Texture Mapping

Introduction to Computer Graphics
Torsten Möller
Reading

• Angel - Chapter 7.4-7.10, 9.9 and 11.8
• Hughes, van Dam, et al: Chapter 20 (+17)
• Shirley+Marshner: Chapter 11
Our aim today

• What is texture mapping / what types of mappings do we have?

• Two-pass mapping

• Texture transformations

• Billboards, bump, environment, chrome, light, and normal maps

• 3D texture mapping / volume rendering

• Hypertextures
Texture Mapping

- (sophisticated) illumination models
  - gave us “realistic” (physics-based) looking surfaces
  - not easy to model
  - mathematically and computationally challenging

- Phong illumination/shading
  - easy to model
  - relatively quick to compute
  - only gives us dull surfaces
Texture Mapping (2)

- surfaces “in the wild” are very complex
- cannot model all the fine variations
- we need to find ways to add surface detail
- How?
Texture Mapping (3)

• solution - (it’s really a cheat!!)

Map surface detail from a predefined (easy to model) table (“texture”) to a simple polygon

• How?
Texture Mapping (4)

- Problem #1
  - Fitting a square peg in a round hole
  - we deal with non-linear transformations
  - which parts map where?
Texture Mapping (5)

• Problem #2
  – Mapping from a pixel to a “texel”
  – aliasing is a huge problem!

© Machiraju/Zhang/Möller/Fuhrmann
What is an image?

• How can I find an appropriate value for an arbitrary (not necessarily integer) index?
  – How would I rotate an image 45 degrees?
  – How would I translate it 0.5 pixels?
What is a texture?

• Given the (texture/image index) \((u,v)\), want:
  
  \(- F(u,v) \implies \text{a continuous reconstruction}\)
  
  • \(= \{ R(u,v), G(u,v), B(u,v) \}\)
  • \(= \{ I(u,v) \}\)
  • \(= \{ \text{index}(u,v) \}\)
  • \(= \{ \text{alpha}(u,v) \}\)
  • \(= \{ \text{normals}(u,v) \}\)
  • \(= \{ \text{surface\_height}(u,v) \}\)
  • \(= \ldots\)
What is a texture? (2)

- Color
- specular ‘color’ (environment map)
- normal vector perturbation (bump map)
- displacement mapping
- transparency
- ...

© Machiraju/Zhang/Möller/Fuhrmann
RGB Textures

- Places an image on the object
- “typical” texture mapping
Dependend Textures

• Perform table look-ups after the texture samples have been computed.

http://geomorph.sourceforge.net/
Intensity Modulation Textures

• Multiply the objects color by that of the texture.
Opacity Textures

- A binary mask, redefines the geometry by setting parts of it transparent.
Bump Mapping

• This modifies the surface normals.

http://www.siggraph.org/education/materials/HyperGraph/hypergraph.htm
© Machiraju, Zhang, Möller, Pinkall
Displacement Mapping

• Modifies the surface position in the direction of the surface normal.

Reflection Properties

• $K_d$, $K_s$
• BRDF’s
  – Brushed Aluminum
  – Tweed
  – Non-isotropic or anisotropic surface micro facets.
Two-pass Mapping

- Idea by Bier and Sloan
- S: map from texture space to intermediate space
- O: map from intermediate space to object space
Two-pass Mapping (2)

- Map texture to intermediate (S map):
  - Plane
  - Cylinder
  - Sphere
  - Box

- Map object to same (O map).

© Machiraju/Zhang/Möller/Fuhrmann
S map

- Mapping to a 3D Plane
  - Simple Affine transformation
    - rotate
    - scale
    - translate
S map (2)

- Mapping to a Cylinder
  - Rotate, translate and scale in the uv-plane
  - $u \rightarrow \theta$
  - $v \rightarrow z$
  - $x = r \cos(\theta)$, $y = r \sin(\theta)$
S map (3)

• Mapping to Sphere
  – Impossible!!!!
  – Severe distortion at the poles
  – \( u \rightarrow \theta \)
  – \( v \rightarrow \phi \)
  – \( x = r \sin(\theta) \cos(\phi) \)
  – \( y = r \sin(\theta) \sin(\phi) \)
  – \( z = r \cos(\theta) \)
S map (4)

• Mapping to a Cube

common seam
S map (5)

• Mapping to a Cube
O map

- O mapping:
  - reflected ray (environment map)
  - object normal
  - object centroid
  - intermediate surface normal (ISN)

- that makes 16 combinations

- only 5 were found useful
O map (2)

- Cylinder/ISN (shrinkwrap)
  - Works well for solids of revolution
- Plane/ISN (projector)
  - Works well for planar objects
- Box/ISN
- Sphere/Centroid
- Box/Centroid

Works well for roughly spherical shapes
O map (3)

• Plane/ISN
  – Resembles a slide projector
  – Distortions on surfaces perpendicular to the plane.
O map (4)

- Cylinder/ISN
  - Distortions on horizontal planes
O map (5)

- Sphere/ISN
  - Small distortion everywhere.
Texture Atlas

a texture atlas is a large image containing a collection, or "atlas", of sub-images, each of which is a texture map for some part of a geometric model.
Texture Atlas

© Machiraju/Zhang/Möller/Fuhrmann
Our aim today

• What is texture mapping / what types of mappings do we have?
• Two-pass mapping
• Texture transformations
  – real-time vs. quality
• Billboards, bump, environment, chrome, light, and normal maps
• 3D texture mapping / volume rendering
• Hypertextures
Texture Mapping - Practice I

• A lot of interaction is required to “get it right” - modeler needs patience or a good algorithm

• can’t really get distortion-free or size-preserving maps → paint the texture on the projected model → (almost) do not care about distortions!

• what else do we care about under the topic “practical”?
Texture Mapping - Practice II

• we need real-time performance
  – “pre”-compute map of vertices (in the modeling package; store these with the mesh data)
  – quickly find an inverse mapping between internal points of polygon and texels
  – hardware support

• we need high quality (anti-aliasing)
  – convolve with anti-aliasing filter
  – “pre”-compute blurred (scaled-down) version
  – “pre”-compute high-res (magnified) version
  – hardware support
Image space scan

• For each y
  – For each x
    • compute $u(x,y)$ and $v(x,y)$
    • copy texture$(u,v)$ to image$(x,y)$

• Samples the warped texture at the appropriate image pixels.
• inverse mapping
Image space scan (2)

- Problems:
  - Finding the inverse mapping
    - Use one of the analytical mappings
    - Bi-linear or triangle inverse mapping
  - May miss parts of the texture map

(c) Pascal Vuylsteker
Texture space scan

• For each $v$
  – For each $u$
    • compute $x(u,v)$ and $y(u,v)$
    • copy texture($u,v$) to image($x,y$)

• Places each texture sample on the mapped image pixel.

• Forward mapping
Texture space scan (2)

- Problems:
  - May not fill image
  - Forward mapping needed
Texture Mapping - Real-Time

- **Given**: map of vertices
- **Need**: a 1-to-1 correspondence between pixels of the screen projection of the polygon to the texels in the texture
- main approaches:
  - triangle mapping
  - projective transformation
Triangle Interpolation

- The equation: \( f(x,y) = Ax+By+C \) defines a linear function in 2D.
- Knowing the values of \( f() \) at three locations gives us enough information to solve for \( A, B, \) and \( C. \)
- We do this for both components \( u(x,y) \) as well as for \( v(x, y) \)
Triangle Interpolation (2)

• Ok for orthogonal projections, but not for the general case
• Need to take care of perspective case →
• We need to find two 3D functions: \( u(x,y,z) \) and \( v(x,y,z) \).
Triangle Interpolation (2)

\[ \text{Cp} = 1 \times 0.5 + 0 \times 0.5 \]

\[ \lambda_1 = 0.5 \]

\[ \text{V1} (1,1,1) \]

\[ \lambda_0 = 0 \]

\[ \lambda_2 = 0.5 (0,0,0) \]

\[ \text{Cp} = (0.5, 0.5, 0.5) \]

\[ \text{Cp} = 1 \times 0.666 + 0 \times 0.333 \]

\[ \lambda_1 = 0.666 \]

\[ \lambda_2 = 0.333 \]

© www.scratchapixel.com
Projective Transformation

- We actually map a square to a square
- we know there was a perspective distortion - let's set up a general model:
Projective Transformation (2)

\[ x = \frac{au + bv + c}{gu + hv + i} \quad y = \frac{du + ev + f}{gu + hv + i} \]

- i can be set to one
- 8 equations, 8 unknowns!!
- Another way to look at it:

\[
\begin{bmatrix}
  x' \\
  y' \\
  w
\end{bmatrix} =
\begin{bmatrix}
  a & b & c \\
  d & e & f \\
  g & h & i
\end{bmatrix}
\begin{bmatrix}
  u' \\
  v' \\
  q
\end{bmatrix}
\]
Projective Transformation (3)

Affine texture mapping directly interpolates a texture coordinate \( u_\alpha \) between two endpoints \( u_0 \) and \( u_1 \):

\[
    u_\alpha = (1 - \alpha)u_0 + \alpha u_1 \quad \text{where} \quad 0 \leq \alpha \leq 1
\]

Perspective correct mapping interpolates after dividing by depth \( z \) then uses its interpolated reciprocal to recover the correct coordinate:

\[
    u_\alpha = \frac{(1 - \alpha)\frac{u_0}{z_0} + \alpha \frac{u_1}{z_1}}{(1 - \alpha)\frac{1}{z_0} + \alpha \frac{1}{z_1}}
\]

https://en.wikipedia.org/wiki/Texture_mapping#Perspective_correctness
Projective Transformation (4)

- does NOT preserve equidistant points
- preserve lines in all orientations
- concatenation is projective again
- inverse mapping is a projective mapping too
Projective Transformation (5)
Quality Considerations

• So far we just mapped one point
• results in bad aliasing (resampling problems)
• we really need to integrate over polygon
• super-sampling is not such a good solution (slow!)
• most popular - mipmaps
Quality Considerations

ideal

no filter (point sampling)

box filter
Quality Considerations

• Pixel area maps to “weird” (warped) shape in texture space
• hence we need to:
  – approximate this area
  – convolve with a wide filter around the center of this area
Quality Considerations

© Machiraju/Zhan
Quality considerations

• the area is typically approximated by a rectangular region (found to be good enough for most applications)
• filter is typically a box/averaging filter - other possibilities
• how can we pre-compute this?
How do we get $F(u,v)$?

- We are given a discrete set of values:
  - $F[i,j]$ for $i=0,\ldots,N$, $j=0,\ldots,M$

- Nearest neighbor:
  - $F(u,v) = F[ \text{round}(N\times u), \text{round}(M\times v) ]$

- Linear Interpolation:
  - $i = \text{floor}(N\times u)$, $j = \text{floor}(M\times v)$
  - interpolate from
    $F[i,j]$, $F[i+1,j]$, $F[i,j+1]$, $F[i+1,j+1]$
How do we get $F(u,v)$? (2)

• Higher-order interpolation
  
  $F(u,v) = \sum_i \sum_j F[i,j] \ h(u,v)$

  $h(u,v)$ is called the reconstruction kernel
  
  • Gaussian
  • Sinc function
  • splines

  – Like linear interpolation, need to find neighbors.
    
    • Usually four to sixteen
Quality considerations - MipMap

- Downsample texture beforehand
- store down-sampled version as well
- find the right level of detail
- assumes symmetric square image region (power of 2)
- problem - upsample?! (high-res)
Quality considerations - MipMap

MIP maps provide more depth realism to objects, because texture maps for varying levels of depth have been prepared.
Quality considerations - MipMap
Quality considerations - SAT

• Summed area table:
  • replace texture map with a sum of all values below the texture entry
  • problem - need to store huge numbers
  • only works for rectangular aligned pixel regions

```
<table>
<thead>
<tr>
<th>Data (a_{i,j})</th>
<th>Summed-Area Table (t_{i,j})</th>
</tr>
</thead>
<tbody>
<tr>
<td>2  1  5</td>
<td>2  3  8</td>
</tr>
<tr>
<td>0  3  2</td>
<td>2  6 13</td>
</tr>
<tr>
<td>4  4  7</td>
<td>6 14 28</td>
</tr>
</tbody>
</table>
```

(c) nVIDIA
Our aim today

- What is texture mapping / what types of mappings do we have?
- Two-pass mapping
- Texture transformations
  - real-time vs. quality
- Billboards, bump, environment, chrome, light, and normal maps
- 3D texture mapping / volume rendering
- Hypertextures
Texture Mapping

• Technicalities of the actual mapping (correspond polygon points with entries in a discrete table) are kind of “solved”
• next question - what are we going to do with those values?
• Typical - just use them as our color values (RGB)
• other methods …
Billboards and Impostors

• BBs a simple plane that rotates always perpendicular to the viewpoint
• combination of color/opacity map
• Impostors contain more complex geometry, e.g. for trees
Bump Mapping

• “real” texture - Many textures are the result of small perturbations in the surface geometry

• Modeling these changes would result in an explosion in the number of geometric primitives.

• Bump mapping attempts to alter the lighting across a polygon to provide the illusion of texture.
Bump Mapping (2)

• Consider the lighting for a modeled surface.
Bump Mapping (3)

- We can model this as deviations from some base surface.
- The question is then how these deviations change the lighting.
Bump Mapping (4)

• Assumption: small deviations in the normal direction to the surface.

\[ \vec{X} = \vec{X} + B \vec{N} \]

• Where $B$ is defined as a 2D function parameterized over the surface:

$B = f(u,v)$
Bump Mapping (5)

• Step 1: Putting everything into the same coordinate frame as $B(u,v)$.
  
  – $x(u,v), y(u,v), z(u,v)$ – this is given for parametric surfaces, but easy to derive for other analytical surfaces.
  
  – Or $O(u,v) = [x(u,v), y(u,v), z(u,v)]^T$
Bump Mapping (6)

- Define the tangent plane to the surface at a point \((u,v)\) by using the two vectors \(O_u\) and \(O_v\).

- The normal is then given by:
  - \(N = O_u \times O_v\)
Bump Mapping (7)

- The new surface positions are then given by:
  - \( O'(u,v) = O(u,v) + B(u,v) \cdot N \)
  - Where, \( N = \frac{N}{|N|} \)

- Differentiating leads to:
  - \( O'_u = O_u + B_u N + B (N)_u \approx O_u + B_u N \)
  - \( O'_v = O_v + B_v N + B (N)_v \approx O_v + B_v N \)

If \( B \) is small.
Bump Mapping (8)

- This leads to a new normal:
  \[ N'(u,v) = O_u \times O_v + B_u(N \times O_v) - B_v(N \times O_u) \]
  \[ + B_u B_v(N \times N) \]
  \[ = N + B_u(N \times O_v) - B_v(N \times O_u) \]
  \[ = N + D \]
Bump Mapping (9)

• For efficiency, can store $B_u$ and $B_v$ in a 2-component texture map.

• The cross products are geometry terms only.

• $N'$ needs to be normalized after the calculation and before lighting.
  – This floating point square root and division makes it difficult to embed into hardware.
Bump Mapping (10)

- Procedurally bump mapped object
Bump Mapping (11)

• bump mapped based on a simple image cylindrical texture space used
Bump Mapping (12)

• Bump mapping is often combined with texture mapping. Here a bump map has been used to (apparently) perturb the surface and a coincident texture map to colour the 'bump objects'.

© Machiraju/Zhang/Möller/Fuhrmann
Bump Mapping (13)

• bump mapping and texture mapping on text
Normal Map

• Pre-computation of modified normal vector $\overrightarrow{N'}$
• Stored in texture (RGB)=$\langle N_x, N_y, N_z \rangle$
• Illumination computation per pixel
  – For example in fragment program
  – Per-vertex light vector (toward light source) is interpolated
Normal Map cont.
Normal Map cont.

- 60k triangles
- 3k triangles + Normals
- 3k triangles without Normals
Normal Map cont.

(animation)

60k triangles
3k triangles + Normals
Environment Mapping

• Used to show the reflected colors in shiny objects.
Environment Mapping (2)

- Create six views from the shiny object’s centroid.
- When scan-converting the object, index into the appropriate view and pixel.
- Use reflection vector to index.
- Largest component of reflection vector will determine the face.
Environment Mapping (3)

• Problems:
  – Reflection is about object’s centroid.
  – Okay for small objects and distant reflections.
Environment Mapping (4)

- Problems! – which one is ray-traced?
Environment Mapping (5)

• Problems:
Chrome Mapping

• Cheap environment mapping
• Material is very glossy, hence perfect reflections are not seen.
• Index into a pre-computed view independent texture.
• Reflection vectors are still view dependent.
Chrome Mapping (2)

• Usually, we set it to a very blurred landscape image.
  – Brown or green on the bottom
  – White and blue on the top.
  – Normals facing up have a white/blue color
  – Normals facing down on average have a brownish color.
Chrome Mapping (3)

- Also useful for things like fire.
- The major point is that it is not important what actually is shown in the reflection, only that it is view dependent.
Light Maps

• Precompute the light in the scene
• typically works only for view-independent light (diffuse light)
• combine (texture-map) these light maps onto the polygon
Light Map

• **Combination:**
  – Structural texture
  – Light texture

• **Light maps for diffuse reflection**
  – Only Luminance channel
  – Low resolution is sufficient
  – Packing in “large” 2D texture

Reflectance  Irradiance  Radiosity  Quake2 light map
Light Map *cont.*

- Combination with textured scene
Light Map cont.

- Example: moving spot light
Our aim today

• What is texture mapping / what types of mappings do we have?
• Two-pass mapping
• Texture transformations
  – real-time vs. quality
• Billboards, bump, environment, chrome, light, and normal maps
• 3D texture mapping / volume rendering
• Hypertextures
Participating Media
Participating Media
Participating Media
3D Textures

- Representation on 3D domain
- Often used for volume representation and rendering
  - Texture = uniform grid

© Machiraju/Zhang/Möller/Fuhrmann
Direct Volume Rendering

- Emission-absorption model (Ch. 9 of PBRT book)
  - No scattering
- Ray casting (ray marching without scattering)
  - Image-space approach
- Possible on modern GPUs
  - Fragment loop for traversal along ray
  - Data stored in 3D texture, accessed by 3D texture lookup
Direct Volume Rendering: Texture Slicing

- Data stored in 3D texture
- Object space approach with slices (= proxy geometry)
- Fast method that does not need programmable GPUs

Slices parallel to image plane  Textured slices  Final image

Texturing (trilinear interpolation)  Compositing (alpha blending)
Solid Textures

• Problems of 2D textures
  – Only on surface (no internal, volumetric structure)
  – Substantial texture deformation for strongly curved surfaces
  – Texture coordinates difficult for complex objects / topology

• Solid textures
  – 3D textures for surface objects
  – Cut a curved surface out of a volumetric texture block
  – Example: wood, marble

• Disadvantages
  – Large memory consumption
Procedural Textures

• 3D textures based on noise and mathematical modeling
• analogous to sculpting or carving
Solid Textures: Examples
Our aim today

- What is texture mapping / what types of mappings do we have?
- Two-pass mapping
- Texture transformations
  - real-time vs. quality
- Billboards, bump, environment, chrome, light, and normal maps
- 3D texture mapping / volume rendering
- Hypertextures
Noise and Turbulence

- How to model texture for natural phenomena
  - Physical simulation: time-consuming
  - Instead: stochastic modeling by noise
- 2D and 3D textures
- Compact description of phenomena
  - Typically only a few parameters
Noise

• Natural objects are subject to stochastic variations
  – On different length scales
  – With spatial coherence

• Goal: stochastic structures, not just white noise

• Requirements:
  – Spatial correlation (important for changing viewpoint)
  – Reproducibility
  – Controlled frequency behavior
  – Band-limited (to avoid aliasing)
  – No (visible) periodicity
  – Rotation and translation invariant
  – Restricted range of values (for mapping to colors, etc.)
Noise cont.

• Basis for stochastic modeling: noise function according to Perlin
• Starting point: white noise (pseudo random numbers)
• Two kinds of Perlin noise:
  – Lattice value noise
  – Gradient noise
Perlin Lattice Value Noise

• Random number on lattice
• In-between: interpolation (higher-order)
  – Leads to spatial correlation
  – Band-limited
• Length scale is important
• Actual implementation:
  1D lattice with random numbers and hash function
    \[
    \text{Index}(ix, iy, iz) = \text{Permut}(ix + \text{Permut}(iy + \text{Permut}(iz)))
    \]
  – Avoids repeating patterns
Perlin Lattice Value Noise cont.

- Smooth across small intervals
Perlin Lattice Value Noise \textit{cont.}

10x10 lattice

64x64 lattice
Perlin Lattice Gradient Noise

- Random *gradients* on lattice
  - Normalized to unit length
- Scalar product of gradients and distance vectors
- Interpolation of scalar products (higher-order)
- Higher frequencies than lattice value noise
Perlin Lattice Gradient Noise

cont.

10x10 lattice gradient noise

10x10 lattice noise
Perlin Lattice Gradient Noise cont.

10x10 lattice gradient noise

20x20 lattice gradient noise
Application of Perlin Noise

- \( \text{noise}(\text{frequency} \times x + \text{offset}) \)
- \( \text{noise}(2 \times x) \) generates noise with doubled spatial frequencies
- Different \( \text{offset} \) leads to different noise of same characteristics
Turbulence

• Spectral synthesis
  – Combine noise of different frequencies:
    \[ \text{turbulence}(x) = \sum_k 1/2^k |\text{noise}(2^k x)| \]
  – Discontinuities in derivatives due to absolute value
  – Alternative: \[ \sum_k 1/2^k \text{noise}(2^k x) \]
  – Adding up details proportionate to size (self similarity, fractals)
  – Frequency spectrum \( 1/f \)
  – \( 1/f \) fractal noise
Turbulence cont.

- Spectral synthesis
  - Example for combination of different frequencies
Turbulence: Examples
Noise and Turbulence

Simple noise

\[ \sum 1/f \text{ noise()} \]
Turbulence with Structures

\[ \sum \frac{1}{f} |\text{noise()}| \]

\[ \sin(x + \sum \frac{1}{f} |\text{noise()}|) \]
Natural Phenomena

• Example: fireball
• Spatial / color variations according to turbulence
• Also temporal variations
Hypertextures

• Perlin


• Idea: mix between geometry and texture
• Extension of solid textures
• Volumetric description of densities
• Types of density functions:
  – Object density function (DF)
  – Density modulation function (DMF): fuzzy boundaries

• Hypertexture = apply DMFs to a DF repeatedly
• DMF often based on noise or turbulence
• Semi-transparent volume rendering
Hypertextures: Examples
Hypertextures: Examples cont.