Texture Mapping

OUTLINE:

- Uses for Texture Mapping
- Parameterization
- Rendering Textures

Uses for Texture Mapping

The idea: map 2-D image or 3-D volume onto surface to add surface detail, so surfaces don't look so smooth, perfect, and unrealistic.

It's a lot cheaper than modeling the details with geometric primitives.

Texture can be used to affect a variety of shading parameters.

If your shading formula is

radiance = k_{dr}*(light radiance)*(reflectance)*(N.L) + k_{sr}*(light rad.)*$(N.H)^e$,

for example, you could use texture to affect:

surface reflectance	- **surface color texture mapping** (most common type)
normal vector N	- **bump mapping**
coefficients k_{dr}, k_{sr}, e	- specularity mapping?
light radiance	- **environment mapping**
geometry	- **displacement mapping**
transparency	- **transparency mapping**

Texture Types

The texture data can be a 2-D image or a 3-D solid texture.

- The coordinates of the texture are called the *texture parameters*, and the assignment of texture parameters to a surface is called *parameterization*. Texture parameters are commonly denoted u, v, (w).
- Texture images, *texture(u,v)*, are good when the surface has a natural parameterization and/or you want a “decal” or “projected” look. This is the most common type of texture mapping.
- Solid texture, *texture(u,v,w)*, is good when you want a surface to appear carved out of a material (e.g. wood, marble).

Textures can be stored as a raster image, a 3-D discrete grid, or can be computed on the fly procedurally.

Texture Parameterization

2-D parameterization (image texture):

Some objects have natural parameterizations:

- **Sphere**: use spherical coordinates, $(\phi,\theta)=(2\pi u,\pi v)$.
- **Cylinder**: use cylindrical coordinates.
- **Parametric surface** (such as B-spline or Bezier): use existing parameters *(u,v)*.

Parameterization is less obvious for:

- **Polygon**: affine mappings (defined by 3 pts) usually suffice.
- **Implicit surfaces** are very hard to parameterize, so try solid texture.

3-D parameterization (solid texture):

- For solid texture, *(u,v,w)* are usually taken to be world space or object space.
- You can also create solid textures from images by projecting them through space, yielding a slide projector effect.

Texture Rendering

Once we've got the texture image and the parameterization, how do we make a picture?

In a scan conversion algorithm such as z-buffer or painter's:

- Find composite mapping from texture space to screen space; now you've got the same problem as in image warping: a 2-D to 2-D image resampling problem. For polygons, the mapping will often be projective (see Notes 4).
- Scan pixels, at each one use inverse mapping to compute (u,v) from (x,y). This can often be done incrementally. Inner loop cost to compute (u,v) for affine mappings: 2 adds; for projective mappings: 3 adds and 2 divides. Some graphics workstations support this in hardware/microcode.
- Read the pixel nearest (u,v), or, to reduce aliasing, do low pass filtering when reading pixels from texture.

In a ray tracer: Do the same, except 2-D to 2-D mapping can't be precomputed; instead you compute parameters (u,v) on the fly when ray intersects surface.

Texture Filtering

Aliasing is a big problem in texture mapping. It's especially visible in animation.

Aliasing problems crop up in texture mapping much more than in image warping because image warps typically involve scale factors near 1 (no extreme shrinking or expansion), while texture mapping can have extreme shrinking at horizons and silhouettes and extreme expansion due to perspective.

Where the texture is being scaled up, you're *upsampling*, and you need to do **reconstruction** (interpolation between texture pixels). Bilinear interpolation usually suffices. (See Notes 6)

Where the texture is being scaled down, you're *downsampling*, and you need to do **prefiltering** (average together many texture pixels). This is more costly. Unweighted averaging (box filter) yields pretty good results, but weighted averaging with a higher quality low pass filter reduces aliasing a bit more.

Typically the mapping from texture space (u,v) to screen space (x,y) is not affine. This implies that the filtering you do changes with position. This is a shift-variant filter (not the linear, shift-invariant (LSI) filters discussed earlier).

Good **data structures to speed texture filtering**: image pyramid with trilinear interpolation, summed area table. This requires memory & some precomp.