CS174A Lecture 13

Announcements & Reminders

- ***11/20/22: A4 due***
- ***11/22/22: Team project proposals due, final version***
- ***11/24/22-12/03/22: Student evaluations of course/instructors/TAs***
- ***11/29/22: Prof Demetri’s talk***
- ***12/02/22 (Discussion Sessions): Team project presentations***
- ***12/05/22-12/06/22: Office hours for final exam, see Canvas***
- ***12/06/22: Final Exam, 6:30-8:30 PM PST, in class, in person***

Discussion Session This Friday

- *Team project demo logistics*
- *Assignment #4*

Last Lecture Recap

- *Barycentric Coordinates, Bilinear 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
- ***Illumination in WS/ES, interpolation in SS***

Illumination Recap

$$I = k_a * I_a * O_d + f_{att} * k_d * I_p * (N \cdot L) * O_d + f_{att} * k_s * I_p * (R \cdot V)^n$$

Material Properties:

k_a, k_d, k_s: ambient, diffuse, specular reflection coefficients of object

O_d: color of object

I_a: intensity of ambient light

I_p: intensity of point light

n: specular exponent

Geometric Properties:

N: orientation of object at point being illuminated

L: depends on location of point and light

R: depends on N and L

V: depends on location of point and eye

f_{att}: depends on distance between point and light

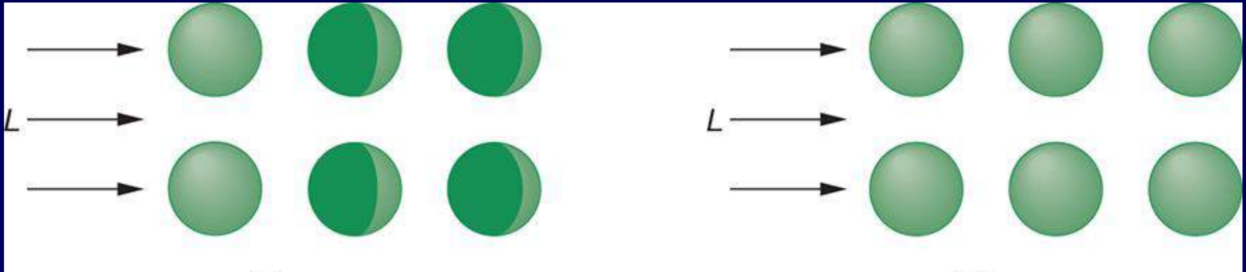
A_{att}: depends on distance between point and eye

Non-Photorealistic Shading

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



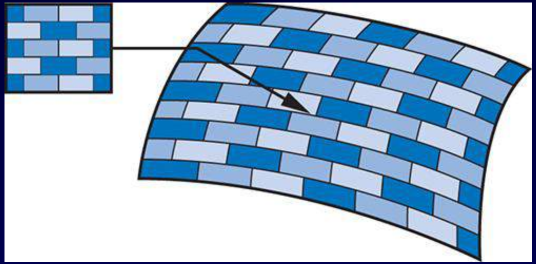
(a) (b)

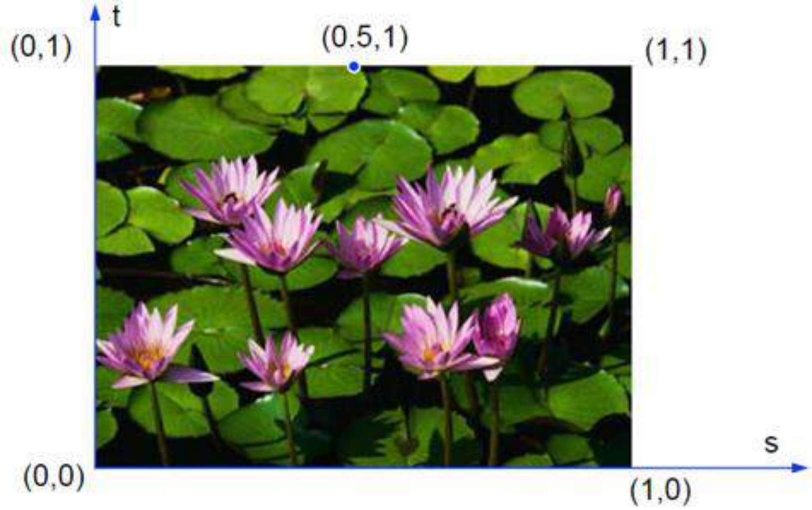
Mappings

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

Texture Mapping

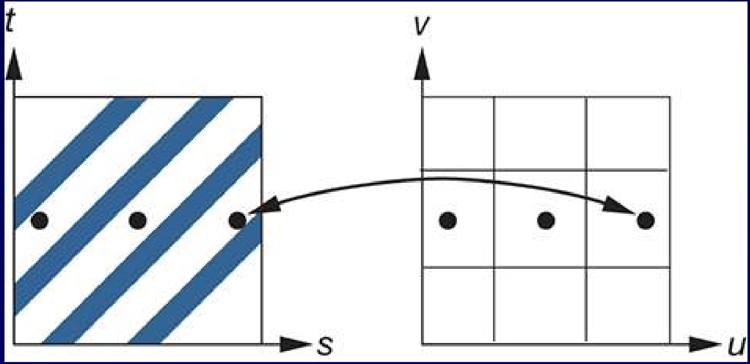
- *AKA Pattern mapping*
- *Map a digitized image onto a poly face*
- *Individual elements are called Texels*
- *u,v and s,t coordinates*
- *Use texel color as diffuse color*

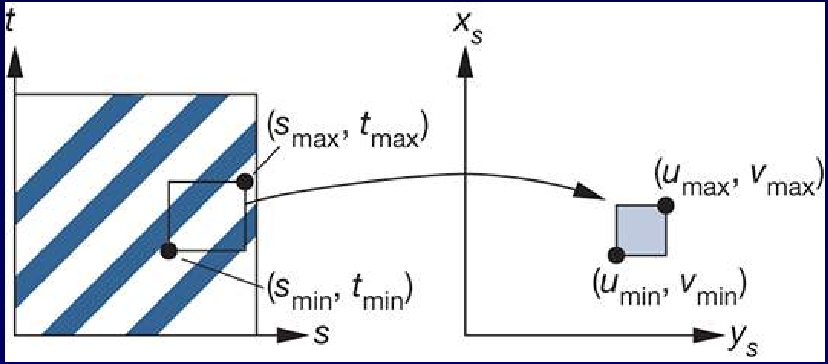




Texture Mapping: Aliasing

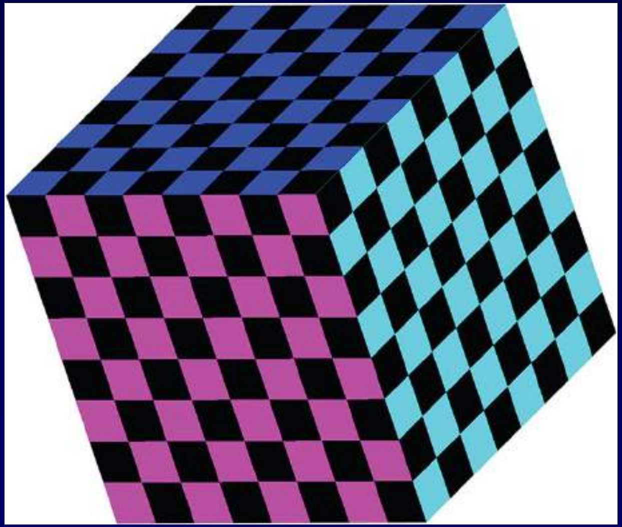
- ***Due to high-frequency patterns***

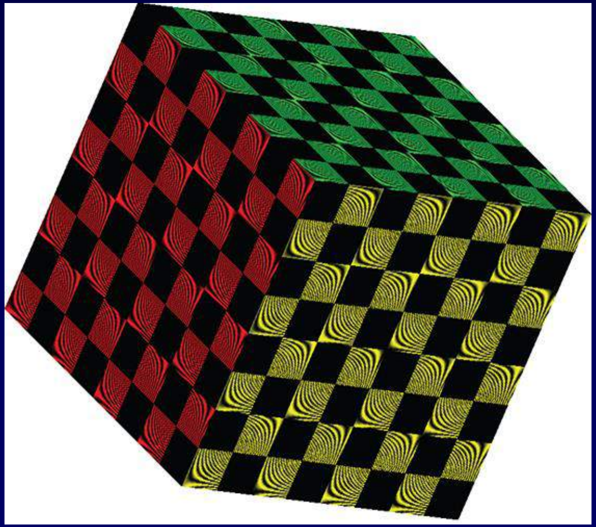




Multi-Texturing

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



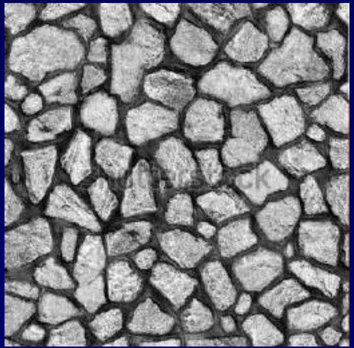


Bump & Displacement Mappings

- ***Displacement Mapping:***
  - Displace point on surface of object
  - Change object's geometry

$$P' = P + B * N$$

- ***Bump mapping***
  - Distort normal to simulate bump without altering object's geometry



$$N' = \frac{B_u(NxP_t) - B_v(NxP_s)}{|N|}$$

B_u,B_v = partial derivatives of B wrt u,v

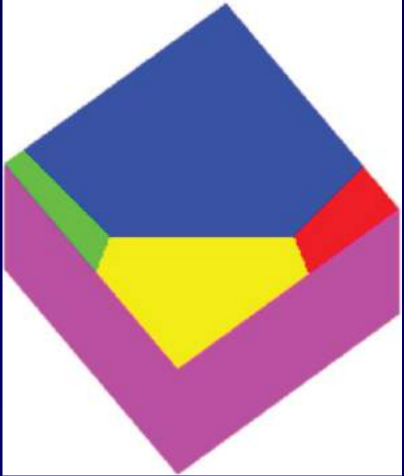
P_s,P_t = partial derivatives of P wrt s,t

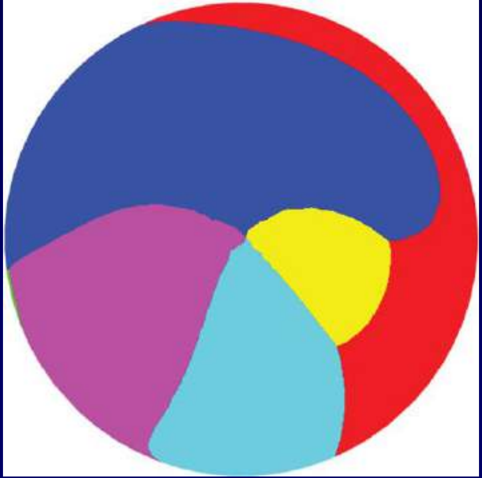
Displacement Mapping

- *Points actually move*
- *Requires hidden surface removal*
- *Bump vs. Displacement 1*
- *Bump vs. Displacement 2*

Environmental Mapping

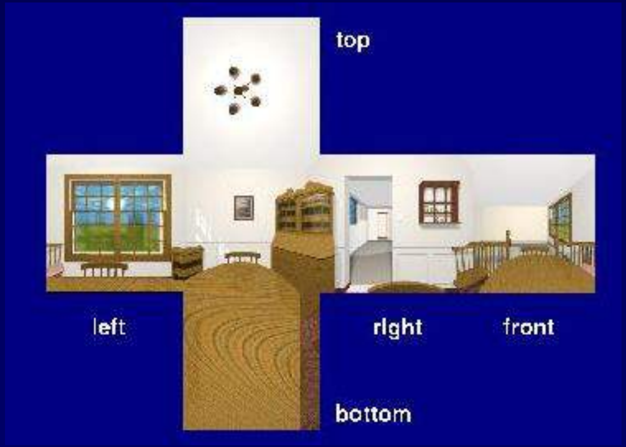
- *AKA Reflection Mapping*
- *Use polar (or spherical) coordinates of reflected ray to map*
- *Map defined usually on 6 faces of a box (cube map) or a sphere*





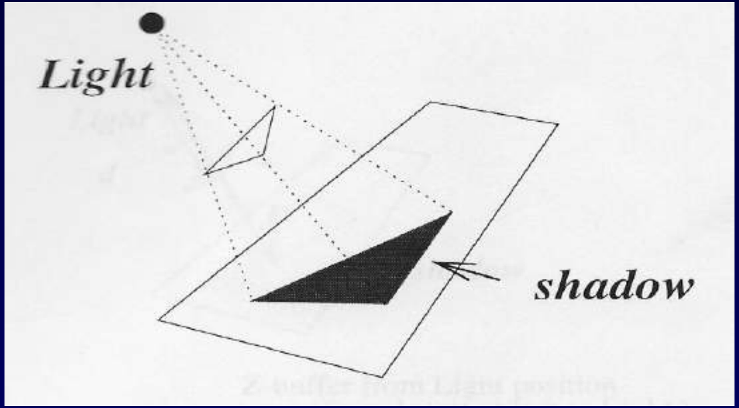
Environmental Mapping

- *More examples*
- $I = ambient + diffuse + specular + k_r * I_r$
  - k_r: coefficient of reflection
  - I_r: reflection intensity



Shadow Algorithms

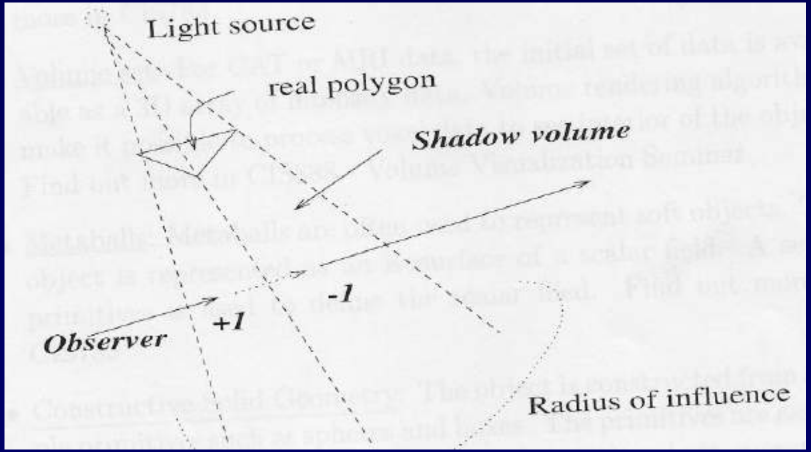
- ***Types***
  - **Object precision**: shadow volume method
  - **Image precision**: 2-pass z-buffer method
- ***Main Idea***
  - P is not visible from light source ⇔ P is in shadow



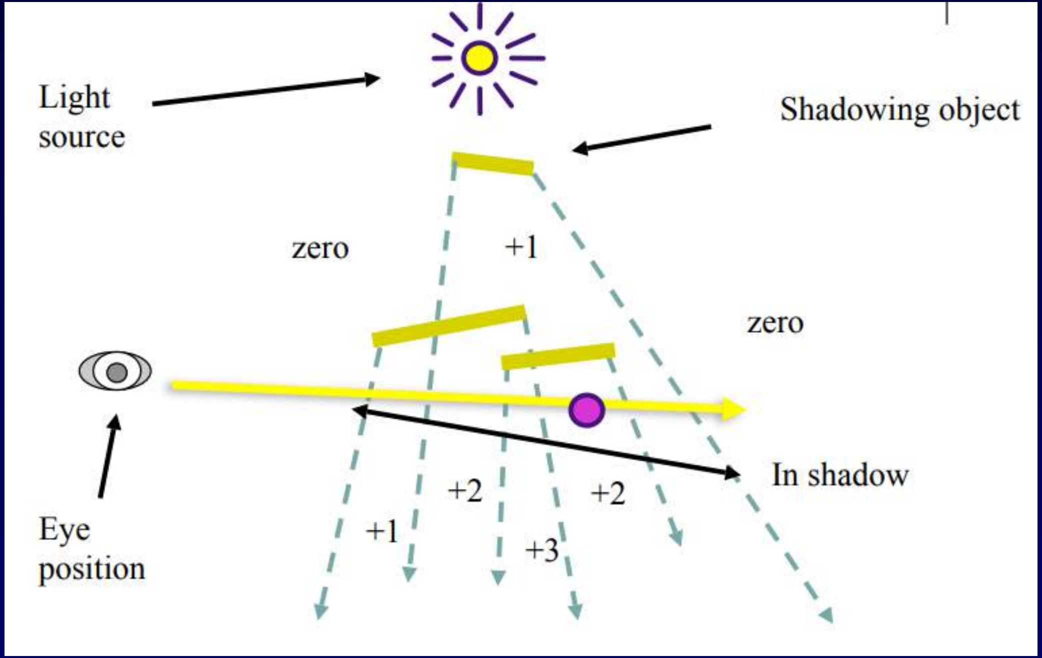
- ***Strategy***
  - Identify areas on polys which are not directly visible from light source
  - Mark these areas
  - Do HSR from eye position
  - Areas marked for shadow are rendered only with ambient light

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*



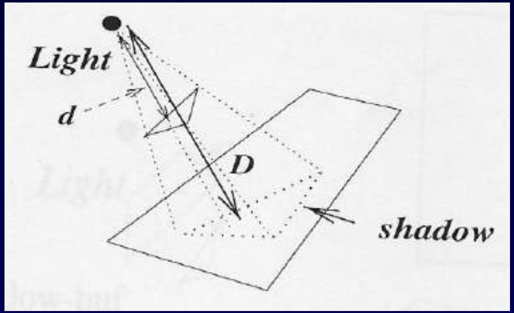
Shadow Volume Method

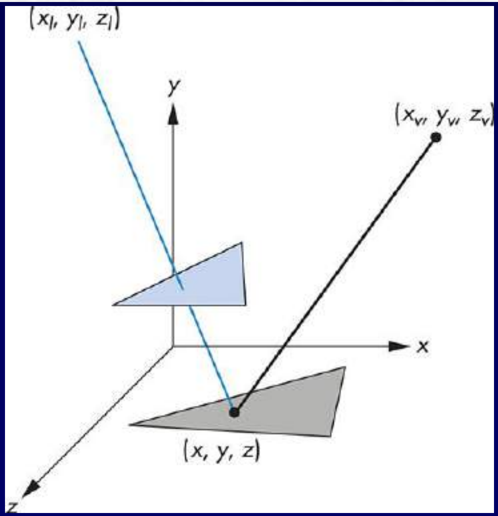


2-Pass Z-Buffer Method

Aka Shadow Map method

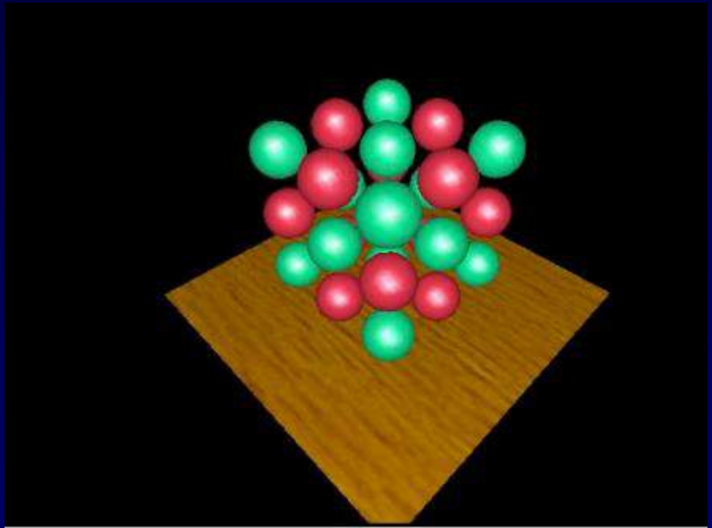
1. ***Do z-buffer (full) from light position***
    a. store results in shadow buffer
2. ***Do z-buffer (scanline or full) from eye position***
    a. For each (visible) pixel in scanline
        i. *Do inverse map point to WS*
        ii. *Map to SS of shadow buffer*
        iii. *Compare z with that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



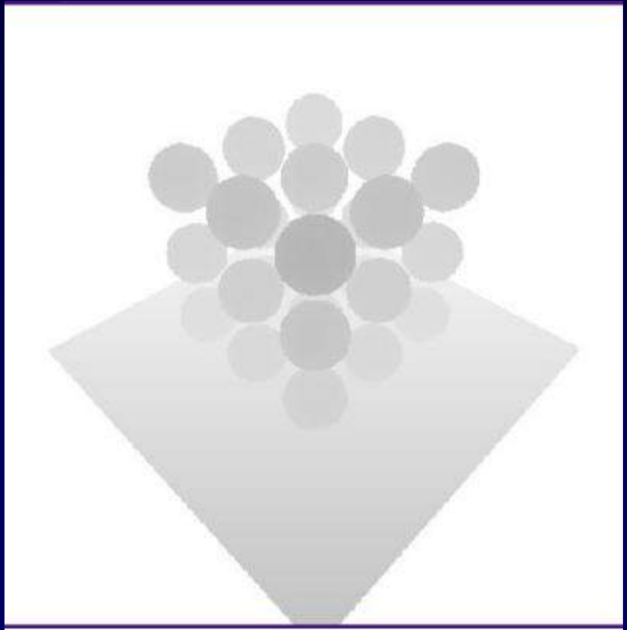


2-Pass Z-Buffer Method

First Pass: from light's position



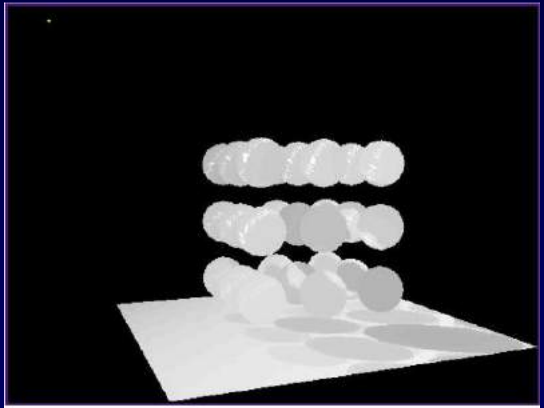
View from light



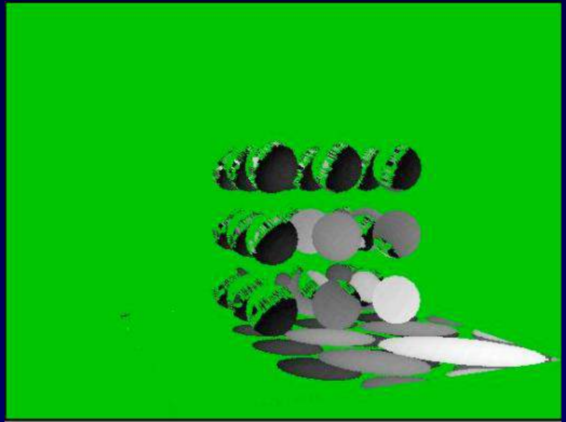
Depth Buffer (shadow map)

2-Pass Z-Buffer Method

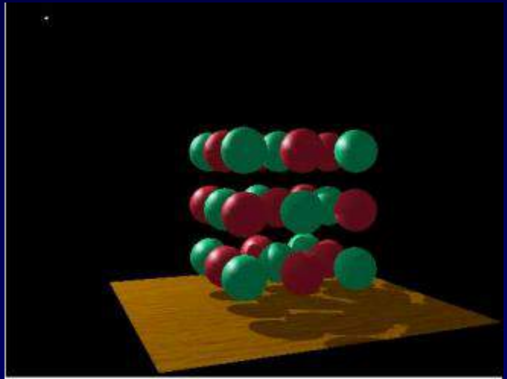
Second Pass: from eye's position



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