CS174A Lecture 13

Announcements & Reminders

- ***Projects due: Dec 1***
- ***Project presentations: Dec 3 and 5, in class***
- ***Final exam: Dec 12, 11:30 AM – 2:30 PM, Place TBD***

TA Session This Friday

- *Return Midterms*
- *Project #4*
- *Collision Detection*

Last Lecture Recap

- ***Barycentric Coordinates, Trilinear Interpolations***
- ***Flat and Smooth Shading***

Next Up

- ***Non-Photorealistic Rendering***
- ***Global Illumination***
- ***Mappings: Texture, Bump, Displacement, Environment***
- ***Shadows***
  - 2-pass z-buffer algorithm
  - Shadow volumes
- ***Hidden Surface Removal***
  - Ray casting

Shading Recap

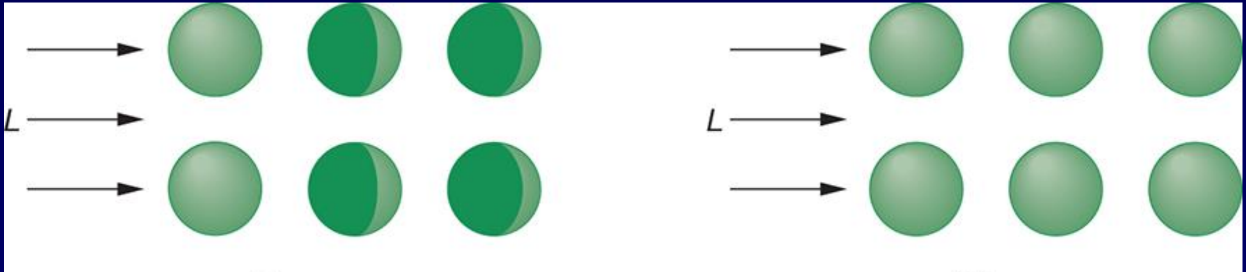
- ***Flat Shading***
  - Illuminate a poly only once
  - No interpolation
- ***Gouraud Shading***
  - Illuminate vertices of poly
  - Interpolate colors at vertices
- ***Phong Shading***
  - Illuminate each point inside poly
  - Interpolate normals at vertices

Non-Photorealistic Shading

- *Cartoon like effects*
- *Color = (L·N > 0.5) ? Color1 : Color2*
- *Color changes with object's shape and light's position*
- *Silhouette: (V·N < 0.01) ? Black : Color*

Global Illumination

- ***Ray Tracing: shadows, reflections, refractions***
- ***Radiosity***
  - Based upon light energy conservation
  - Requires solution of a large set of equations involving all surfaces



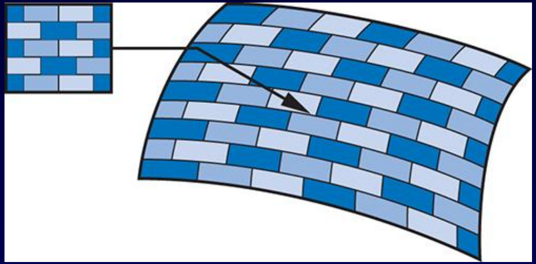
(a) (b)

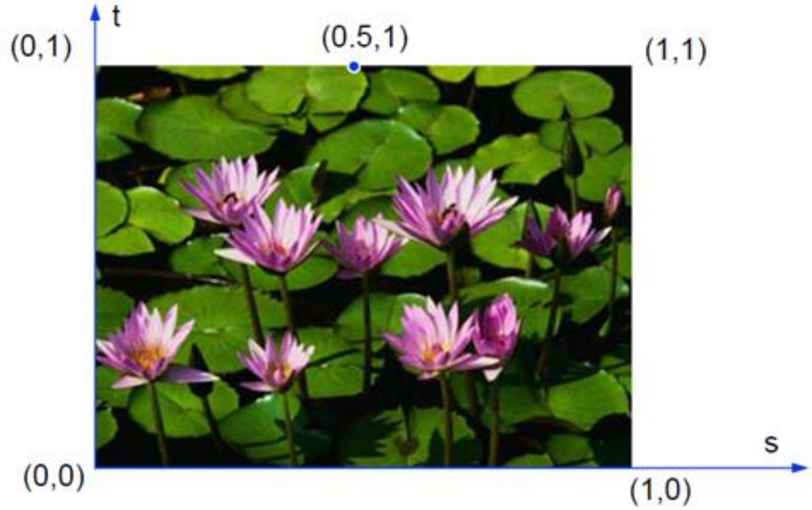
Mappings

- *Texture Mapping*
- *Bump Mapping*
- *Displacement Mapping*
- *Environment Mapping*
- *Procedural Mapping*

Texture Mapping

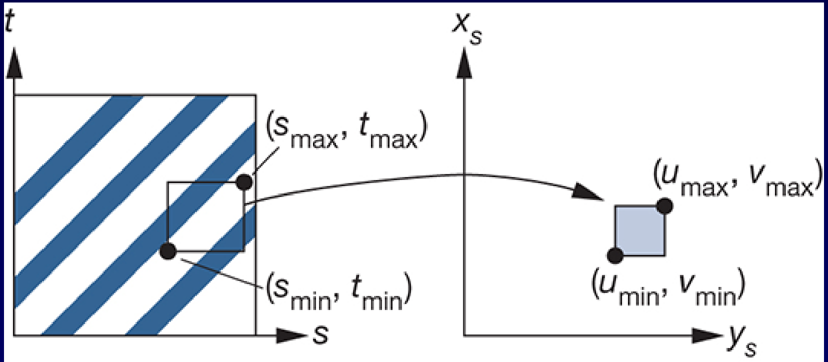
- *AKA Pattern mapping*
- *Map a digitized image onto a poly face*
- *Individual elements are called Texels*
- *u,v and s,t coordinates*
- *Use texel color as diffuse color*

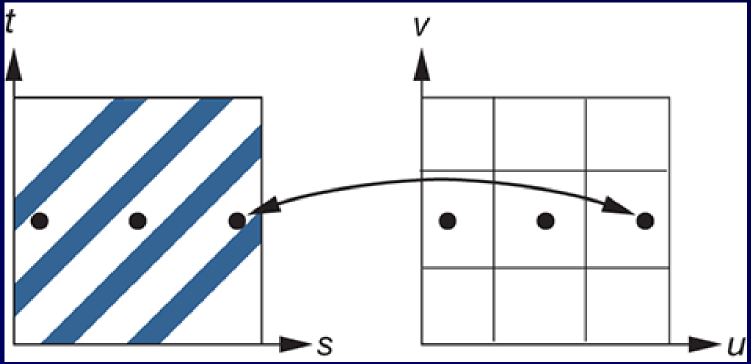




Texture Mapping - Aliasing

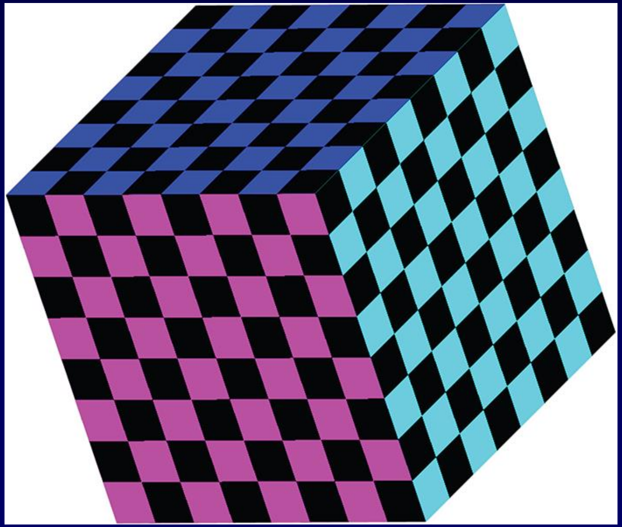
- *Due to high-frequency patterns*

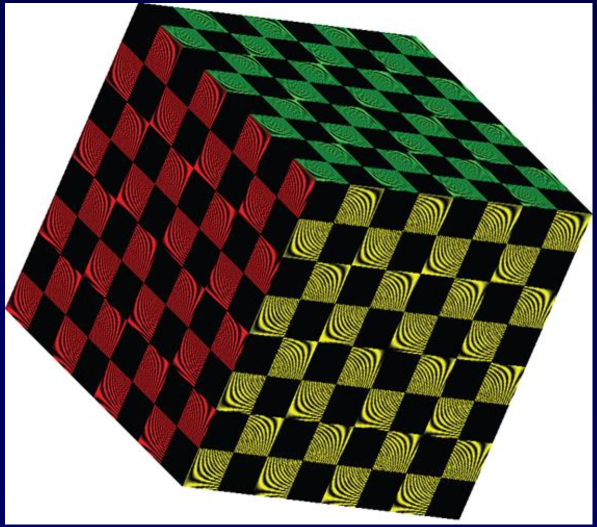




Multi-Texturing

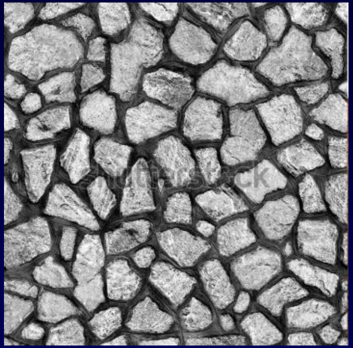
- *Apply multiple textures on the same object – blend*





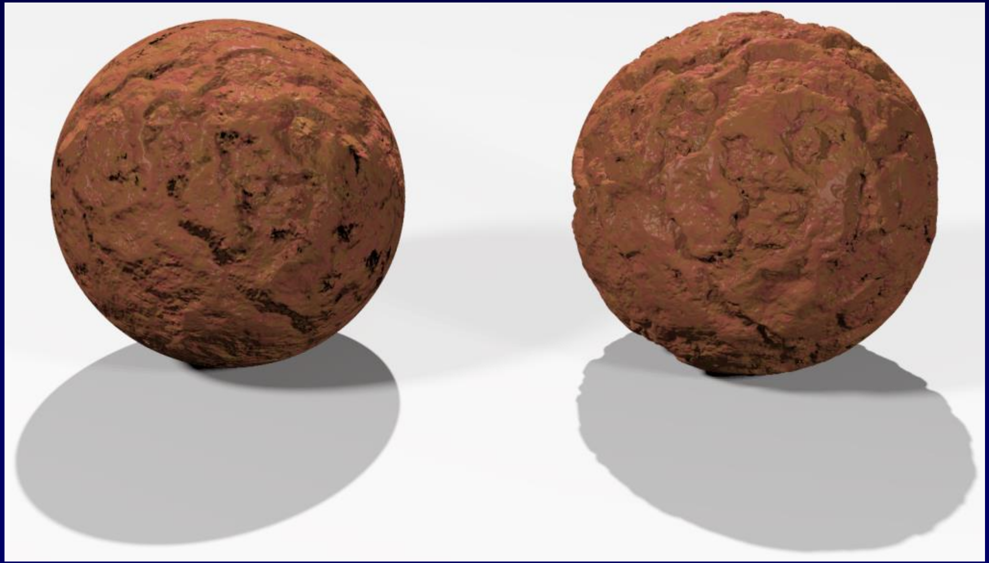
Bump Mapping

- *Displace point on surface of object*
- *P' = P + B * N*
- $N' = \frac{B_u(NxP_t) - Bv(NxPs)}{|N|}$
- *Bu,Bv = partial derivatives of B wrt u,v*
- *Ps,Pt = partial derivatives of P wrt s,t*



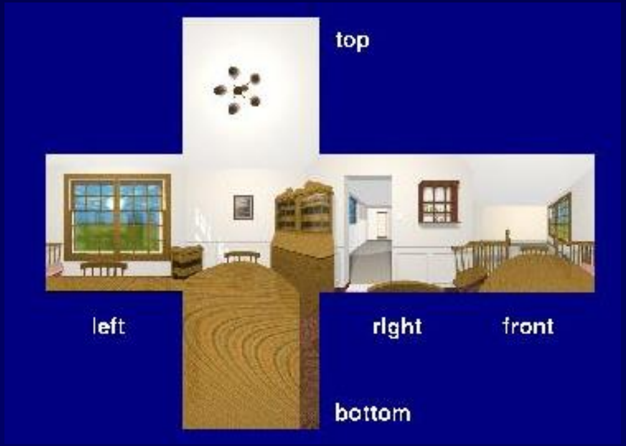
Displacement Mapping

- *Points actually move*
- *Requires hidden surface removal*



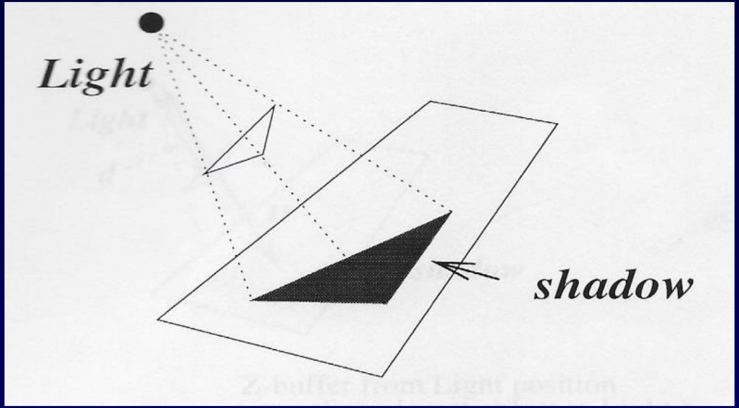
Environmental Mapping

- *AKA Reflection Mapping*
- *Use polar (or spherical) coordinates of reflected ray to map*
- *Map defined usually on 6 faces of a box (cube map) or a sphere*



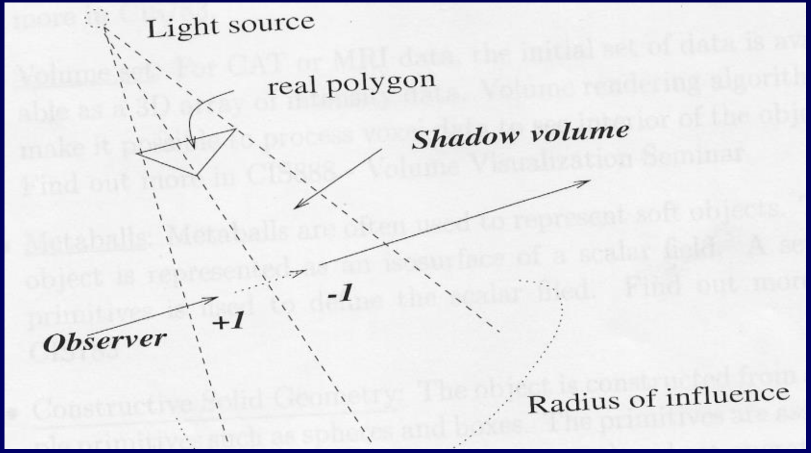
Shadow Algorithms

- ***Types***
  - **Image precision**: 2-pass z-buffer method
  - **Object precision**: shadow volume method
- ***Main Idea***
  - P is not visible from light source ⇔ P is in shadow
- ***Strategy***
  - Identify areas on polys which are not directly visible from light source
  - Mark these areas
  - Do HSR from eye position
  - Areas marked for shadow are rendered only with ambient light

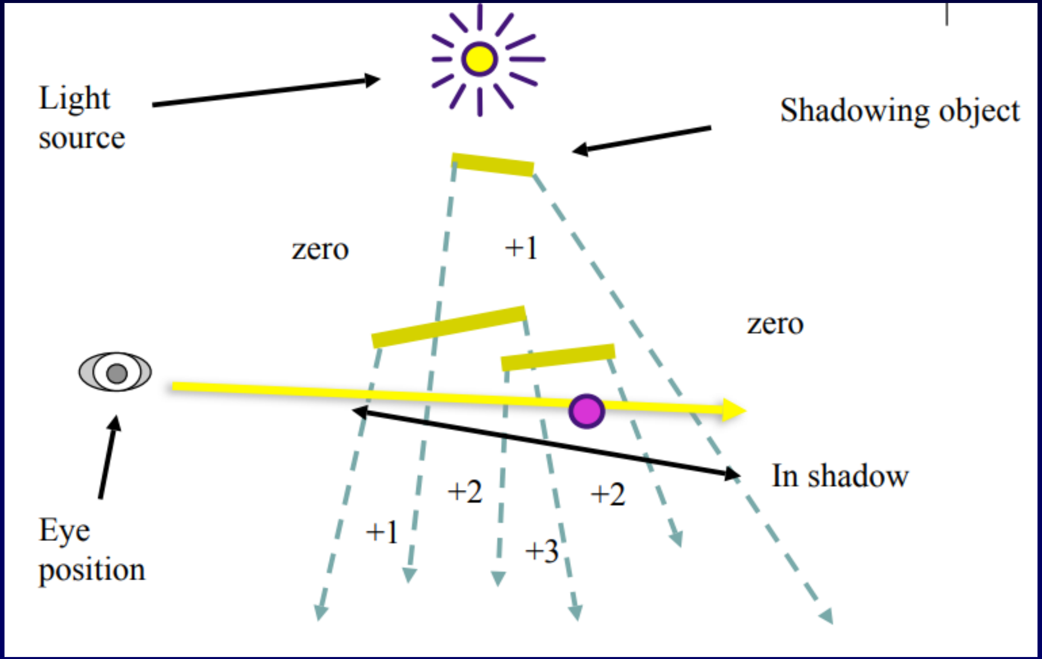


Shadow Volume Method

1. ***Create a shadow volume for each front facing poly***
2. ***Put shadow volume in poly database***
3. ***Do parity test to determine if a visible point is in shadow***
    a. *Initial value = # of shadow volumes containing eye position*
    b. *Increment for front-facing shadow poly*
    c. *Decrement for back-facing shadow poly*
    d. *If parity > 0 ⇒ point is in shadow*



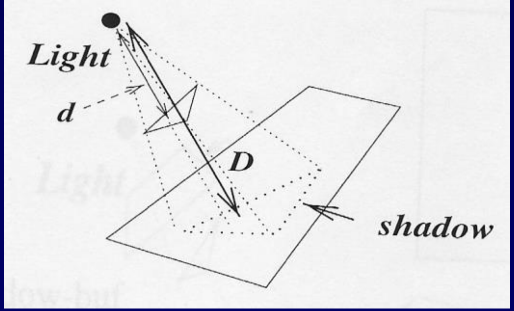
Shadow Volume Method



2-Pass Z-Buffer Method

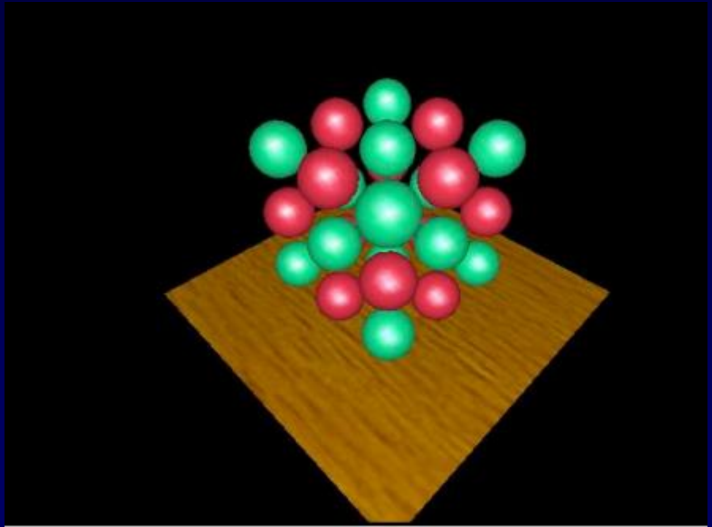
Aka Shadow Map method

1. ***Do z-buffer (full) from light position***
   a. store results in shadow buffer
2. ***Do z-buffer (scanline or full) from eye position***
   a. For each (visible) pixel in scanline
      i. *Do inverse map point to WS*
      ii. *Map to SS of shadow buffer*
      iii. *Compare z with that of shadow buffer at x,y*
      iv. *If shadow_buf[x][y] < z, then point is in shadow*

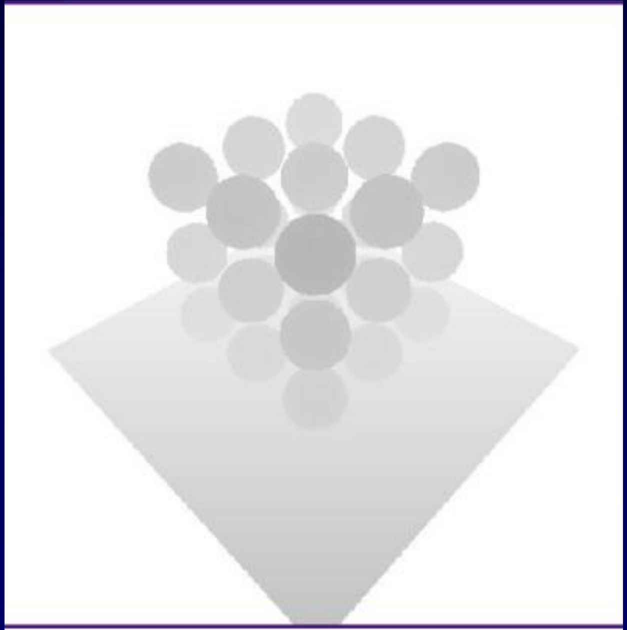


2-Pass Z-Buffer Method

First Pass: from light's position



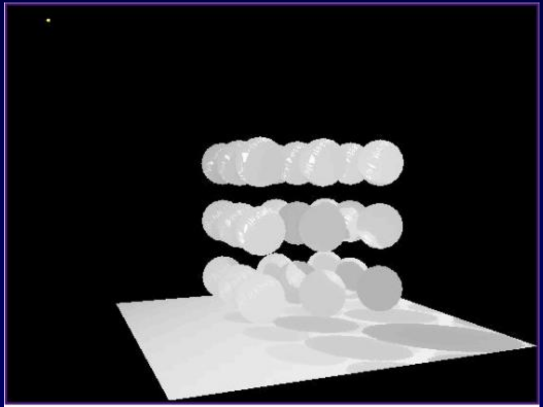
View from light



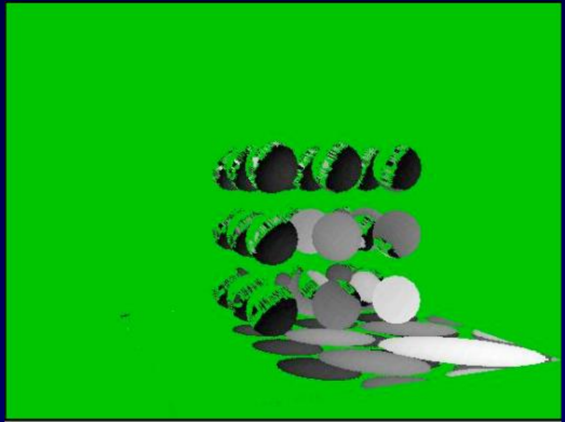
Depth Buffer (shadow map)

2-Pass Z-Buffer Method

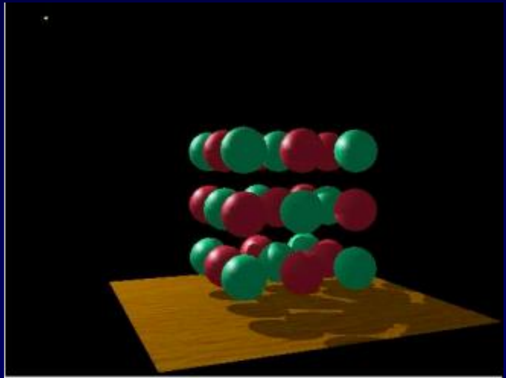
Second Pass: from eye's position



Visible surface depth



Non-green in shadow



Final Image

2-Pass Z-Buffer Method (Contd.)

- ***Advantages***
  - Simple to implement
- ***Disadvantages***
  - Shadow distant from light source may appear blocky
  - A large size of shadow-buffer is required
  - Depth buffer bit resolution – usually 8-bit
  - Umbra vs. penumbra
  - Light source in the view volume is problematic