CS6533/CS4533 Lecture 13 Slides/Notes

Texture Mapping Variations (Environment, Bump, and Shadow Mapping); Ray Tracing & Radiosity (Notes, Ch 15, Notes, Ch 20)

By Prof. Yi-Jen Chiang
CSE Dept., Tandon School of Engineering
New York University

1

Mapping Techniques

1. Texture Mapping: Use an image as a texture to be "glued" (mapped) onto the surface of simple geometry to create sophisticated images.

   2 main ways to produce texture images:
   ① computer programs (eg. checkerboard)
   ② digital cameras (image files)

2. Environment Mapping: For a shiny object, the environment is reflected on its surface.
   ⇒ Render the environment into an image, treat it as a texture, then map the texture onto the obj. surface.

2

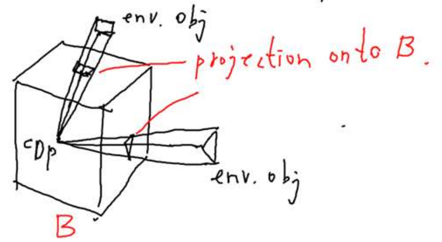
e.g. a shiny ball in the middle of a room.

2-Step Method: Let A be the shiny obj whose surface we want to perform environment mapping onto.

Step 1: Place the COP at the center of obj A, remove A.

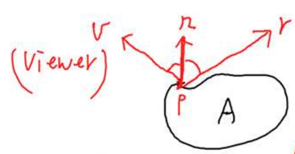
Project the environmental objs onto an intermediate surface B.

e.g. Typically we use "cube map"



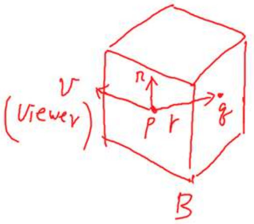
* Put the viewer (cop) at the center of A. remove A. Render the environment onto the 6 faces of cube B using each face as the view plane, for each face.

Step 2: Place obj A back. transfer the image on B onto A using texture mapping.



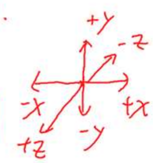
The point q of cube B where r hits, p should get the texture color of q.

Let $r = (r_x, r_y, r_z)$



① Decide which face of cube B the ray r hits.

[We will then use the texture on that face.]



To decide, look at the largest among $|r_x|, |r_y|, |r_z|$

say $|r_x|$. If $r_x > 0 \Rightarrow +x$ face] suppose it's $+x$ face.

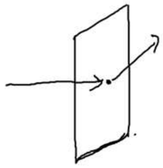
$r_x < 0 \Rightarrow -x$ face.

② Decide the texture coord. of q on the +x face

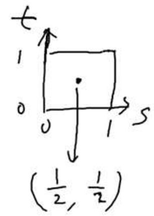
by $\frac{r}{|r_x|} = \left(+1, \frac{r_y}{|r_x|}, \frac{r_z}{|r_x|}\right) : \left(\frac{r_y}{|r_x|}, \frac{r_z}{|r_x|}\right) \longrightarrow (s, t)$

$\left(\frac{r_y}{|r_x|}, \frac{r_z}{|r_x|}\right) \in [-1,1]\times[-1,1]$

$r = (1, 0, 0)$



center of $+x$ face. texture coord. is $\left(\frac{1}{2}, \frac{1}{2}\right)$.



$(s,t) \in [0,1]\times[0,1]$.

$\left(\frac{r_y}{|r_x|}, \frac{r_z}{|r_x|}\right) \longrightarrow (s,t)$: origin $(0,0) \longrightarrow \left(\frac{1}{2}, \frac{1}{2}\right)$

$\underbrace{[-1, 1]}_{2} \longrightarrow \underbrace{(0, 1)}_{1}$ in both dimensions $(y \to s,\ z \to t)$. scale by factor of $\frac{1}{2}$.

$$(s,t) = \frac{1}{2}\left(\frac{r_y}{|r_x|}, \frac{r_z}{|r_x|}\right) + \left(\frac{1}{2}, \frac{1}{2}\right)$$

verify: $\underset{(0,\ 0)}{\left(\frac{r_y}{|r_x|}, \frac{r_z}{|r_y|}\right)} \longrightarrow \left(\frac{1}{2}, \frac{1}{2}\right)$ i.e. $(0,0) \to \left(\frac{1}{2}, \frac{1}{2}\right)$ ✓

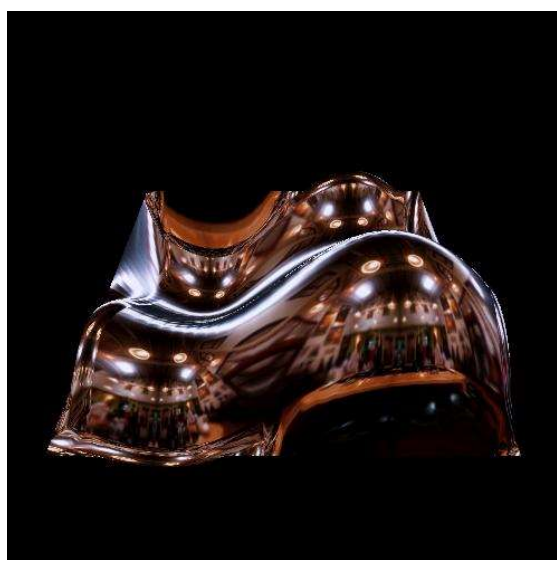
$\frac{1}{2}(-1,-1) + \left(\frac{1}{2}, \frac{1}{2}\right) = (0, 0)$ ie $(-1,-1) \to (0,0)$

$\frac{1}{2}(1,1) + \left(\frac{1}{2}, \frac{1}{2}\right) = (1,1)$ i.e. $(1,1) \to (1,1)$

ok.

5

Examples of Environment Mapping (1)



6

Examples of Environment Mapping (2)

7

3. Bump Mapping: Goal: create a bumpy surface (e.g. orange)

Naive attempt: Take a picture of the real bumpy obj (e.g. orange) then use texture mapping to map the picture onto the obj surface. (e.g. sphere) (smooth)

Issue: When we move the light or rotate the obj. we will notice that it is NOT realistic looking. (e.g. "shadows" corresponding to the bumps are incorrect)

Bump Mapping: We can vary the look of the shape of the surface by perturbing the normal vectors then using the perturbed normal vectors in shading computation.

8

Note: * The actual geometry is unchanged (still a flat polygon)

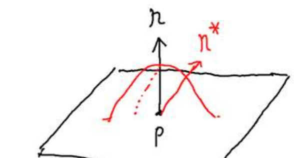
* Bump mapping does NOT change the $\left\{\begin{matrix}\text{boundary}\\ \text{silhouette}\end{matrix}\right\}$ of the obj

$p^* = p + d(p)\, n$ p: original surface pt.

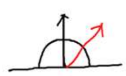
p^*: new surface pt displaced from p.

$d(p)$: displacement amount at p.

n^*: normal vector at p^*



tangent plane at p.



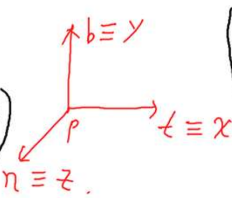
[There is a formal math derivation for n^*. We omit it here, and only put the conclusion below:]

$\Rightarrow$ n^* is n perturbed along a direction on the tangent plane, which can be decomposed as x & y components on the tangent plane.

9

$\Rightarrow$ Take t: tangent vector at p $\left(\begin{matrix}\text{Normalized:}\\ |n| = |t| = 1\end{matrix}\right)$

$b = n \times t$: binormal vector at p $\left(\begin{matrix}\text{Normalized:}\\ |b| = 1\end{matrix}\right)$



(* Tangent frame:
- origin is at p
- t, b, n are 3 basis vectors)

This is called the tangent frame (also called surface local frame.) where t, b, n are in the eye frame.

Create a 3×3 matrix $M = \begin{bmatrix} [\,t\,] \\ [\,b\,] \\ [\,n\,] \end{bmatrix}$ We can use M to transform vectors from eye frame to tangent frame.

[Verification: $M \cdot t = \begin{bmatrix} [\,t\,] \\ [\,b\,] \\ [\,n\,] \end{bmatrix} \begin{bmatrix} t \end{bmatrix}_{eye} = \begin{bmatrix} t\cdot t \\ b \cdot t \\ n \cdot t \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}_{tangent} \equiv x.$ $M \cdot b = \begin{bmatrix} [\,t\,] \\ [\,b\,] \\ [\,n\,] \end{bmatrix} \begin{bmatrix} b \end{bmatrix}_{eye} = \begin{bmatrix} t\cdot b \\ b \cdot b \\ n \cdot b \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}_{tangent} \equiv y.$

Similarly. $M \cdot n = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}_{tangent} \equiv z.$]

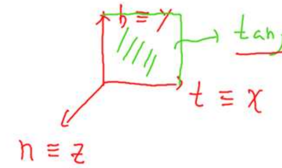
10

Main Idea: Use matrix M to transform everything from eye frame to tangent frame

then perform shading computation in tangent frame.

* In the tangent frame: original normal $n = z = (0, 0, 1)$

perturbed normal n^* is n perturbed along x direction & y; in the tangent frame.



tangent plane is the x-y plane in the tangent frame.

* original normal $n = (0, 0, 1)$

perturbed normal $n^* = (x, y, 1)$

with $\begin{cases} x = f(s) \\ y = f(t) \end{cases}$

i.e. x, y are results of a function f on the texture coord. (s, t) of the current fragment

$n^* \leftarrow$ normalize (n^*)

Use n^* for shading computation in tangent frame.

11

Gouraud (Per-Vertex) Shading

Phong (Per-Fragment) Shading

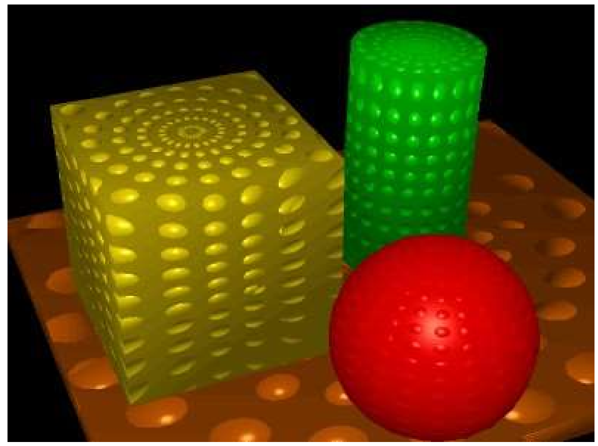
Bump Mapping (1)

Bump Mapping (2)

12

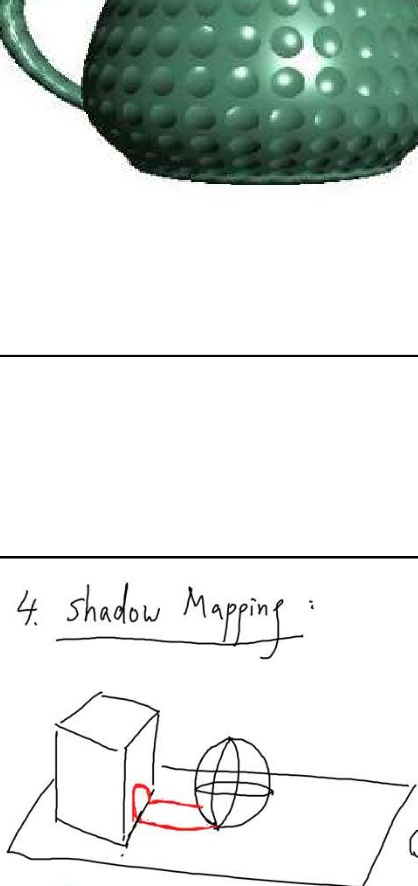
Bump Mapping (3)

Bump Mapping (4)



13

4. Shadow Mapping:



(shadow map created in ① is stored & read as a texture)

Main Issue: self shadowing

Sol: Add some tolerance ε (small and > 0): treat $|z_f - M_p| \le \varepsilon$ as (b)

(test case (b) first. and "else" is case (a))

① Put the viewer at the light position L. Render the scence to the z-buffer to create a "shadow map": The shadow map contains the z-values of all points that can be seen from L.

② Render the scene from the viewer. For each fragment f, also compute its "z-value" when viewed from L. (when rendered from L f is at pixel p with z-value z_f)

Look at the shadow map. (in shadow map, pixel p has value M_p)

(a) if $z_f > M_p$ then f can NOT be seen from L.

$\Rightarrow$ No contribution of light L to fragment f

(b) else ($z_f == M_p$) then f can be seen from L.

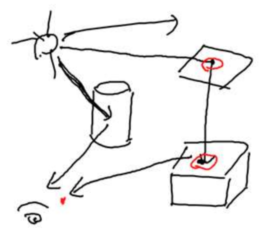
$\Rightarrow$ Add contribution of L to f

14

Global shading : 1. Ray Tracing

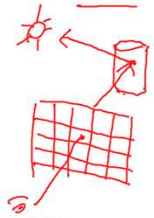
2. Radiosity

1. Ray Tracing : Simulate the interactions of rays with objs where the rays are bouncing among objs.



Issues : ① Infinite # rays to trace : too slow !!

Sol : We only care about the rays coming into the eye ⇒ trace rays backwards !!



Cast rays from cop thru each pixel center into the scene. then tracy the rays there

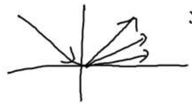
15

② How do we trace the bounced rays ?

(I) (i) For diffuse surface, reflected rays are along all directions.

⇒ can't trace so many rays !

(ii) For highly specular surface, reflected rays are along a narrow range of angles.



: ok to trace them.

Typically we assume to have a single reflected ray to trace

ok.

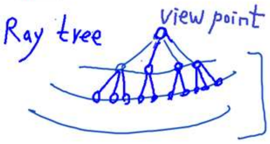
⇒ Ray tracing works mainly for highly specular surfaces

NOT for diffuse surfaces.

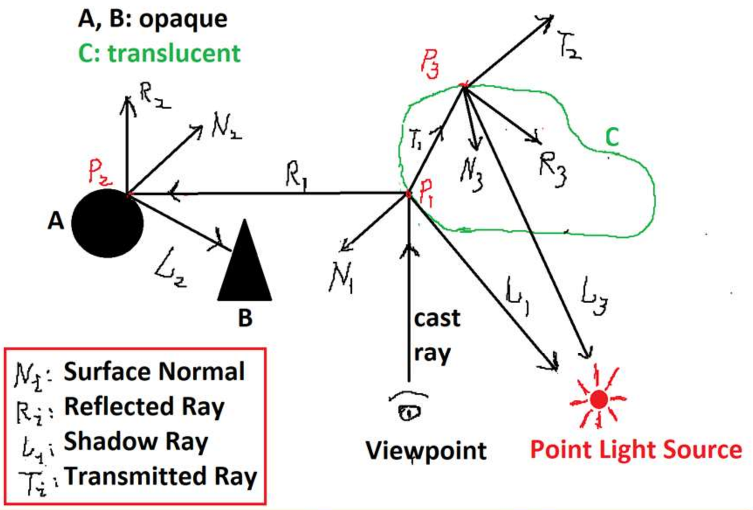
(II) We only trace rays with a pre-fixed # of bounces

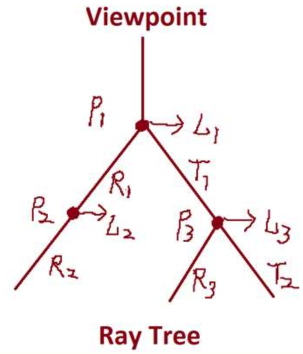
Fixed depth of the ray tree.

Ray tree



16





* **Shadow Ray:** a ray from a surface point p to a light source s.
If such ray intersects an opaque object, then p is in shadow from s.
E.g., L_2 is from P_2 toward the light source, but is blocked by B, so P_2 is in shadow from the light source.

The shade at P_1 is decided by
1. global ambient light
2. shading at P_1 from each light source (here: shadow ray L_1)
3. light along R_1 (decided by shade at P_2 recursively)
4. light along T_1 (decided by shade at P_3 recursively)

17

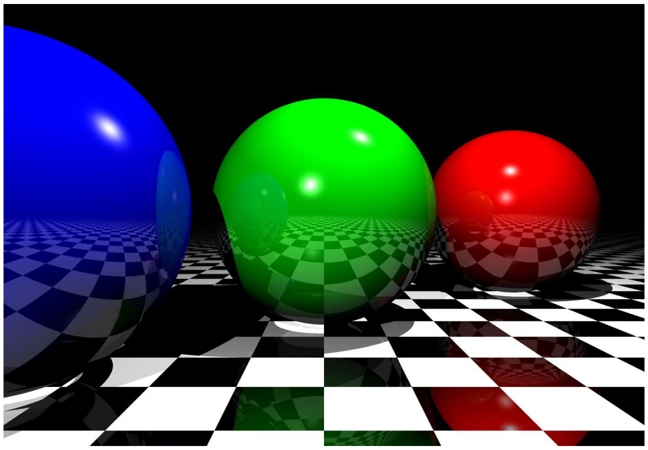
* Every time the viewer moves or changes the viewing direction, we need to re-do the ray-tracing computation
(expensive, NOT suitable for interactive applications).

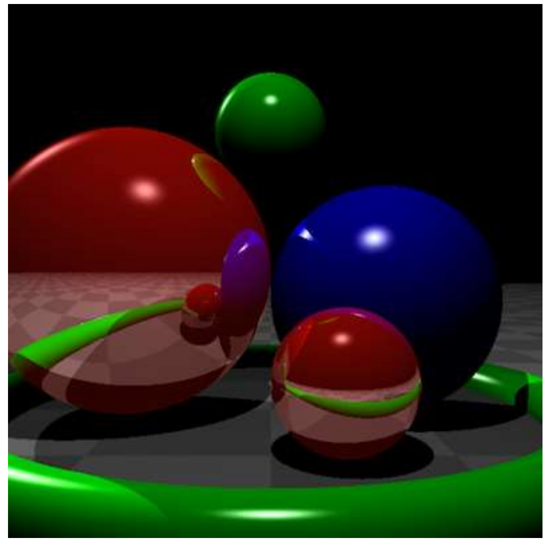
2. Radiosity: Works for diffuse surfaces
In closed environment, compute how the light energy is interacting between each pair of surface patches. (small flat polygons)

Assume: All surfaces are perfectly diffuse: reflected rays are along all directions
⇒ Computation is independent of the viewer position & viewing direction.
* Can be done in preprocessing (pre-computation)
* Suitable for interactive applications at run-time.
* Radiosity and ray tracing are complementary to each other.

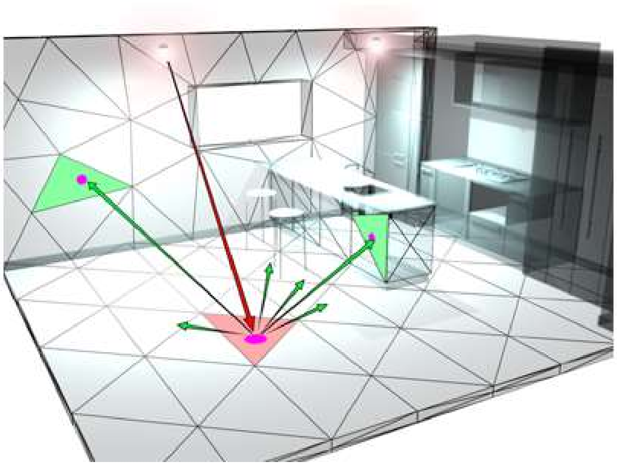
18

Example Results of Ray Tracing

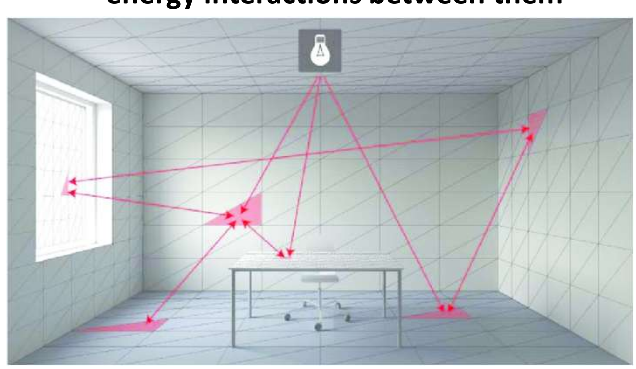




19

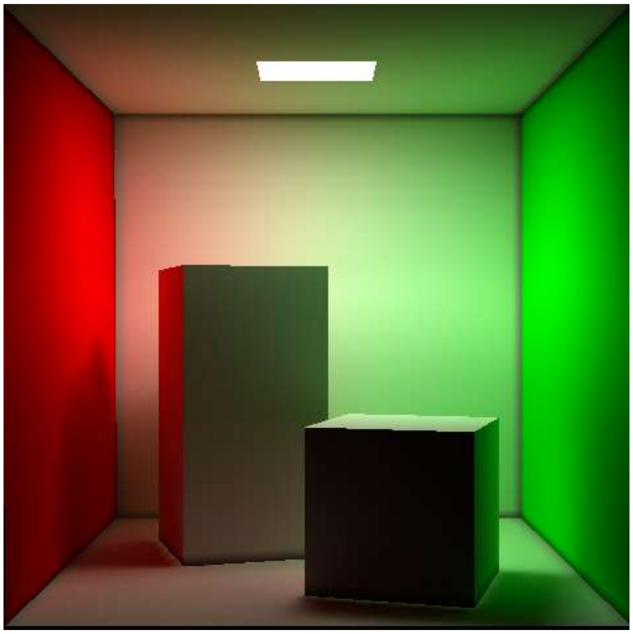


Radiosity Computation: For each pair of patches, compute the light energy interactions between them



Radiosity: Concepts of the Method

20



Radiosity Result:
The ``Color Bleeding'' Effect

21