Computer Graphics Models

For example, SGI shading model

$$L_r = k_a L_a + \sum \left(k_d (\mathbf{N} \bullet \mathbf{L}_i) + k_s (\mathbf{N} \bullet \mathbf{H}_i)^s \right) L_i$$

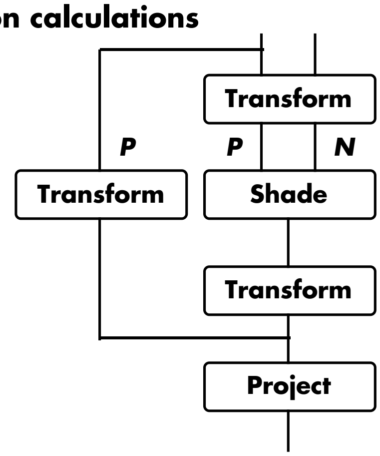
Terms:

- **Ambient - Hack!**
- **Diffuse**
- **Specular**

Lighting Calculations

Illumination and reflection calculations

- **Normals**
- **Coordinate systems**
  Lighting calculations are derived from laws of physics must be done in cs metric to world

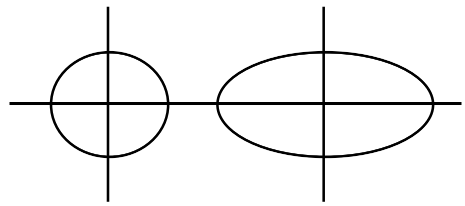


Transformation of Normals

Normals remain perpendicular to tangents

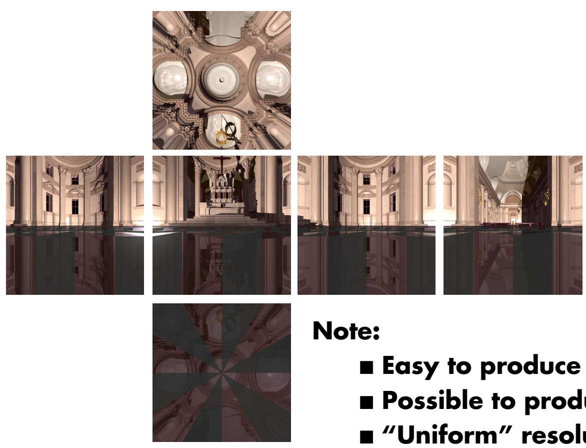
$$N \bullet V = 0$$

$$N' \bullet V' = 0$$



Rotations special case!

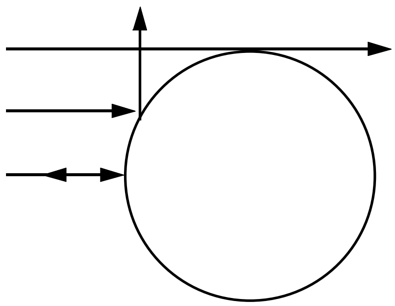
Cubical Environment Map



Note:

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

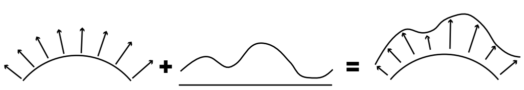
Gazing Ball

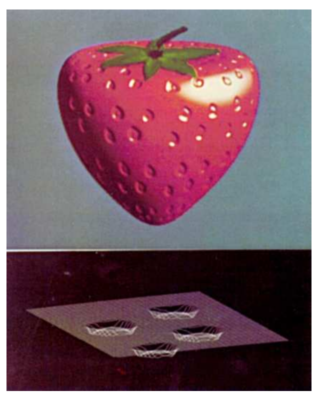


Note:

- **Photograph of reflective ball**
- **Reflection indexed by normal**
- **Maps entire field of view to circle**
- **Resolution function of orientation; maximum head-on**
- **Fish eye camera lens similar**

Displacement/Bump Mapping





From Blinn 1976

Offset surface position

- **Displacement**

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

- **Perturb normal**

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

Shadow Maps

- **Shadow mattes / Slide projector**

- **Light depth buffers**

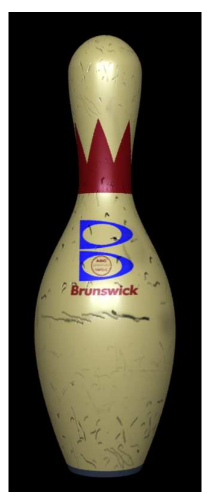
  Step 1: Render z-buffer as seen from light source

  Two alternatives while rendering (from the eye)

  Step 2A: Transform (x,y,z)'s into light coordinate system and test if they are hidden

  Step 2B: Rendering shadow z-buffer as mesh and keep track of parity

Tom Porter's Bowling Pin



RenderMan Companion, Pls. 12 & 13

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

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