Texture Mapping: Concepts

- Generalized Texturing
- Image Texturing
  - Mipmapping, Ripmapping, Summed-area Tables
  - Unconstrained Anisotropic Filtering
- Texture Caching, Compression
- Multipass Texture Rendering
- Multitexturing
- Texture Animation
Texturing

- Process of gluing n-dimensional images onto geometric objects
- Why? Dramatic increase in realism and an inexpensive means to do it.
- Hardware support for 2D and 3D texturing.
Why Texturing

- Consider accurately representing the geometry of a brick wall
- Takes significant amount of resources (geometry, memory, rendering)
- Instead apply image of a brick wall to a polygon.
- Can be unconvincing, due to uniform lighting (brick/mortar)
- Can add additional textures for added realism (shininess texture, bump maps)
Generalized Texturing

- **Normal Shading**: Lighting, material properties, and viewer position, transparency, and fog effects are taken into account.

- Texturing *modulates* the values used in the lighting equation.
  - Color could be replaced by texel color
  - Specular texture modifies shininess
  - Bump map modifies the surface normal
Texture Pipeline

1. Compute object space location
2. Use Projector function to find \((u, v)\)
3. Use corresponder function(s) to find texel
4. Modify equation or fragment value
5. Apply value transform function
Texture Pipeline

Eye

(x, y, z)
object space
(−2.3, 7.1, 88.2)

(u, v)
parameter space
(0.32, 0.29)

texture image space
(81, 74)

texel
(0.9, 0.8, 0.7)
Texture Pipeline: Projector Function

Goal is to generate **texture coordinates**, as a function of object position.

- Transforms object positions, \( (x, y, z) \) into texture space parameters, \( (u, v) \), eg.
- Can use planar (orthographic), spherical, cylindric projections.
- Curved surfaces have a natural \((u,v)\) coordinate system that can be used.
- Other possibilities: view direction, surface normal or other attributes.
Texture Pipeline: Planar Projector Function

- **Left Image**: Z coordinate discarded
- **Right Image**: Y coordinate discarded
What is the orientation of the cylinder axis in the 2 lower images?
Texture Pipeline: Spherical Projector Function
Texture Pipeline: Cubical Projector Function
Projector Function: Issues

- Near edge-on orientations causes severe distortion; must decompose model into near-planar pieces.
- Can also unwrap the mesh.
- Specification: Some systems use 3 or 4 coordinate vectors, multixtectures.
- Parameter values are always interpolated across the surface
Texture Pipeline: Corresponder Functions

Functions that convert parameter-space values to texture-space locations

- Why need them? : Flexibility in applying textures
- Examples:
  - Select a subset of an existing texture
  - Use $4 \times 4$ matrix transform (scale, rotate, shear textures).
  - Determine boundary effects.
Corresponding Functions: Wrapping Modes

- **Wrap (DirectX), repeat (OpenGL), tile**: Image repeats across the surface, usually the default.
- **Mirror**: Image repeats but mirrored every other time; provides continuity along texture edges.
- **Clamp (DirectX), Clamp to Edge (OpenGL)**: Values outside of (0,1) clamped to the edge values of texture.
- **Border (DirectX), Clamp to Border (OpenGL)**: Values outside (0,1) are clamped to a border color; e.g., useful in stitching textures in terrain applications.
3D Textures

■ 2D textures form the vast majority of textures in real-time rendering applications.

■ Covering an arbitrary 3D surface with 2D textures can be challenging, due to texture stretching and compression; Example: A solid cone.

■ Can extend textures to 3D, parameterized by \((s, t, r)\); Example: Moving a polygon through a 3D medical dataset displays slices of the data.

■ 3D texture mapping – equivalent to carving a model from a 3D volume of the material, eg. wood, marble

■ Disadvantages: Significant memory, bandwidth resources.
3D Texture Generation

- Perlin Noise Functions (Example above); can be expensive
- Can generate Perlin noise on the GPU (375 passes on GeForce 2)
- Current GPUs - upto 512M texture memory.
Texture Application To Surfaces

- Normally, RGB triplets (or RGB$\alpha$) returned from texture lookup.
- Must perform perspective correction is performed on texture coordinate values, not Gouraud interpolation, to avoid distortions.
- Normally, modify the Gouraud shaded color is modified, or light’s direction (bump mapping).
Texture Combine Functions

- **Replace**: Original color replaced by texture color
- **Decal**: Texture is blended using the $\alpha$ value from the texture; original $\alpha$ value is not modified.
- **Modulate**: Modify the surface color by the texture color, resulting in a shaded, textured surface.
Summarize: Brick Wall Example

- Modeler sets \((u, v)\) texture coords for all vertices of (wall) model.
- Texture is read into the renderer, and wall polygons are sent down the pipeline.
- A white material is used to compute illumination at each vertex; color and \((u, v)\) values interpolated across the surface.
- Brick image texel retrieved at each pixel and modulates the interpolated color, boosted by a scale factor (value function).
- Lit, textured brick wall is displayed.
Image Texturing

- Typically texture sizes are powers of 2; newer accelerators relax this restriction.
- Normally pixel color should be influenced by samples outside the grid cell (higher quality)
- What if the projected texture square does not match the screen size?
- Magnification, Minification is required.
Texture Magnification

- Ideally, can use the sinc filter
- Not feasible for real-time applications
- Box filter is of poor quality
Texture Magnification

- Tent filter (linear interpolation) is more feasible
- In 2D, using \textit{bilinear interpolation}
Bilinear Interpolation

- Texture coords: \((p_u, p_v) \in [0, 1]\)
- Texture images size: \(n \times m\) texels
- Nearest neighbor would access: \([n \ast u, m \ast v]\)
- Bilinear Interpolation: Interpolate 1D (along edges) in \(y\), then along \(x\).

Given \((u', v') = (p_u - \lfloor p_u \rfloor, p_v - \lfloor p_v \rfloor)\), and the texture color \(t(x, y)\),

\[
b(p_u, p_v) = (1 - u')(1 - v')t(x_l, y_b) + u'(1 - v')t(x_r, y_b) \\
(1 - u')v't(x_l, y_t) + u'v't(x_r, y_t)
\]
Bilinear Interpolation

Figure 5.8. Texture magnification. Here, a texture of size $32 \times 64$ was applied to a rectangle, which was viewed very closely (with respect to texture size). Therefore, the texture had to be magnified. On the left, the nearest neighbor filter is used, which simply selects the nearest texel to each pixel. Bilinear interpolation is used on the rectangle on the right. Here, each pixel is computed from a bilinear interpolation of the closest four neighbor texels.
Texture Minification

- A pixel can map to **many texels (tens, hundreds!)**.
- Must integrate the effect of texels affecting each pixel.
- As before, Sinc is too expensive.
- Nearest neighbor: **terrible aliasing problems!**
- Bilinear Interpolation: slightly better (uses 4 texels)
Texture Minification

Notice the breakup of texture on the horizon.

Figure 5.11. The top image was rendered with point sampling (nearest neighbor), the center with mipmapping, and the bottom with summed area tables.
Texture Modification (Anti-aliasing) Algorithms

- Can take average of texels inside pixel, also expensive.
- Need real-time approaches (fixed amount of resources)
  - Mip-mapping
  - Rip-mapping
  - Summed Area Tables
  - Unconstrained Anisotropic Filtering
Mipmapping

- *multum-in-parvo: many things in a small place*
- Original texture augmented with a set of smaller versions (*subtextures*) of the original.
- **Crucial:** Good filtering, Gamma correction
- Best to use a Gaussian, Lanczos, Kaiser filters
- Gamma correction is needed to ensure faithfulness of original texture.
Mipmapping: Texture Access

- **Texture Access**: Compute level of detail $d$ (OpenGL calls it $\lambda$); $(u, v, d)$ is an index into the mip-map.
- Goal is to find the texture that best approximates the pixel-texel ratio of 1:1.
- Retrieve the two texture maps (two texture accesses) above and below $d$ and perform trilinear interpolation.
Mipmapping: Level of Detail ($d$)

- Use the longer edge of quadrilateral formed by the pixel’s cell.
- Largest absolute value of the 4 differentials: $\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial v}{\partial x}, \frac{\partial v}{\partial y}$
- Usually include a LOD bias (to accommodate blurry or poorly filtered images)
Mipmapping: Problems

- **Overblurring:** Can be caused by texture projections that are non-square (edge-on viewing)
- Might need minification on \( u \) and magnification on \( v \)!
To avoid overblurring, can use a ripmap (HP)

Mipmap extended to include rectangular areas as subtextures.

Texture access: \((u, v)\) and two values for rectangular map location.

\((u, v)\) extents determine the rectangular area chosen.
Summed Area Tables

- A table the same size of the texture but has more bits of precision.
- At each location, store the sum of all texels of rectangle formed with the origin.
- During texture lookup, determine texture bounding rectangle. The average texture value is given by

\[
c = \frac{s[x_{ur}, y_{ur}] - s[x_{ur}, y_{ll}] - s[x_{ll}, y_{ur}] + s[x_{ll}, y_{ll}]}{(x_{ur} - x_{ll})(y_{ur} - y_{ll})}
\]
Unconstrained Anisotropic Filtering

- Deals effectively with variable sized texture quad projections
- Use the smaller size of quad to estimate LOD, $d$.
- Use longer side to create a line of anisotropy
- For anisotropy between 1:1 and 1:2, use two samples along the line, additional samples for higher degrees of anisotropy.
Unconstrained Anisotropic Filtering Example

Figure 24. Creating a Set of Anisotropically Filtered Images

Figure 25. Geometry Orientation and Texture Aspect Ratio
Texture Caching

- Balance between speed and minimizing number of textures in memory.
- Keep smaller textures, group polygons by their use of texture.
- Tiling or Mosaicing: Combine smaller textures to avoid excessive switching - must ensure no bleeding occurs (using mipmapping
- **Caching Strategies:** LRU used to make way for new textures - must ensure and prevent *thrashing* - switch temporarily to MRU.
- Minimize texture loading time, by *prefetching* over a few frames.
Texture Compression

- Textures in flight simulators, GIS can be huge.
- Fixed-rate texture compression is preferable for predictable frame-rate.
- S3 Texture Compression (S3TC) : standard for Direct X (called DXTC)
- Compression rates are usually from 4:1 to 6:1
- **DXTC Compression**: Image broken into $4 \times 4$ tiles.
- Two 16 bit (5:6:5) colors and 16 two bit values (two derived colors)
- **Lossy** scheme, not good for normal maps.
Multipass Texture Rendering

- Normally, illumination is calculated in one pass.
- **Multipass Rendering** can be used to effect motion blur, anti-aliasing, soft shadows, etc.
- **Why?** Hardware limitations on number of textures that can be applied simultaneously, selective application of texture to parts of lighting equation.
- Example: Texture to modify only diffuse component; can be done in 2 passes (Plate V, page 274)
- Multi-pass Rendering Operations: *add, blend*
- Example: Quake III engine uses 10 passes.
Current graphics h/w permits 2 or more textures to be applied in a single pass.

Texture blending cascade is a series of texture stages (texture units) in the form of a pipeline.

Can compute expressions of the form $AB + CD$, impossible with multipass rendering.