Texture Mapping

Why use texture mapping to increase detail?

- Expensive solution: add more detail to model
  - extra Incorporated as a part of object
  - modeling tools aren’t very good for adding detail
  - model takes up more space in memory
  - complex detail cannot be reused

- Efficient solution: map a texture onto model
  - texture maps can be reused
  - texture maps take up space in memory, but can be shared, and compression and caching
  - texture mapping can be done quickly (we’ll see later)
  - placement and creation of texture maps can be made intuitive (e.g., tools for adjusting mapping, painting directly onto object)
  - texture maps do not affect the geometry of the object

Texture Mapping

What kind of detail can go into these maps?

- diffuse, ambient and specular colors
- specular exponents
- transparency, reflectivity
- fine detail surface normals (bumps)
- data to visualize
- projected lighting and shadows

Games use “billboards” for distant detail.

Texture Mapping

What are Mappings?

- A function is a mapping
  - function maps values in its subset of a domain into its subset of a co-domain
  - each value in the domain will be mapped to one value in the co-domain
  - can transform one space into another with a function
  - viewing pipeline takes the WCS to VCS to CCS to NDCS – these are all mappings.

- What is Texture Mapping?
  - we have points on a surface in object space – the domain
  - we want to get values from a texture map – the co-domain
  - what function(s) should we use?

Basic Idea

Texture mapping is the process of mapping a geometric point to a color in a texture map

- We want to map arbitrary geometry to a pixmap of arbitrary dimension
- We do this in two steps
  - map a point on the arbitrary geometry to a point on an abstract unit square (we call this the UV map)
  - map a point on an abstract unit square to a point on the actual pixmap of arbitrary dimension

Second step is easier, so we present it first
Texture Mapping

- From a unit square to a pixmap
  - A 2D example: mapping from unit square to texture map (arbitrary pixmap, possibly with alpha values)
  - Step 1: transform a point on abstract continuous texture plane to a point in discrete texture map
  - Step 2: Get color at transformed point in texture image, for e.g., (0.0, 0.0) => (0, 0), (1.0, 1.0) => (100, 100), (0.75, 0.45) => (75, 45)

In general, for any point \((u, v)\) on unit plane, the corresponding point on the texture image is: \((u \times \text{pixmap width}, v \times \text{pixmap height})\).

There are infinitely many points on unit plane so we will get sampling errors.
The \(uv\) plane is continuous version of the texture pixmap.
Now the unit \(uv\) square acts as stretchable rubber sheet to wrap around the texture mapped object.

Cylinders and Cones

- Imagine a standard cylinder or cone as a stack of circles. Use position of point on perimeter to determine \(u\), the height of point in stack to determine \(v\). Map top and bottom caps separately, as onto a plane
- Calculating \(v\): height of point in object space, which ranges between \([-0.5, 0.5]\), gets mapped to a \(v\) range of \([0, 1]\).
- Calculating \(u\): map points on circular perimeter to \(u\) values between 0 and 1; \(0\) radians is 0, \(2\pi\) radians is 1.
- Then \(r = r / 0.5\). Those mappings are arbitrary: any function mapping the angle around the cylinder \((\theta)\) to the range \([0, 1]\) will work.
Cylinders and Cones

- Want to convert a point P on the perimeter to an angle. θ is measured clockwise due to the right-handed coordinate system, but one still expects u to increase as we travel circle counter-clockwise, as shown by colored arrows below.

- Circle is not necessarily a unit circle
- So use arctangent instead of sine or cosine
- Use atan2 to get whole circle (atan gives only half a circle).

Sphere

- A sphere is a stack of circles of varying radii.
- Calculating u: as explained previously.
- Calculating v: v is the latitude and varies from 0 to 1.

- When v is 0 or 1, there is a singularity and u should equal a specific value.
- Do not specifically check for this because we will rarely see this value within floating point precision.

Complex Geometries

- Texture mapping of simple polygons was easy.
- How do we texture map more complicated shapes? E.g. A house
- How should we texture map it?
- We could texture map each polygonal face of the house (we know how to do this already as they are planar).

- This causes discontinuities at edges of polygons. We want smooth mapping without edge discontinuities
- Intuitive approach: Pretend the house is a sphere for texture-mapping purposes
- Texture mapping is raytracing

- Another Method using an inscribed sphere
- Assume intersection point lies on some sphere and calculate its uv coordinates using spherical projection.
- Use the distance from the center of the object to the intersection point as radius for the sphere.
- The sphere changes with the point’s location.
Environment Mapping

- Faking reflections

Cube (environment) map is made by stitching together 6 projective textures.

[Image of environment mapping]

Shadow Mapping

- Faking shadows
  - Transform camera to each directional light source
  - Render the scene from light source point of view (keeping the same far clipping plane), only updating Z buffer
  - Read Z buffer into a texture map (the shadow map)
  - While rendering the scene from the original eye point, convert every pixel to light space
  - If in light space the distance from the pixel to the light is greater than value in shadow map, it is in shadow
  - Major aliasing problem because size of shadow maps is limited
  - Implemented on hardware

[Image of shadow mapping]

Texture Aliasing

- This is expensive to do if averaging is done over large areas repeatedly

Mipmapping alleviates this problem by pre-calculating the average colours.
In the new, smaller texture map, each pixel contains the average of four pixels from the original texture.

[Image of mipmapping]

[Image of filtering]

[Image of supersampling]
Surface Detail

- Ideal Geometry
- Adjust normals with a bump map.
- Approximate geometry with a displacement map.
- Replace surface normals with a normal map.

Bump Mapping

- Bump mapping uses a single-channel (grayscale) map to perturb existing normals on the surface;
  - treat grayscale map as height field
  - perturb the surface normals by the gradient of the map
  - lighting is calculated using the normal
- Normal mapping uses a multichannel (rgb) map to completely replace existing normals. RGB values of each pixel correspond to the x,y,z components of the normal vector
  - x,y,z components of the mapped normals are usually in object-space but other spaces are sometimes used
  - level of detail of output render is limited by resolution of normal maps instead of by the number of polygons in rendered meshes
  - runtime surface normal retrieval is trivial—it's just a look-up—making this technique particularly useful for real-time applications
- Limitations:
  - silhouette do not reflect added detail
  - though runtime is easy, initial creation of normal maps is not straightforward

Normal Mapping

- Normal mapping can be used to fake additional geometry.
- Creation of meaningful normal maps is not simple
- Unlike bump maps, normal maps can not simply be painted by hand because color in a normal map is constantly varied and r,g,b values are spatially meaningful.
- Normal maps must be generated by specialized software, such as Pixologic's ZBrush or the free normal map editor, NormalEdit. (www.normality.com)
- A high resolution model is created by the artist and its normals are used to generate maps for lower-resolution of the model.
Displacement Mapping

Actual geometric position of points over the surface are displaced along the surface normal according to the values in the texture.

http://www.xbitlabs.com/articles/video/display/matrox-parhelia.html