Texture Mapping
What adds visual realism?

- Geometry only
- Phong shading
- Phong shading + Texture maps
Textures Supply Rendering Detail

Without texture

With texture
Textures Make Graphics Pretty

Texture → detail, detail → immersion, immersion → fun

Microsoft Flight Simulator X
Texture mapping

Texture mapping allows you to take a simple polygon and give it the appearance of something much more complex.

- Due to Ed Catmull, PhD thesis, 1974
- Refined by Blinn & Newell, 1976

Texture mapping ensures that “all the right things” happen as a textured polygon is transformed and rendered.
Non-parametric texture mapping

- With “non-parametric texture mapping”:
  - Texture size and orientation are fixed
  - They are unrelated to size and orientation of polygon
  - Gives cookie-cutter effect
With “parametric texture mapping,” texture size and orientation are tied to the polygon.

**Idea:**
- Separate “texture space” and “screen space”
- Texture the polygon as before, but in texture space
- Deform (render) the textured polygon into screen space

A texture can modulate just about any parameter – diffuse color, specular color, specular exponent, …
Implementing texture mapping

- A texture lives in its own abstract image coordinates parameterized by $(u,v)$ in the range $([0..1], [0..1])$:

- It can be wrapped around many different surfaces:

- Computing $(u,v)$ texture coordinates in a ray tracer is fairly straightforward.

- Note: if the surface moves/deforms, the texture goes with it.
Mapping to texture image coords

- The texture is usually stored as an image. Thus, we need to convert from abstract texture coordinate: 
  \((u, v)\) in the range \(([0..1], [0..1])\)
  to texture image coordinates:
  \((u_{\text{tex}}, v_{\text{tex}})\) in the range \(([0.. w_{\text{tex}}], [0.. h_{\text{tex}}])\)

**Q**: What do you do when the texture sample you need lands between texture pixels?
Texture resampling

- We need to resample the texture:

- A common choice is **bilinear interpolation**:

  \[
  T(a,b) = T[i + \Delta_x, j + \Delta_y]
  \]

  \[
  = (1 - \Delta_x)(1 - \Delta_y)T[i, j] + \Delta_x(1 - \Delta_y)T[i + 1, j]
  \]

  \[
  + (1 - \Delta_x)\Delta_yT[i, j + 1] + \Delta_x\Delta_yT[i + 1, j + 1]
  \]
Texture Coordinates

- Interpolated over rasterized primitives
Source of texture coordinates?

- Assigned ad-hoc by artist
  - Tedious!
  - Has gift wrapping problem

- Computed based on XYZ position
  - Texture coordinate generation ("texgen")
  - Hard to map to "surface space"
  - Function maps \((x,y,z)\) to \((s,t,r,q)\)

- From bi-variate parameterization of geometry
  - Good when geometry is generated from patches
  - So \((u,v)\) of patch maps to \((x,y,z)\) and \((s,t)\)
Texture Arrays

- Multiple skins packed in texture array
- Motivation: binding to one multi-skin texture array avoids texture bind per object

Texture array index

Mipmap level index

0      1      2      3      4
Textured Polygonal Models

Key-frame model geometry

Decal skin

Result
Multiple Textures

* Id Software’s Quake 2 circa 1997

lightmaps only

combined scene

decal only

(modulate)
Can define material by program

- A ‘surface shader’ computes the color of each ray that hits the surface.
- Example: Renderman surface shader

```
/*
 * Checkerboard
 */
surface checker(float Kd=.5, Ka=.1) {
    float smod = mod(10*s, 1);
    float tmod = mod(10*t, 1);
    if (smod < 0.5) {
        if (tmod < 0.5) Ci=Cs; else Ci=color(0,0,0);
    } else {
        if (tmod < 0.5) Ci=color(0,0,0); else Ci=Cs;
    }
    Oi = Os;
    Ci = Oi*Ci*(
        Ka*ambient() +
        Kd*diffuse(faceforward(normalize(N),I)));
}
```
Solid textures

- Q: What kinds of artifacts might you see from using a marble veneer instead of real marble?

- One solution is to use solid textures:
  - Use model-space coordinates to index into a 3D texture
  - Like “carving” the object from the material

- One difficulty of solid texturing is coming up with the textures.
Solid textures (cont'd)

- Here's an example for a vase cut from a solid marble texture:

  ![Solid marble texture by Ken Perlin, (Foley, IV-21)]
Displacement and Bump Mapping

- Use surface offsets stored in texture
  - Perturb or displace the surface
  - Shade on the resulting surface normals

\[
P(u,v) + S(u,v) \frac{\partial P(u,v)}{\partial u} + T(u,v) \frac{\partial P(u,v)}{\partial v} = N(u,v) = S \times T
\]

- **Displacement**
  \[
P'(u,v) = P(u,v) + h(u,v)N(u,v)
\]

- **Perturbed normal**
  \[
  N'(u,v) = P'_u \times P'_v = N + h_u(T \times N) + h_v(S \times N)
  \]

From Blinn 1976
Normal Mapping

- Bump mapping via a normal map texture
  - Normal map – x,y,z components of actual normal
  - Instead of a height field 1 value per pixel
  - The normal map can be generated from the height field
  - Otherwise have to orient the normal coordinates to the surface

\[
diffuse \times decal + specular =
\]
Displacement vs. bump mapping

- Input texture

- Rendered as displacement map over a rectangular surface
Displacement vs. bump mapping (cont'd)

Original rendering

cylinder

Rendering with bump map
wrapped around a

cylinder

Bump map and rendering by Wyvern Aldinger

University of Texas at Austin   CS354 - Computer Graphics   Don Fussell
Bump mapping example

Texture #1
(diffuse color)

Texture #2
(bump map)

Rendered Image
Combining texture maps

- Using texture maps in combination gives even better effects.

Diffuse color

Environment map (not necessary in ray tracer)

Specular coefficient

Material properties (coefficients in shading equation)
Multiple Textures

Key-frame model geometry

Bump skin texture

Decal skin texture

Gloss skin texture
Multitexturing

\[(\text{Diffuse}) \times (\text{Decal}) + (\text{Specular}) \times (\text{Gloss}) = \text{Final result!}\]
Environment mapping

- In **environment mapping** (also known as **reflection mapping**), a texture is used to model an object's environment:
  - Rays are bounced off objects into environment
  - Color of the environment used to determine color of the illumination
  - Really, a simplified form of ray tracing
  - Environment mapping works well when there is just a single object – or in conjunction with ray tracing
- Under simplifying assumptions, environment mapping can be implemented in hardware.
- With a ray tracer, the concept is easily extended to handle refraction as well as reflection.
Cube Map Textures

- Instead of one 2D images
  - Six 2D images arranged like the faces of a cube
  - +X, -X, +Y, -Y, +Z, -Z

- Indexed by 3D (s,t,r) un-normalized vector
  - Instead of 2D (s,t)
  - Where on the cube images does the vector “poke through”?
  - That’s the texture result
More Cube Mapping
Omni-directional Lighting

Access texture by surface reflection vector
Dynamic Cube Map Textures

Dynamically created cube map image

Image credit: “Guts” GeForce 2 GTS demo, Thant Thessman

Rendered scene
How do we anti-alias textures?

- We could just super-sample.
- But textures (and shader programs) are a special case; we can use true area integration!

- Approximate footprint as parallelogram
- Determine this approximate footprint using discrete differences
Pre-filtered Image Versions

- Base texture image is say 256x256
- Then down-sample 128x128, 64x64, 32x32, all the way down to 1x1

**Trick:** When sampling the texture, pixel the mipmap level with the closest mapping of pixel to texel size

**Why?** Hardware wants to sample just a small (1 to 8) number of samples for every fetch—and want constant time access
Cost of filtering can be reduced

- Store a pyramid of pre-filtered images:

- During texture lookup, read from appropriate level of the pyramid.
Mipmap LOD Selection

- Tri-linear mip-mapping means compute appropriate mipmap level
- Hardware rasterizes in 2x2 pixel entities
  - Typically called quad-pixels or just *quad*
  - Finite difference with neighbors to get change in u and v with respect to window space
    - Approximation to $\partial u/\partial x$, $\partial u/\partial y$, $\partial v/\partial x$, $\partial v/\partial y$
    - Means 4 subtractions per quad (1 per pixel)
- Now compute approximation to gradient length
  - $p = \max(\sqrt{((\partial u/\partial x)^2 + (\partial u/\partial y)^2)},\sqrt{((\partial v/\partial x)^2 + (\partial v/\partial y)^2)})$

one-pixel separation
LOD Bias and Clamping

- Convert p length to power-of-two level-of-detail and apply LOD bias
  - $\lambda = \log_2(p) + \text{lodBias}$
- Now clamp $\lambda$ to valid LOD range
  - $\lambda' = \max(\min\text{LOD}, \min(\max\text{LOD}, \lambda))$
Determine Levels and Interpolant

- Determine lower and upper mipmap levels
  - \( b = \text{floor}(\lambda') \) is bottom mipmap level
  - \( t = \text{floor}(\lambda' + 1) \) is top mipmap level

- Determine filter weight between levels
  - \( w = \text{frac}(\lambda') \) is filter weight
Determine Texture Sample Point

- Get \((u, v)\) for selected top and bottom mipmap levels
  - Consider a level \(l\) which could be either level \(t\) or \(b\)
    - With \((u, v)\) locations \((u_l, v_l)\)
  - Perform \texttt{GL\_CLAMP\_TO\_EDGE} wrap modes
    - \(u_w = \max(1/2*\text{widthOfLevel}(l),\min(1-1/2*\text{widthOfLevel}(l), u))\)
    - \(v_w = \max(1/2*\text{heightOfLevel}(l),\min(1-1/2*\text{heightOfLevel}(l), v))\)
- Get integer location \((i, j)\) within each level
  - \((i, j) = (\\text{floor}(u_w * \text{widthOfLevel}(l)),\\text{floor}(v_w * ))\)
Determine Texel Locations

- Bilinear sample needs 4 texel locations
  - (i0,j0), (i0,j1), (i1,j0), (i1,j1)
- With integer texel coordinates
  - i0 = floor(i-1/2)
  - i1 = floor(i+1/2)
  - j0 = floor(j-1/2)
  - j1 = floor(j+1/2)
- Also compute fractional weights for bilinear filtering
  - a = frac(i-1/2)
  - b = frac(j-1/2)
Determine Texel Addresses

- Assuming a texture level image’s base pointer, compute a texel address of each texel to fetch
  - Assume bytesPerTexel = 4 bytes for RGBA8 texture
- Example
  - addr00 = baseOfLevel(l) + bytesPerTexel*(i0+j0*widthOfLevel(l))
  - addr01 = baseOfLevel(l) + bytesPerTexel*(i0+j1*widthOfLevel(l))
  - addr10 = baseOfLevel(l) + bytesPerTexel*(i1+j0*widthOfLevel(l))
  - addr11 = baseOfLevel(l) + bytesPerTexel*(i1+j1*widthOfLevel(l))
- More complicated address schemes are needed for good texture locality!
Mipmap Texture Filtering

- point sampling
- mipmapped point sampling
- linear filtering
- mipmapped linear filtering
Anisotropic Texture Filtering

- Standard (isotropic) mipmap LOD selection
  - Uses magnitude of texture coordinate gradient (not direction)
  - Tends to spread blurring at shallow viewing angles
- Anisotropic texture filtering considers gradients direction
  - Minimizes blurring