\[
(n_T, n_B, n_N) = 2 \times (R, G, B)/255 - (1, 1, 1)
\]

- Larger storage for RGB image as opposed to a single channel for a height field
- But less computation to compute the normal
Normal Maps

- normal mapping on a plane
- note the variation of specular highlights created by the variation of normals
Perturbing Geometry
Displacement Mapping

- Perturb the surface geometry
  - Make new (temporary) geometry on-the-fly at render time
- Texture stores a height map $h(u,v)$. Texel values are used to perturb vertices in the normal direction

Pros:
- self-occlusion, self-shadowing
- silhouettes and silhouette shadows appear correctly

Cons:
- expensive
- requires adaptive tessellation to ensure that the surface mesh is fine enough near perturbations
- still need to bump/normal map for sub-triangle lighting variations
Displacement Mapping

Bump mapping

Displacement mapping

*note the difference in the silhouette
Displacement Mapping

- Compute normals on the vertices of the original mesh
- Move each vertex in the original normal direction a distance $h(u, v)$
- Compute new normals for the new mesh

$$\text{original geometry} \quad h(u, v) \quad \text{perturbed geometry}$$
Displacement Mapping

- Note self-occlusion, self-shadows, and silhouette shadows
Environment Mapping
Environment Mapping

- Objects reflect light transmitted to them from other objects
  - e.g. the reflection of a tree on a car
- An environment texture map stores the reflected image of the environment as a texture image

![Car with a reflection of a tree](image1.png)

![Color bleeding effect](image2.png)
Environment Mapping

- Photograph of a chrome sphere (light probe) contains the intensities of the environmental light shone onto the sphere from almost all directions.

Miller and Hoffman, 1984, “Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments”
Environment Mapping

- Assume objects in the environment are infinitely far away, so that the intensity of environmental light depends only on the direction $I$ (not on position).
- $R$ is the direction from the light probe to the camera.
- $I$ and $R$ are equal-angle from $N$ according to mirrored reflection.
  - Thus, $N$ has a one-to-one correspondence with $I$.
- In the texture (inside the red square below), each texel stores light from one direction $I$ corresponding to one surface normal $N$.

\[ (u,v) = (0,0) \quad (1,0) \quad (0,1) \quad (1,1) \]
Environment Mapping

- Coordinate system at the sphere center gives surface normal \( N = (n_x, n_y, n_z) = (x, y, z) / \sqrt{x^2 + y^2 + z^2} \)

- \( n_x \) and \( n_y \) in the range \([-1, 1]\) are transformed into the range \([0, 1]\) to get corresponding texture coordinates \((u, v) = (n_x + 1, n_y + 1)/2\)

- when rendering a CG object, use the local surface normal to compute texture coordinates in order to fetch the color and direction of the incoming light

Side view of a light probe
Environment Mapping
Texture Tiling
Texture Tiling

- Create a large image from small texture samples by repeating the small samples side by side
Texture Tiling

- Match across boundaries can be impossible if the texture doesn’t have natural periodic properties
- May look artificial because of the repetitive patterns

(From: http://procworld.blogspot.com/2013/01/introduction-to-wang-tiles.html)
Texture Synthesis
Texture Synthesis

- Create a large non-repetitive texture from a small sample by using its structural content

- Pixel-based algorithms – generate one pixel at a time

- Patch-based algorithms – generate one block/patch at a time
Texture Synthesis: Pixel-based

To generate the texture for pixel $p$

- compare $p$’s neighboring pixels in the (red) stencil with all potential choices in the sample
- choose the one with the smallest difference to fill pixel $p$

- Generate the texture in a raster scan ordering
- When the stencil tries to look up values outside the domain, periodic boundary conditions are applied (so at the beginning, the last few rows and columns of pixels are filled up with random values)

Search stencil
Texture Synthesis: Pixel-based

- Reduces repetitive patterns compared to texture tiling
- Generated texture has similar content to the input sample
- May lose too much structural content and/or create noisy or poorly structured textures

Sample

Heeger and Bergen’s method

Efros and Leung’s method

Wei and Levoy’s method
Texture Synthesis: Patch-based

Similar to texture tiling, but ...
- only uses a subset of the original texture sample to avoid repetitive blocks
- blends the overlapped regions to remove “seams”

For each patch being considered,
- search the original sample to find candidates which best match the overlap regions on the boundary
- choose from the good candidates

Advantages
- uses some structural content directly, so noise is less problematic
Texture Synthesis: Patch-based
Procedural Textures
Procedural Textures

- Created via mathematical/computational algorithms
- Good for generating natural elements
  - e.g. wood, marble, granite, stone, etc.
- Natural look is achieved using noise or turbulence functions
- Turbulence functions are used as a numerical representation of the “randomness” found in nature
Example: Marble Texture

- Marble is metamorphosed limestone
- Typically contains a variety of material impurities that are chaotically distributed during metamorphosis
Marble Texture

- Predefine layers of different color marble
- Use a function to map \((u, v)\) locations to layers
- E.g. a sine function:

\[
\text{marble}(u, v) = \text{layer\_color}(\sin(k_u u + k_v v))
\]
Marble Texture

\[ marble(u, v) = \text{layer\_color}(\sin(k_u u + k_v v)) \]

- \( k_u, k_v \) are spatial frequencies set by the user
- \( 2\pi/\sqrt{k_u^2 + k_v^2} \) determines the spatial periodicity
  - e.g. \( \sqrt{k_u^2 + k_v^2} \) is larger on the left figure
- The vector \((k_u, k_v)\) determines the direction of the patterns
- Problem: too regular, most things in nature have a degree of randomness
- Solution: add noise...
Perlin Noise

- To add noise to the marble texture, one could call a random number generator at every point of the texture
  - But this is “white noise” and has no structure
- Want smoother and more structured noise

- Make a large grid with random numbers on each grid node
  - Interpolate the noise to the points inside the lattice cells
  - This gives spatial coherency
- Ken Perlin proposed a specific method like this
Perlin Noise

- Lay a 2D grid over the image, and assign a pseudo-random unit gradient to each grid point
- For each pixel with texture coordinates \((u, v)\), find out which grid cell it is inside
- Compute a weighted average of the gradients dot-product(ed) with the distance to that corner to get a noise value at \((u, v)\)

\[
n(u, v) = \sum_{i,j} w\left(\frac{u - u_i}{\Delta u}\right) w\left(\frac{v - v_j}{\Delta v}\right) \left( g(u_i, v_j) \cdot \left( (u, v) - (u_i, v_j) \right) \right)
\]

- Cubic weighting function: \(w(t) = 2|t|^3 - 3|t|^2 + 1 \) (for \(-1 < t < 1\))
Perlin Noise

- Many natural textures contain a variety of feature sizes in the same texture.
- Perlin Noise function recreates this by adding together noises with different frequencies and amplitudes

\[ \text{perlin}(u, v) = \sum_{k} n(\text{frequency}(k) \ast (u, v)) \ast \text{amplitude}(k) \]

- Typically, frequencies and amplitudes are chosen via

\[ \text{frequency}(k) = 2^k \]
\[ \text{amplitude}(k) = \text{persistence}^k \]

- Each successive noise function added is called an octave, because it is twice the frequency of the previous one.
- Persistence is a parameter (\(\leq 1\)) diminishing the relative amplitudes of higher frequencies.
Perlin Noise

- 1D example
- Smaller persistence = smaller higher frequency noise = smoother result

<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 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 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 / root2 Amplitude:</td>
<td>1</td>
<td>1/1.414</td>
<td>1/2</td>
<td>1/2.828</td>
<td>1/4</td>
<td>1/5.656</td>
</tr>
<tr>
<td>Persistence = 1 Amplitude:</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(from: http://freespace.virgin.net/hugo.elias/models/m_perlin.htm)
Perlin Noise

- Used to generate height-fields, marble textures, etc.
- Can rescale it and add to itself to get a variety of natural looking maps
Perlin Noise

- Can be scaled and added to the marble texture
  \[
  \text{marble}(u, v) = \text{layer}\_\text{color}(\sin(k_u u + k_v v + A \times \text{perlin}(u, v)))
  \]
- Set the value of the parameter \( A \) to scale the noise
3D Textures
3D Textures

- Typically generated procedurally, 3D images are rare
- Although, one could slice up a 3D object and take a bunch of 2D pictures to make a 3D texture
- Or use some sort of 3D imaging technology

- Human Slices:

  Vertical  Horizontal  Brain
3D Textures

- Generate a 3D texture representing the material
- “Carve” the object out of this 3D texture
- Eliminate the difficulty of wrapping a 2D texture over a complex 3D object
- No need to worry about matching up the texture at the seams
3D Textures

- Marble texture function with Perlin noise for a 3D texture

\[
\text{marble}(u, v, w) = \text{layer\_color}(\sin(k_u u + k_v v + k_w w + A \times \text{perlin}(u, v, w)))
\]
3D Textures

- Wood texture is caused by tree rings

Compute cylindrical coordinates for \((x, y, z)\) object points

\[
R = \sqrt{x^2 + z^2} \\
\theta = \tan^{-1}\left(\frac{z}{x}\right) \\
H = y
\]
3D Textures

- Results

![Diagram of 3D textures showing different effects: rings, added eccentricity, added twist, added tilt.](image)
Machine Learning
Machine Learning Approaches…

Neural Texture Synthesis: Gram Matrix

Each layer of CNN gives $C \times H \times W$ tensor of features; $H \times W$ grid of $C$-dimensional vectors.

Outer product of two $C$-dimensional vectors gives $C \times C$ matrix measuring co-occurrence.

Average over all $H \times W$ pairs of vectors, giving Gram matrix of shape $C \times C$.

Efficient to compute; reshape features from $C \times H \times W$ to $=C \times HW$.

Then compute $G = FF^T$.
Machine Learning Approaches...

Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape \( C_i \times H_i \times W_i \)
3. At each layer compute the Gram matrix giving outer product of features:
   \[ G^l_{ij} = \sum_k F^l_{ik} F^l_{jk} \] (shape \( C_i \times C_i \))
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5

\[ E_l = \frac{1}{4N_i^2 M_i^2} \sum_{i,j} (G^l_{ij} - \hat{G}^l_{ij})^2 \]
\[ \mathcal{L}(x, \hat{x}) = \sum_{l=0}^L w_l E_l \]

Figure copyright Leon Gatys, Alexander S. Ecker, and Matthias Bethge, 2015. Reproduced with permission.
Machine Learning Approaches...

Neural Texture Synthesis

Reconstructing texture from higher layers recovers larger features from the input texture

Figure copyright Leon Gatys, Alexander S. Ecker, and Matthias Bethge, 2015. Reproduced with permission.

Fei-Fei Li & Justin Johnson & Serena Yeung  Lecture 11 - 62  May 10, 2017
Homework Tip
DEBUG with checkerboard textures
Question 1 (short/long)

- Email TAs an image of a texture sample of Perlin noise (or the texture applied to an object)
- Exceptionally nice images receive an extra credit point