10 Texture Mapping

10.1 Overview

We would like to give objects a more varied and realistic appearance through complex variations in reflectance that convey textures. There are two main sources of natural texture:

- Surface markings — variations in *albedo* (i.e. the total light reflected from ambient and diffuse components of reflection), and
- Surface relief — variations in 3D shape which introduces local variability in shading.

We will focus only on surface markings.

Examples of surface markings and surface relief

These main issues will be covered:

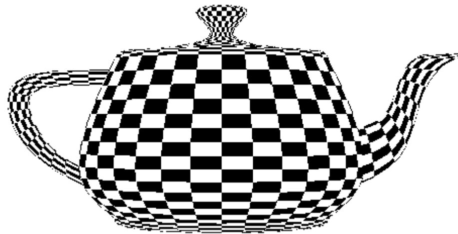
- Where textures come from,
- How to map textures onto surfaces,
- How texture changes reflectance and shading,
- Scan conversion under perspective warping, and
- Aliasing

10.2 Texture Sources

10.2.1 Texture Procedures

Textures may be defined procedurally. As input, a procedure requires a point on the surface of an object, and it outputs the surface albedo at that point. Examples of procedural textures include checkerboards, fractals, and noise.

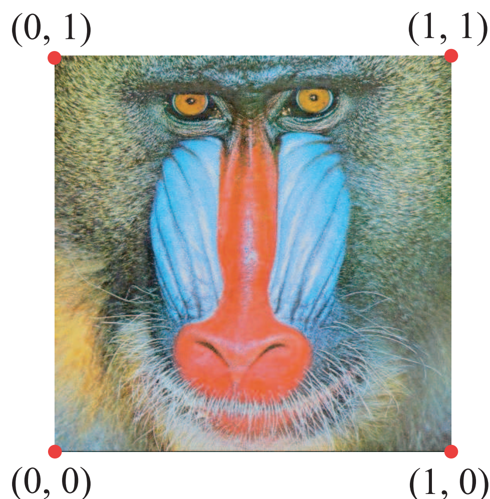
Copyright © 2005 David Fleet and Aaron Hertzmann



A procedural checkerboard pattern applied to a teapot. The checkerboard texture comes from the OpenGL programming guide chapter on texture mapping.

10.2.2 Digital Images

To map an arbitrary digital image to a surface, we can define texture coordinates $(u, v) \in [0, 1]^2$. For each point $[u_0, v_0]$ in texture space, we get a point in the corresponding image.



Texture coordinates of a digital image

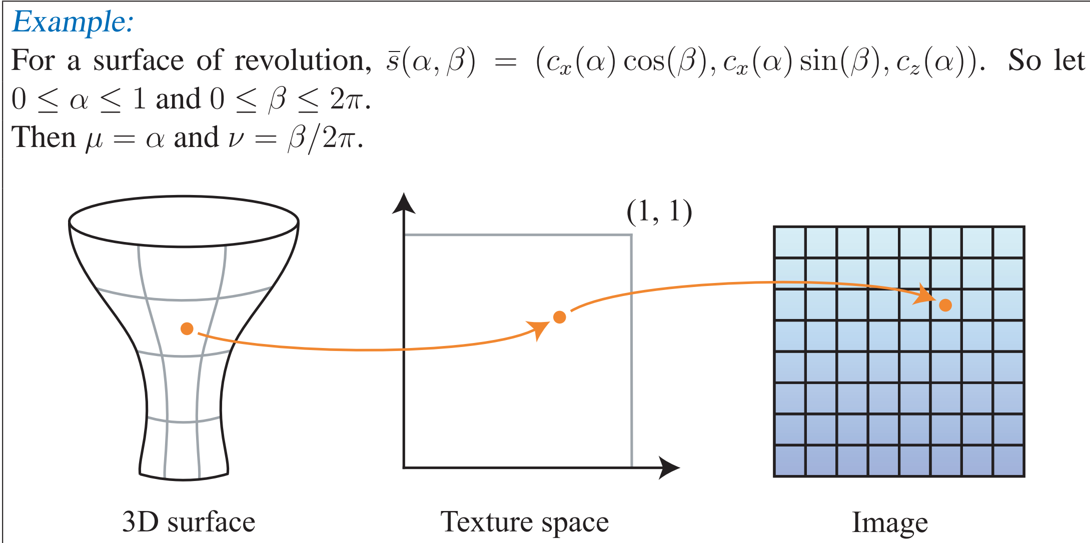
10.3 Mapping from Surfaces into Texture Space

For each face of a mesh, specify a point (μ_i, ν_i) for vertex $\bar{p}_i$. Then define a continuous mapping from the parametric form of the surface $\bar{s}(\alpha, \beta)$ onto the texture, i.e. define m such that $(\mu, \nu) = m(\alpha, \beta)$.

Example:
For a planar patch $\bar{s}(\alpha, \beta) = \bar{p}_0 + \alpha\vec{a} + \beta\vec{b}$, where $0 \leq \alpha \leq 1$ and $0 \leq \beta \leq 1$.
Then we could use $\mu = \alpha$ and $\nu = \beta$.

Copyright © 2005 David Fleet and Aaron Hertzmann

Example:
For a surface of revolution, $\bar{s}(\alpha, \beta) = (c_x(\alpha)\cos(\beta), c_x(\alpha)\sin(\beta), c_z(\alpha))$. So let $0 \le \alpha \le 1$ and $0 \le \beta \le 2\pi$.
Then $\mu = \alpha$ and $\nu = \beta/2\pi$.



10.4 Textures and Phong Reflectance

Scale texture values in the source image to be in the range $0 \le \tau \le 1$ and use them to scale the reflection coefficients r_d and r_a. That is,

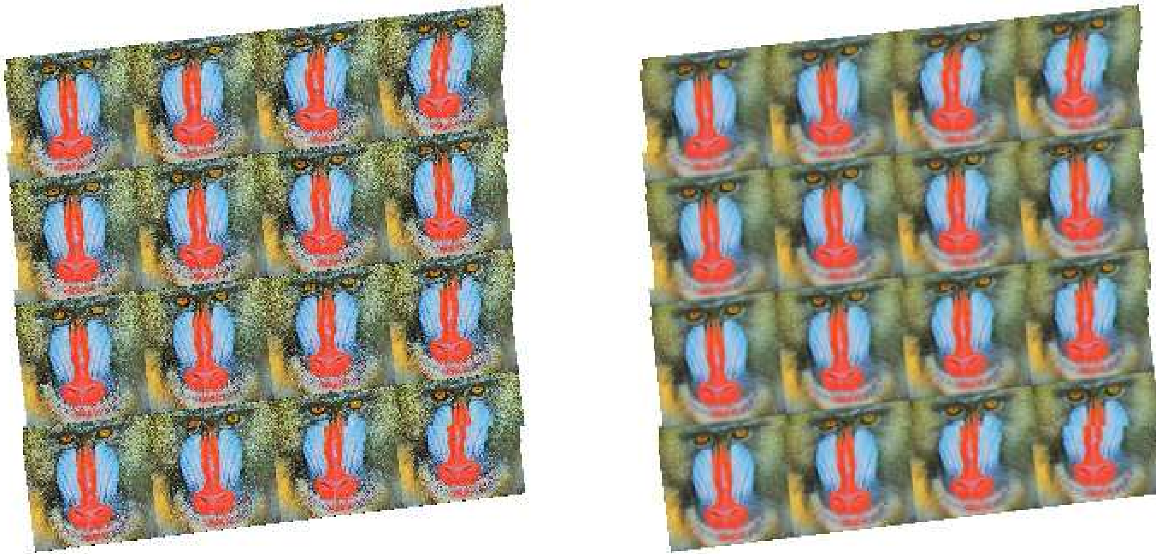
$$\begin{aligned} \tilde{r}_d &= \tau r_d, \\ \tilde{r}_a &= \tau r_a. \end{aligned}$$

We could also multiply τ by the specular reflection, in which case we are simply scaling E from the Phong model.

10.5 Aliasing

A problem with high resolution texturing is aliasing, which occurs when adjacent pixels in a rendered image are sampled from pixels that are far apart in a texture image. By down-sampling—reducing the size of a texture—aliasing can be reduced for far away or small objects, but then textured objects look blurry when close to the viewer. What we really want is a high resolution texture for nearby viewing, and down-sampled textures for distant viewing. A technique called *mipmapping* gives us this by prerendering a texture image at several different scales. For example, a 256x256 image might be down-sampled to 128x128, 64x64, 32x32, 16x16, and so on. Then it is up to the renderer to select the correct mipmap to reduce aliasing artifacts at the scale of the rendered texture.

Copyright © 2005 David Fleet and Aaron Hertzmann



An aliased high resolution texture image (left) and the same texture after mipmapping (right)

10.6 Texturing in OpenGL

To use texturing in OpenGL, a texturing mode must be enabled. For displaying a 2D texture on polygons, this is accomplished with

```
glEnable(GL_TEXTURE_2D);
```

The dimensions of texture in OpenGL must be powers of 2, and texture coordinates are normalized, so that $(0, 0)$ is the lower left corner, and $(1, 1)$ is always the upper right corner. OpenGL 2.0, however, does allow textures of arbitrary size, in which case texture coordinates are based on the original pixel positions of the texture.

Since multiple textures can be present at any time, the texture to render with must be selected. Use `glGenTextures` to create texture handles and `glBindTexture` to select the texture with a given handle. A texture can then be loaded from main memory with `glTexImage2D` For example:

```
GLuint handles[2];
glGenTextures(2, handles);

glBindTexture(GL_TEXTURE_2D, handles[0]);
// Initialize texture parameters and load a texture with glTexImage2D

glBindTexture(GL_TEXTURE_2D, handles[1]);
// Initialize texture parameters and load another texture
```

Copyright © 2005 David Fleet and Aaron Hertzmann

There are a number of texture parameters that can be set to affect the behavior of a texture, using `glTexParameteri`. For example, texture wrap repeating can be enabled to allow a texture to be tiled at the borders, or the minifying and magnifying functions can be set to control the quality of textures as they get very close or far away from the camera. The texture environment can be set with `glTexEnvi`, which controls how a texture affects the rendering of the primitives it is attached to. An example of setting parameters and loading an image follows:

```
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP)
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, imageWidth, imageHeight,
             0, GL_RGB, GL_UNSIGNED_BYTE, imagePointer);
```

Mipmaps can be generated automatically by using the GLU function `gluBuild2DMipmaps` instead of `glTexImage2D`.

Once a texture is bound and texturing is enabled, texture coordinates must be supplied for each vertex, by calling `glTexCoord` before `glVertex`:

```
glTexCoord2f(u, v);
glVertex3f(x, y, z);
```

When textures are no longer needed, they can be removed from the graphics hardware memory with

```
glDeleteTextures(2, handles);
```