Rendering Techniques

Texture maps

- Color and opacity
- Illumination
  - Environment maps
  - Shadow maps
  - Irradiance maps
- Reflectance
  - Reflectance maps
- Geometry
  - Bump maps
  - Displacement maps

Shading and texturing languages

Texture Maps

How is texture mapped to the surface?

- Dimensionality: 1D, 2D, 3D
- Texture coordinates (s,t)
  - Surface parameters (u,v)
  - Direction vectors: reflection R, normal N, halfway H
  - Projection: cylinder
  - Developable surface: polyhedral net
  - Reparameterize a surface: old-fashion model decal

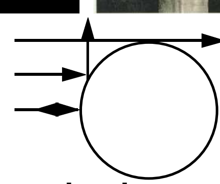
What does texture control?

- Surface color and opacity
- Illumination functions: environment maps, shadow maps
- Reflection functions: reflectance maps
- Perturb geometry: bump and displacement maps

Tom Porter's Bowling Pin

RenderMan Companion, Pls. 12 & 13

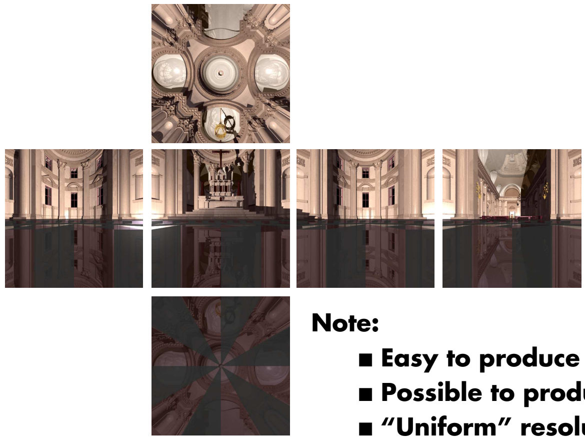
Gazing Ball



Note:

- Photograph of reflective ball
- Reflection indexed by normal
- Maps entire field of view to circle
- Resolution function of orientation; maximum head-on
- Alternatives: Fish eye, map projections

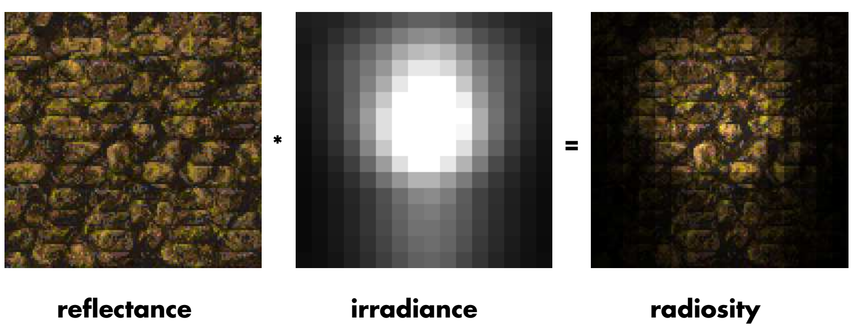
Cubical Environment Map



Note:

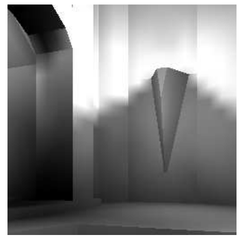
- Easy to produce with rendering system
- Possible to produce from photographs
- "Uniform" resolution
- Simple texture coordinates calculation

Illumination Maps



Quake Light Maps

*



Lower resolution

Projected Textures

- Project light source into the scene

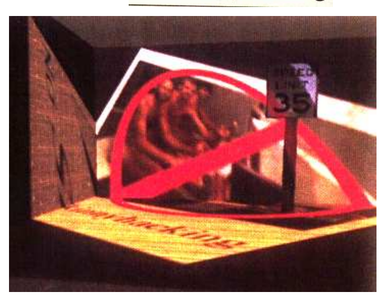
For each pixel

1. Transform from e.c.s. to l.c.s.

   Requires 4x4 matrix multiply and both (x,y) and z coordinates

2. Compute texture coordinates
   (u,v) = (x/w, y/w)

3. Lookup color in textures



From Segal et al.

Approximate Shadow Maps

William's Shadow algorithm

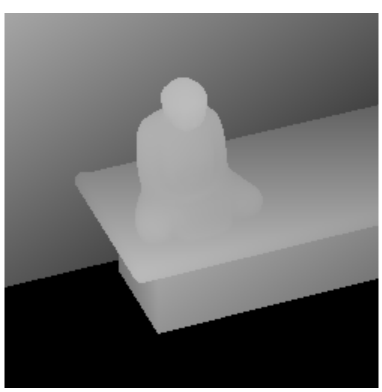
- **Step 1: Render z-buffer as seen from light source**
- **Step 2. Render scene as seen from the eye**
- **Step 3. Generate shadow matte**
  - **Transform each pixel in eye c.s. into light c.s.**
  - **Alpha = (ze < zl)**
- **Step 4. Modulate color by shadow matte**

Problems:

- **Self-shadowing (introduce bias, Woo's algorithm)**
- **Good filtering (precentage closer filtering)**
- **Attenuates after reflection, not before!**

Correct Shadow Maps

May incorporate shadow maps into lighting calculations



Correct Shadow Maps

Multi-pass shadow algorithm

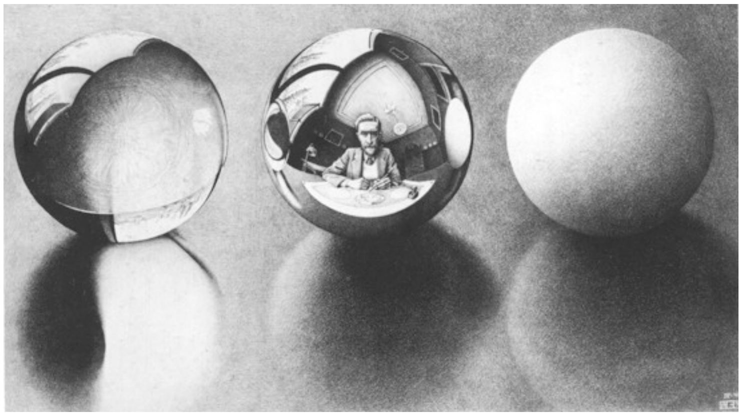
- **Step 1: Render z-buffer as seen from light source**
- **Step 2. Render scene as seen from the eye**
  - **Compute z-buffer and ambient lighting**
- **Step 3: For each light**
  - **A. Redraw scene (ZEQ) to generate shadow matte**
    - **When drawing transform pixels from e.c.s. to l.c.s.**
    - **Alpha = (ze < zl)**
  - **B. Accumulate shaded image attenuated by alpha**

Shadow Mesh Algorithm

Multi-pass shadow mesh algorithm

- **Step 1: Render z-buffer as seen from light source**
- **Step 2. Render scene as seen from the eye**
- **Step 3: Draw shadow mesh keep tracking of parity**
- **Step 4: Generate shadow matte and attenuate image**
  - **Alpha = (Parity == 1)**

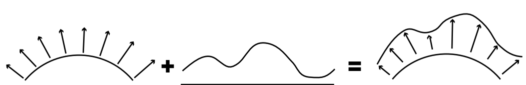
Reflection Maps



Integrate over a hemisphere: BRDF * L

Very low resolution often sufficient: At NYIT 49x49

Displacement/Bump Mapping



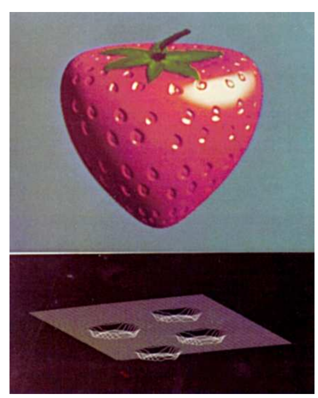
Offset surface position

- **Displacement**

$$\mathbf{P}'(u,v) = \mathbf{P}(u,v) + h(u,v)\mathbf{N}(u,v)$$

- **Perturb normal**

$$\mathbf{N}(u,v) = \frac{\partial \mathbf{P}(u,v)}{\partial u} \times \frac{\partial \mathbf{P}(u,v)}{\partial v}$$



From Blinn 1976

Shading and Texturing Language

Flexibility

- **Control over mapping onto surface**
- **Modulate multiple parameters**

Texture access

```
float/color texture( "image", s, t, ...)
point bump("heights", N, Ps, Pt, s, t, ...)
float/color environment("cubefaces", D, ...)
float shadow("depths", P, ...)
```

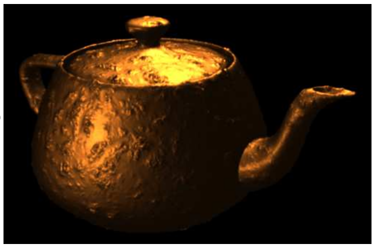
Create procedural texture models/solid textures

RenderMan Surface Shader

```
surface corrode(float Ks=0.4, Ka=0.1, rough=0.25)
{
  float i, freq=1, turb=0;

  // compute fractal texture
  for( i=0; i<6; i++ ) {
    turb += 1/freq * noise(freq*P);
    freq *= 2;
  }
  // perturb surface
  P -= turb * normalize(N);
  N = faceforward(normalize(calculatenormal(P)));

  // compute color
  Ci = Cs*(Ka*ambient()+Ks*specular(N,I,rough));
}
```



Perlin's Noise Function

1. Generate a table of random numbers
2. Hash a 3D lattice into a table entry
3. Use random values as the gradient
4. Perform cubic interpolation
5. Scale and add octaves