Computer Graphics

Lecture 14

Bump-mapping, Global Illumination (1)

Today

- **Bump mapping**
- Displacement mapping
- Global Illumination
    - Radiosity

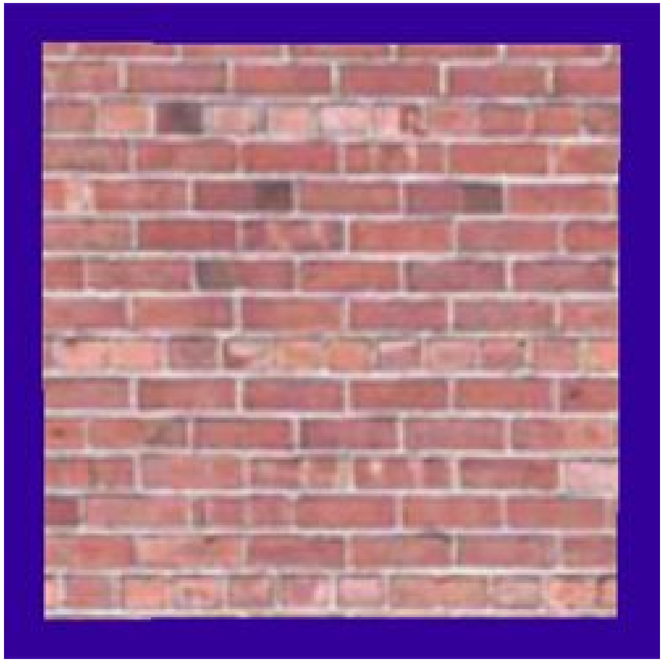
Bump Mapping

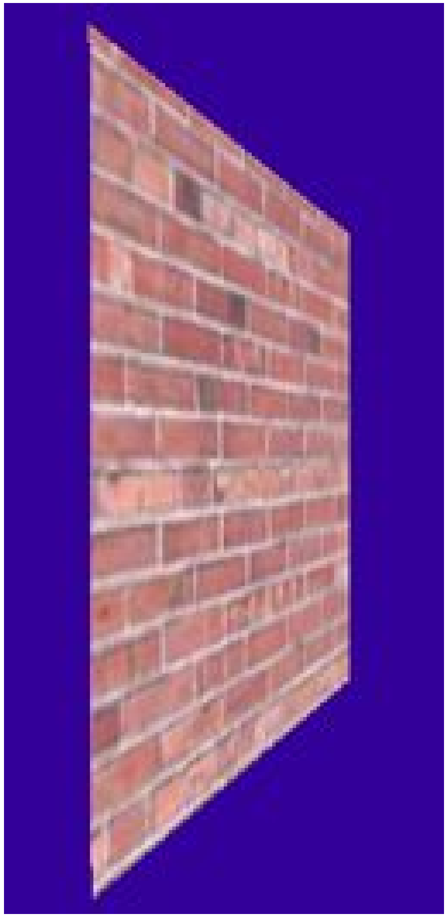
- A method to increase the realism of 3D objects without editing their geometry
- By adding high resolution bump maps
- In a way similar to texture mapping

What's Missing?

What's the difference between a real brick wall and a photograph of the wall texture-mapped onto a plane?

No shadows between the bricks





This is what we want

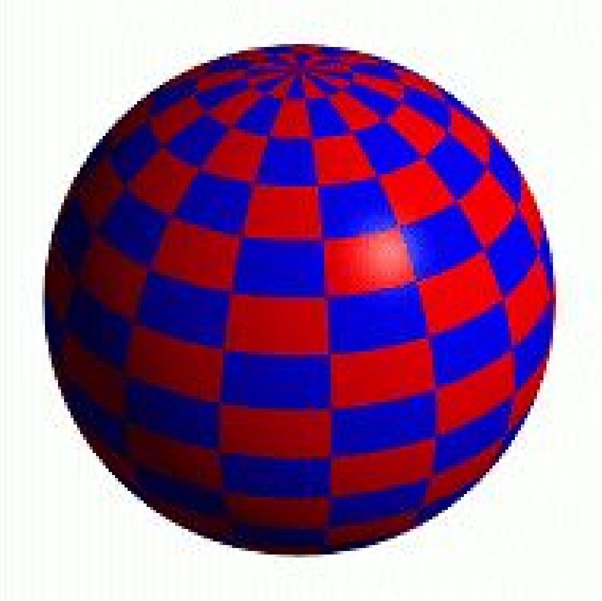
How do we achieve this?

High resolution brick wall?

- Difficult to prepare the model
- Very costly to handle during run-time

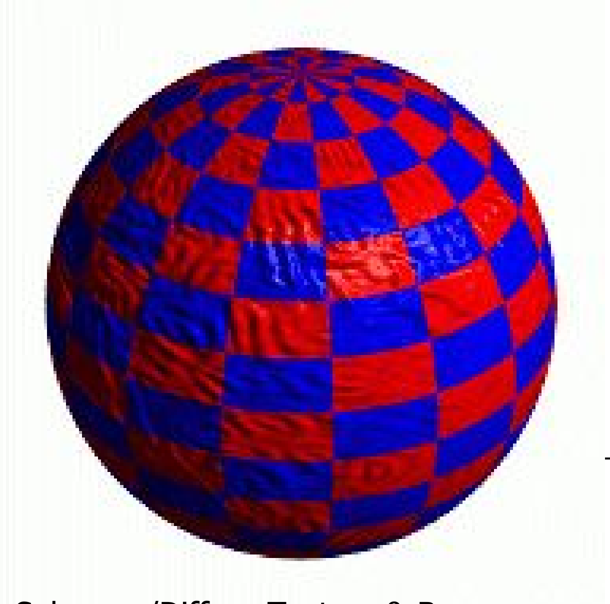
Bump Mapping

- Use textures to alter the surface normal but not the geometry
- Done during the rasterization stage
  - Can synthesize realistic images without using high resolution meshes



Sphere w/Diffuse Texture

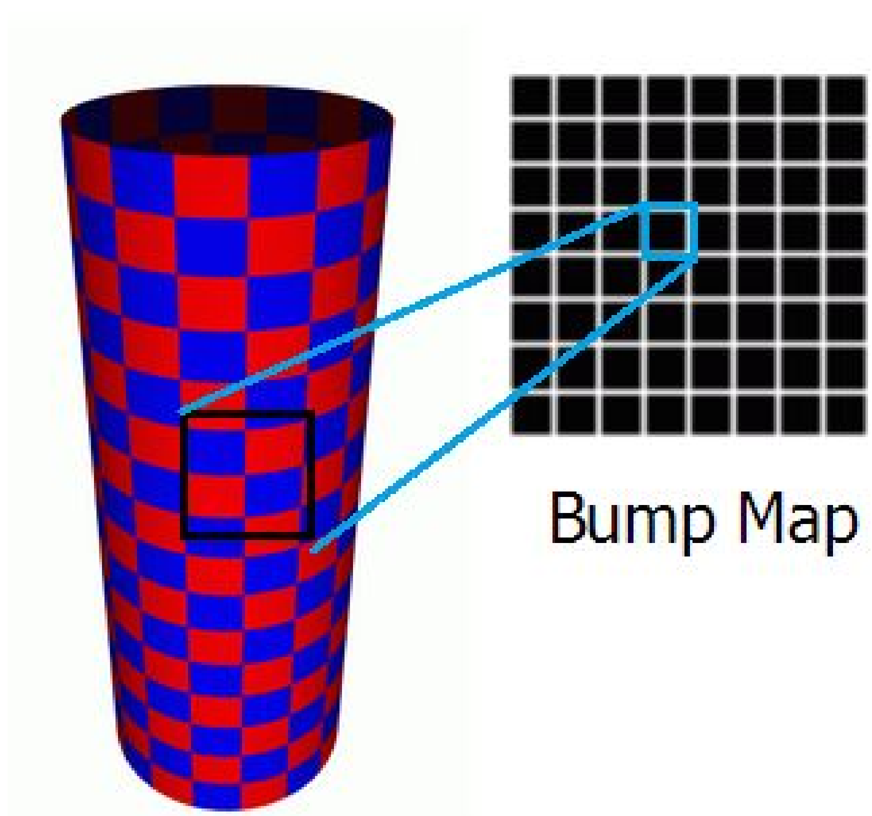
Swirly Bump Map

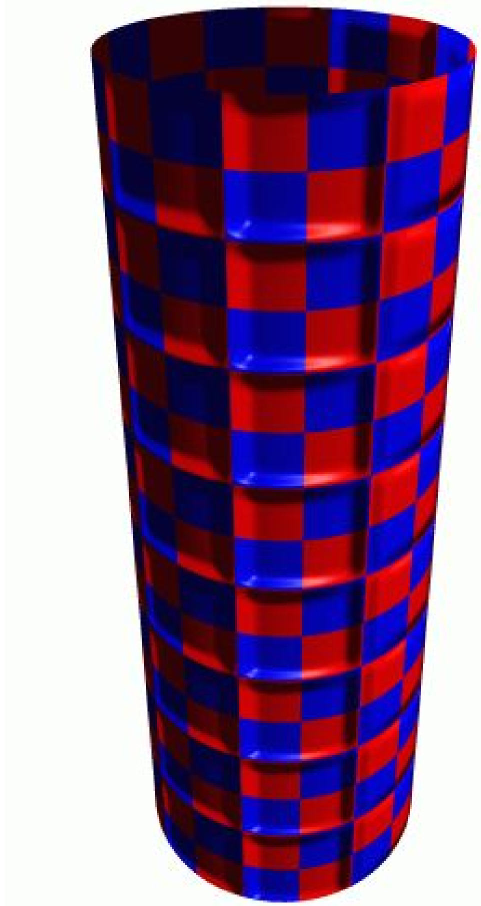


Sphere w/Diffuse Texture & Bump Map

Preparing Data

Prepare a bump map whose values change from 0 to 1
Prepare a mapping of vertices to uv coordinates

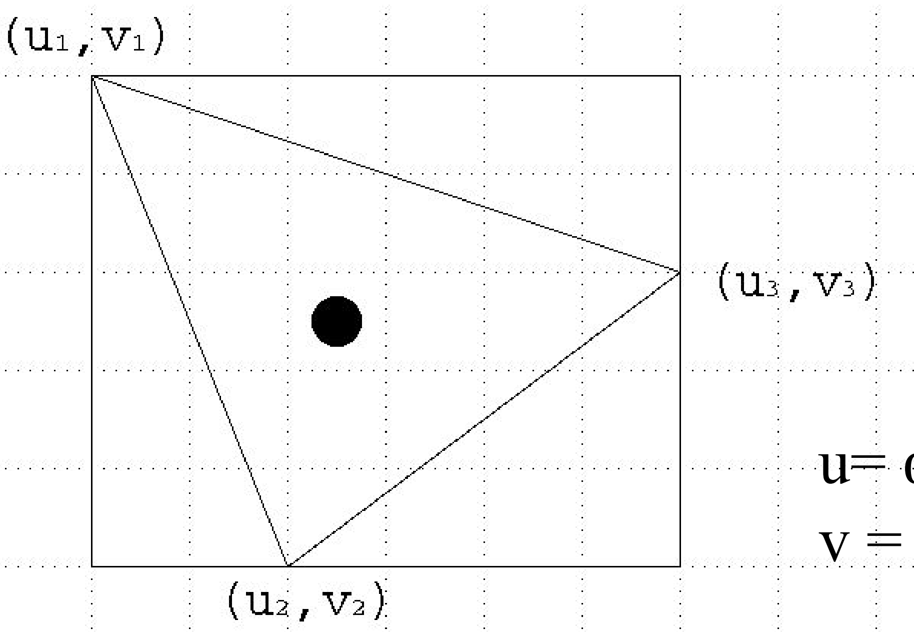


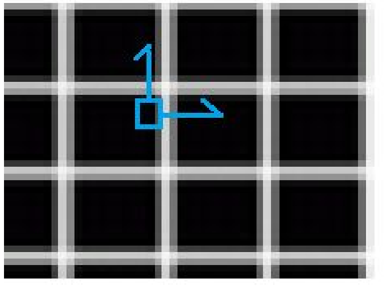


Cylinder w/Texture Map & Bump Map

Procedure

- During rasterization, compute the barycentric coordinates of the pixel and compute its uv coordinates by interpolation
- Lookup the bump map using the uv coordinates
- Compute the finite difference of the bump map (Fu, Fv) = (dF/du, dF/dv)



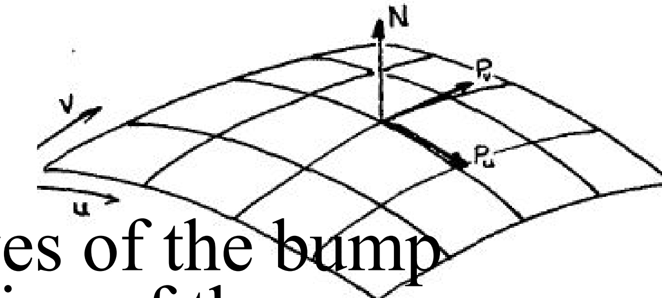


$$u = \alpha\, u_1 + \beta\, u_2 + \gamma\, u_3$$
$$v = \alpha\, v_1 + \beta\, v_2 + \gamma\, v_3$$

Procedure - continued

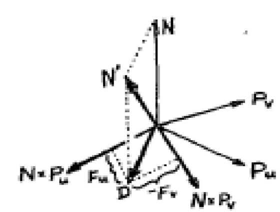
- When rasterizing a pixel in the triangle, compute the original normal vector **n** and the its uv coordinates by barycentric coordinates
- Perturb the normal vector by

$$\mathbf{n}' = \mathbf{n} + \frac{F_u(\mathbf{n} \times \mathbf{P_v}) - F_v(\mathbf{n} \times \mathbf{P_u})}{||\mathbf{n}||}$$



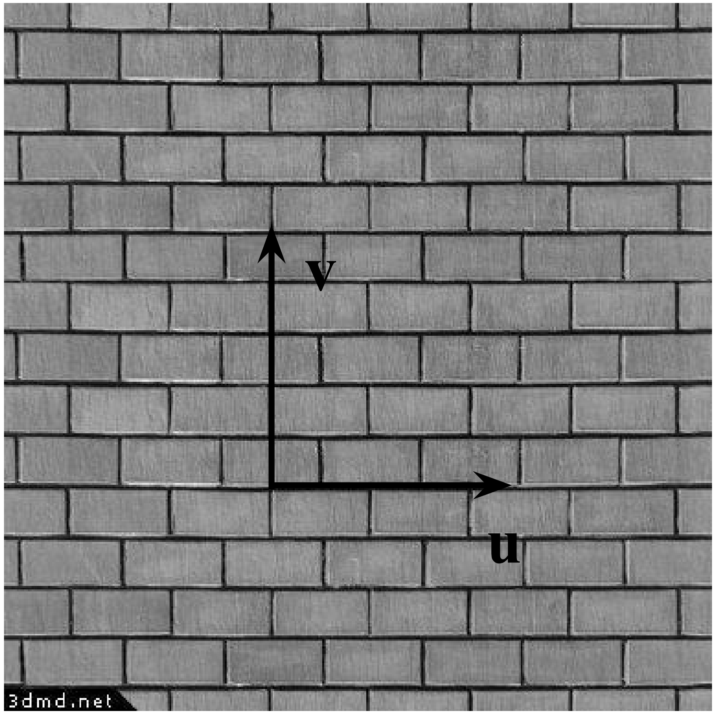
where (Fu, Fv) are the partial derivatives of the bump map, and **Pu, Pv** are the partial derivative of the geometry with respect to the uv

- Do the lighting computation using **n’**



Computing Fu, Fv

- Simply compute the finite difference of the height map at the corresponding uv



Computing Pu, Pv

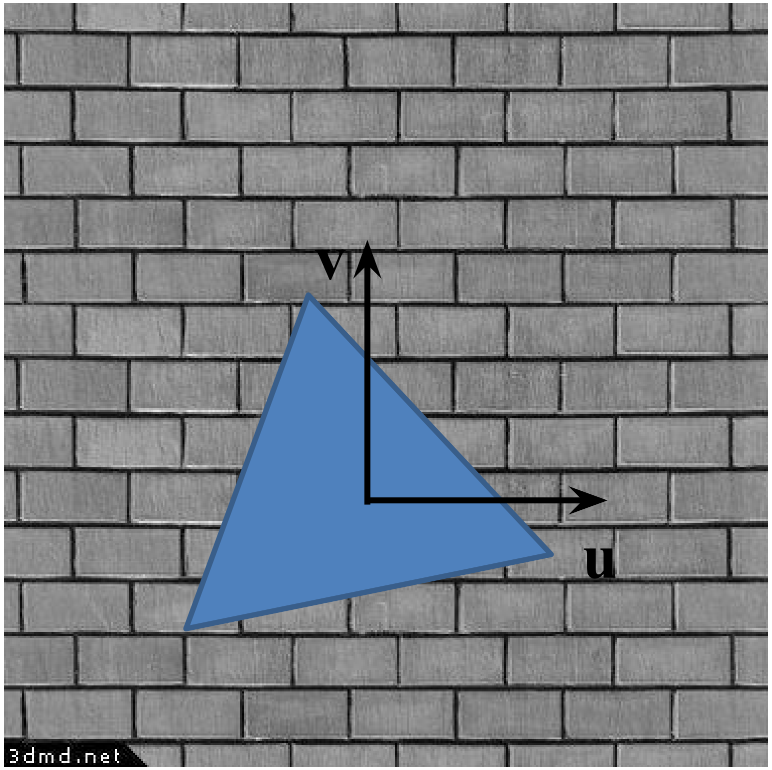
How P changes according to u,v in the texture space

$$P = (Px, P_y, P_z)$$

$$Pu = (\frac{\partial P_x}{\partial u}, \frac{\partial P_y}{\partial u}, \frac{\partial P_z}{\partial u}),$$

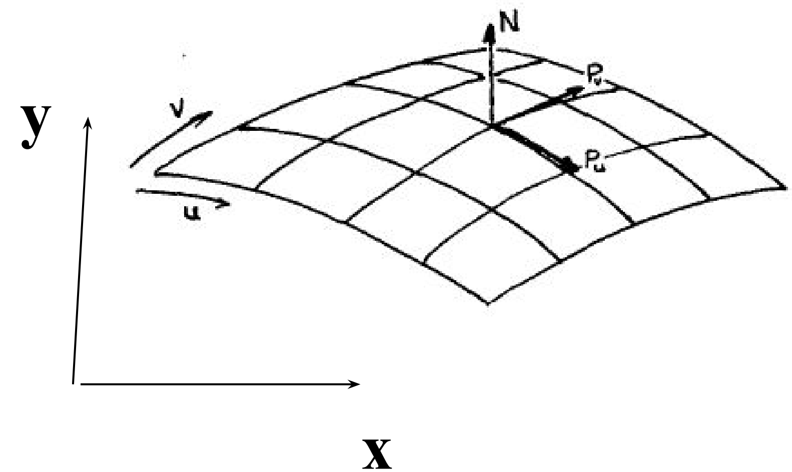
$$Pv = \left(\frac{\partial P_x}{\partial v}, \frac{\partial P_y}{\partial v}, \frac{\partial P_z}{\partial v}\right)$$

1. Map the 3D coordinates of the triangle (**P**s) back into the uv space and compute its finite difference (Pu, Pv are same within the triangle)
2. Use the chain rule



Or compute **Pu, Pv** in screen space using chain rule

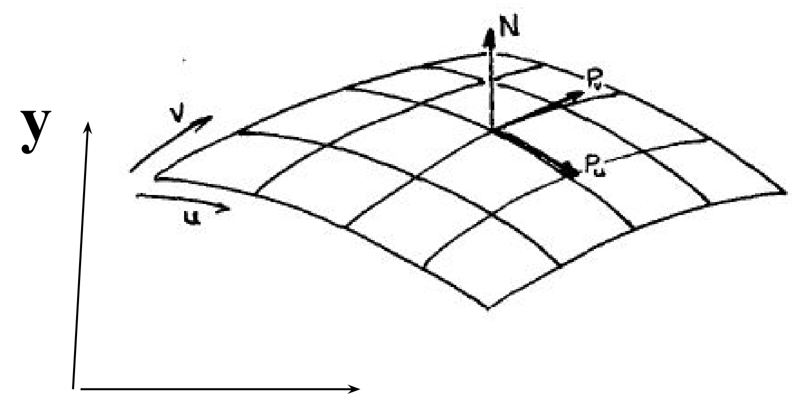
$$\frac{\partial P}{\partial x} = \frac{\partial P}{\partial u}\frac{\partial u}{\partial x} + \frac{\partial P}{\partial v}\frac{\partial v}{\partial x}$$
$$\frac{\partial P}{\partial y} = \frac{\partial P}{\partial u}\frac{\partial u}{\partial y} + \frac{\partial P}{\partial v}\frac{\partial v}{\partial y}.$$



Or compute **Pu, Pv** in screen space using chain rule

$$\frac{\partial P}{\partial x} = \frac{\partial P}{\partial u}\frac{\partial u}{\partial x} + \frac{\partial P}{\partial v}\frac{\partial v}{\partial x}$$
$$\frac{\partial P}{\partial y} = \frac{\partial P}{\partial u}\frac{\partial u}{\partial y} + \frac{\partial P}{\partial v}\frac{\partial v}{\partial y}.$$

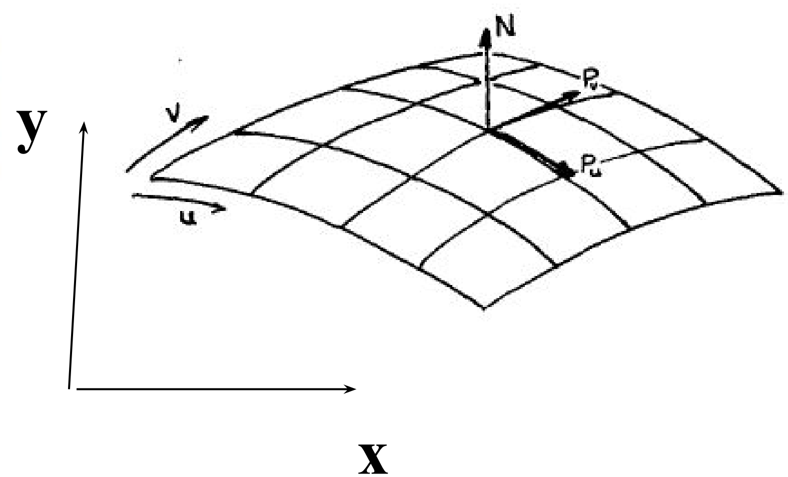
These can be computed in the screen space by finite difference



$$\frac{\partial u}{\partial x} = \frac{u_{i+1,j} - u_{i-1,j}}{2d}$$
$$\frac{\partial u}{\partial y} = \frac{u_{i,j+1} - u_{i,j-1}}{2d}$$
$$\frac{\partial v}{\partial x} = \frac{v_{i+1,j} - v_{i-1,j}}{2d}$$
$$\frac{\partial v}{\partial y} = \frac{v_{i,j+1} - v_{i,j-1}}{2d}.$$

Or compute **Pu, Pv** in screen space using chain rule

$$\begin{pmatrix} \frac{\partial P}{\partial u} \\ \frac{\partial P}{\partial v} \end{pmatrix} = \frac{1}{M} \begin{pmatrix} \frac{\partial v}{\partial y} & -\frac{\partial v}{\partial x} \\ -\frac{\partial u}{\partial y} & \frac{\partial u}{\partial x} \end{pmatrix} \begin{pmatrix} \frac{\partial P}{\partial x} \\ \frac{\partial P}{\partial y} \end{pmatrix}$$

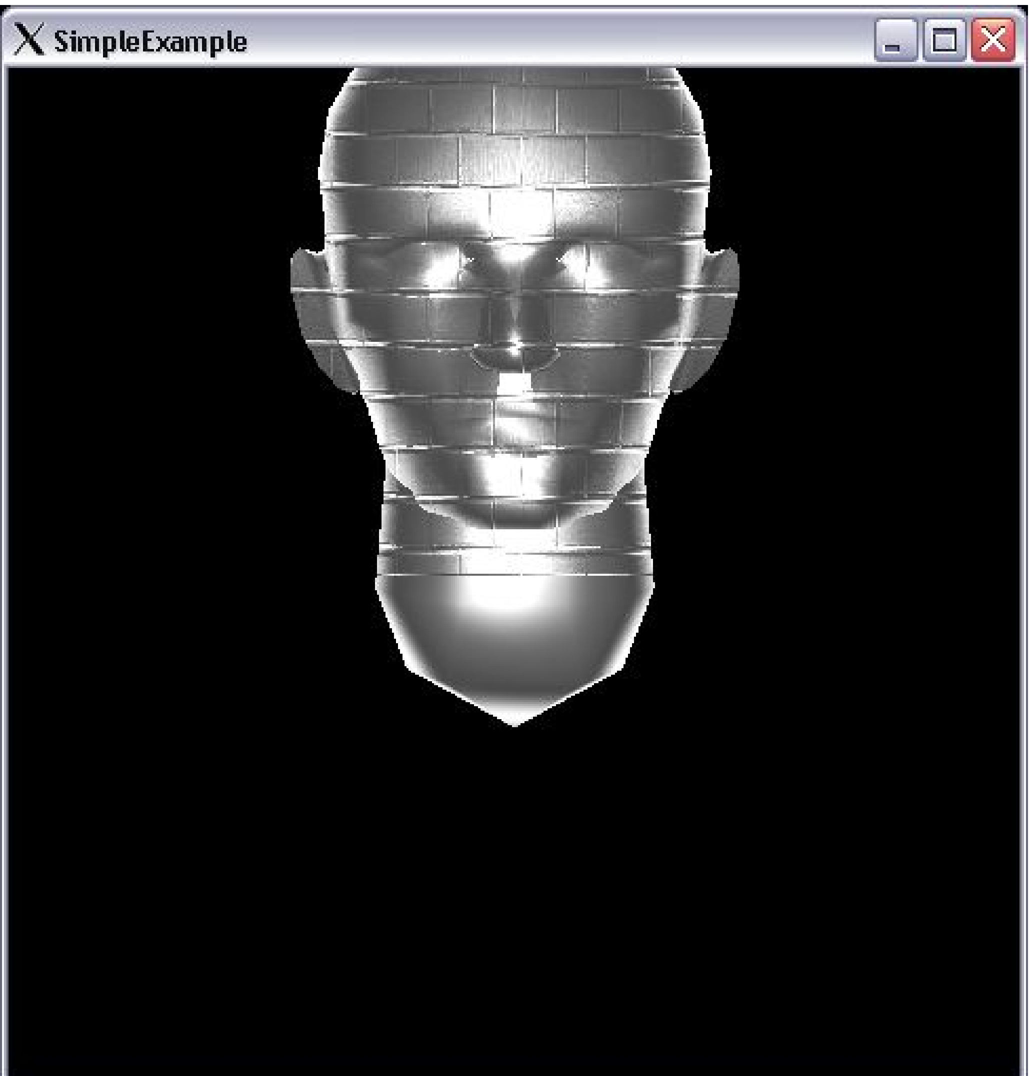


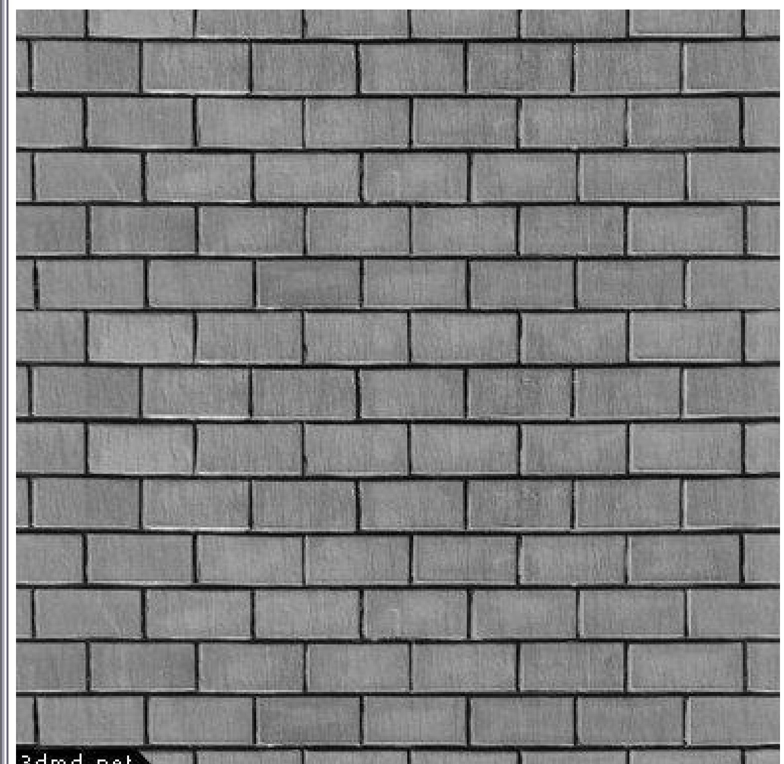
where $M = \frac{\partial v}{\partial y}\frac{\partial u}{\partial x} - \frac{\partial v}{\partial x}\frac{\partial u}{\partial y}$

Preparing Bump Maps

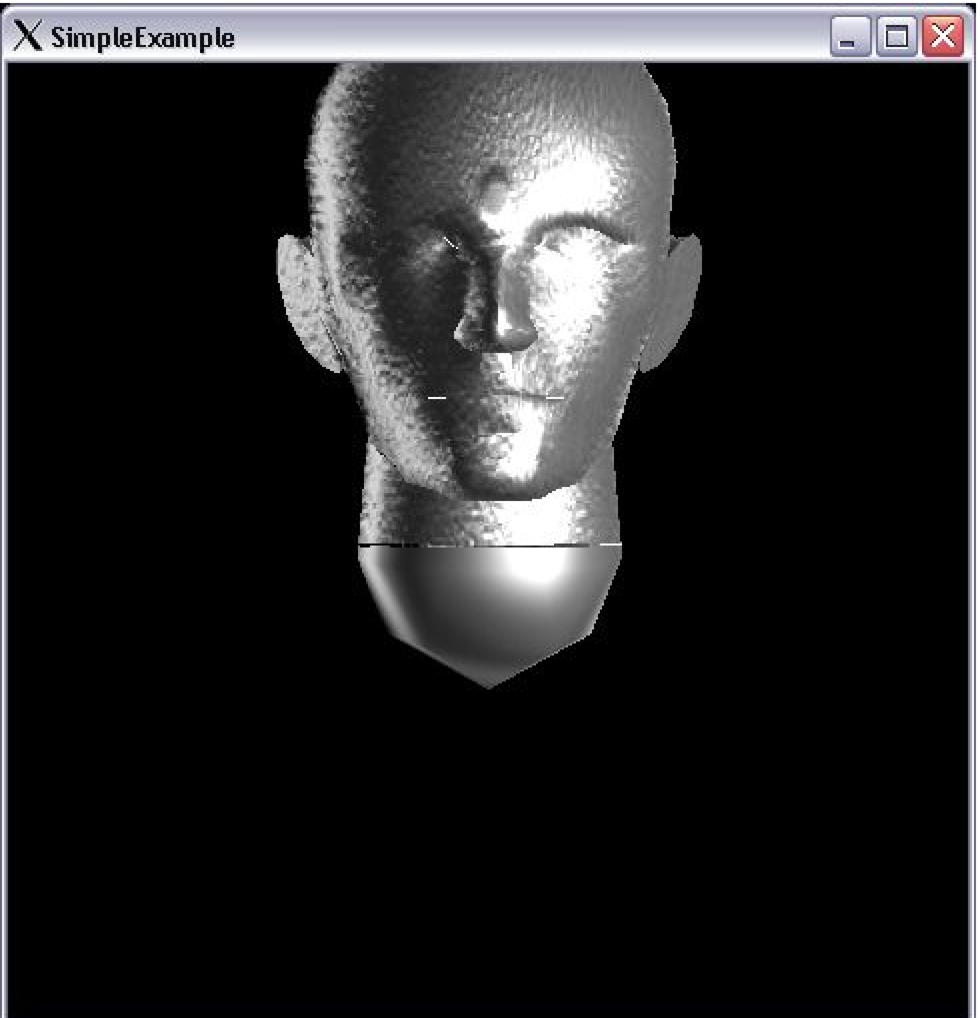
- Can be produced from photos or designed by the user
- If it is from an image, convert the image to greyscale
- Adjust the contrast, such the min is 0 and max is 1
- White part bumps out and black color bump inwards
    - Need to make sure there is no specular highlights in the image

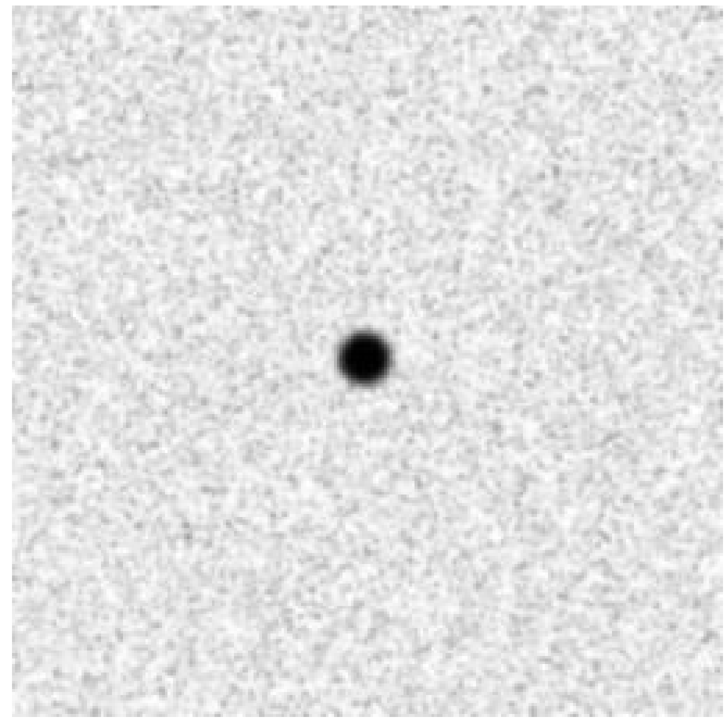
Some more examples



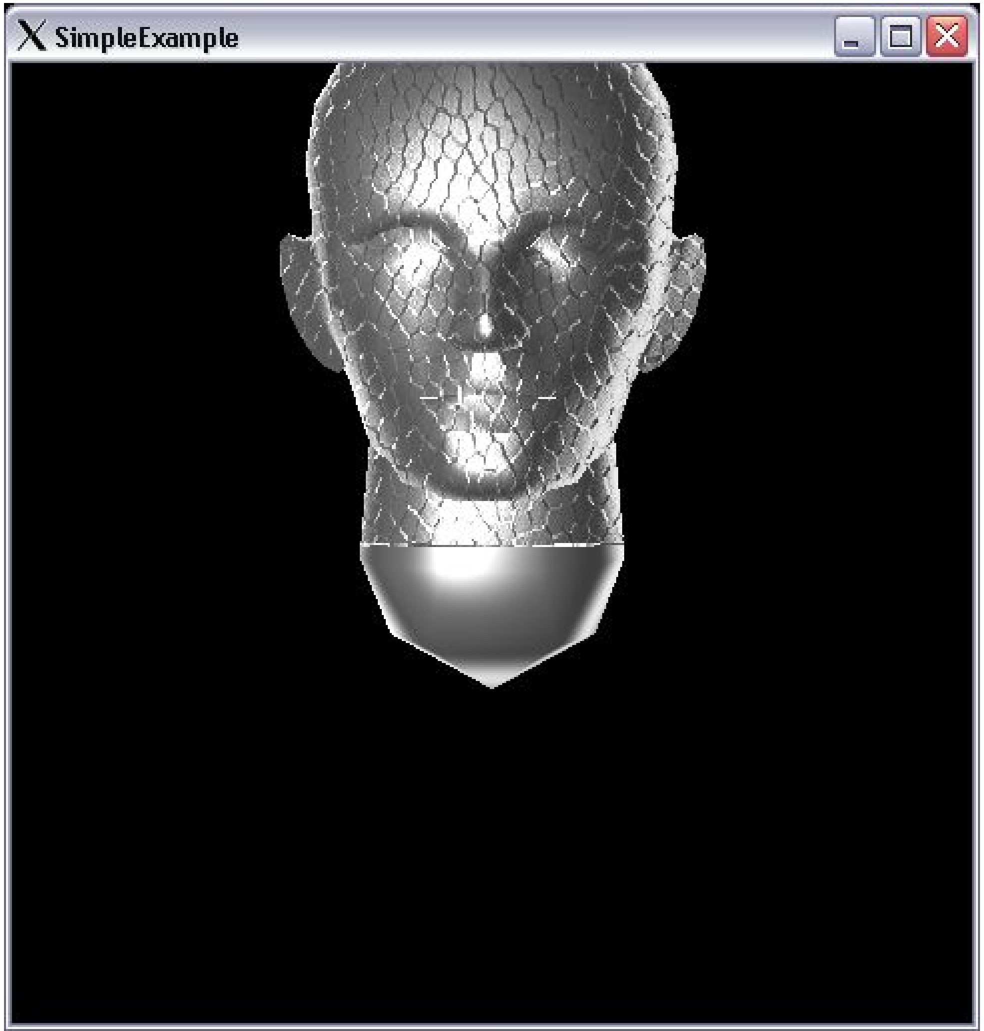


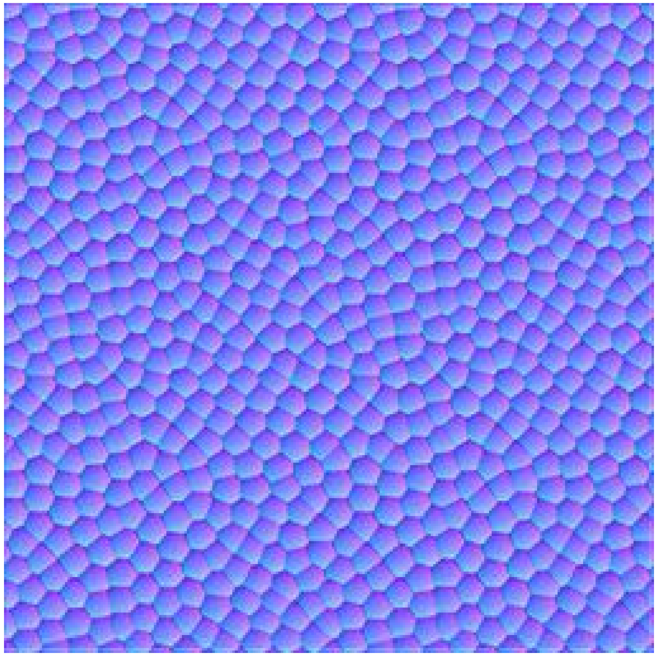
Some more examples





Some more examples



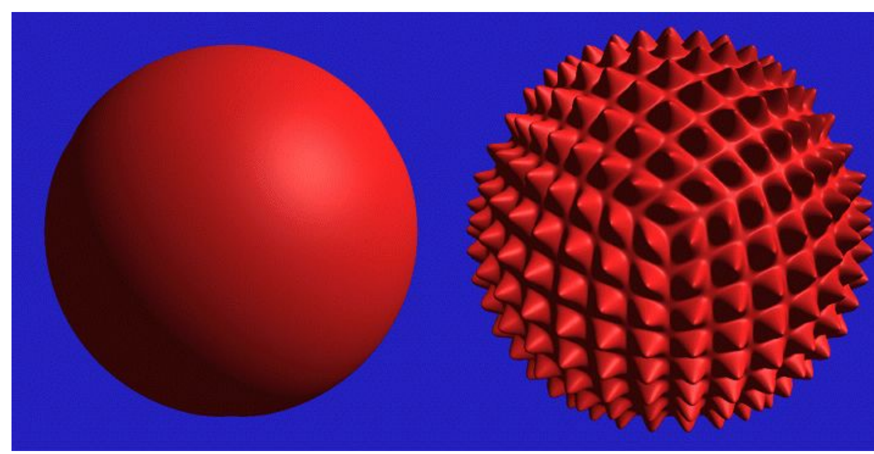


What's Missing?

There are no actual bumps on the silhouette of a bump-mapped object

Displacement Mapping

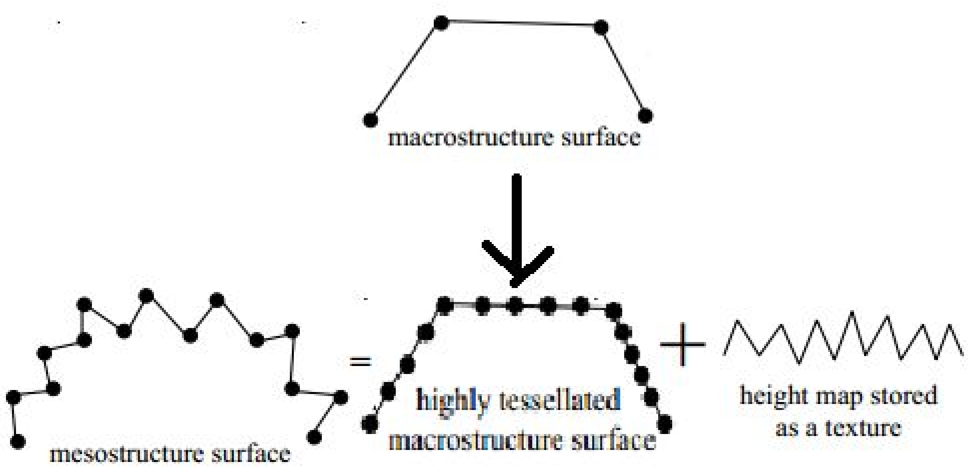
- Use the texture to actually move the surface point
- The texture is a 3D mesh defined in the uv space

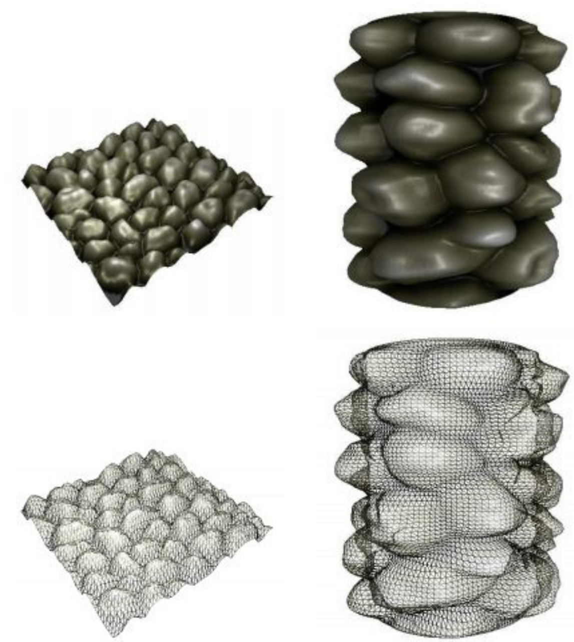


How it works

1. Tessellate (subdivide) the base polygon mesh such that its resolution is same as the displacement map

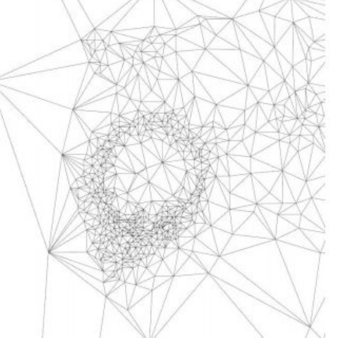
2. Move each vertex in the normal direction of the base polygon mesh as much as the height in the displacement map





Displacement Mapping: Discussions

- The cost increases as the polygon numbers is high
- Done in the geometry stage and not in the rasterization stage
- Can adaptively tessellate the base polygon mesh for the region the spatial frequency is high

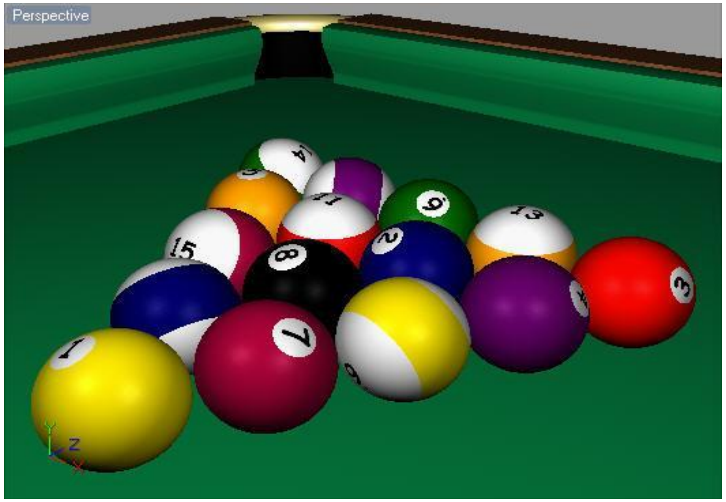


Today

- Bump mapping
- Displacement mapping
- **Global Illumination**
  - **Radiosity**

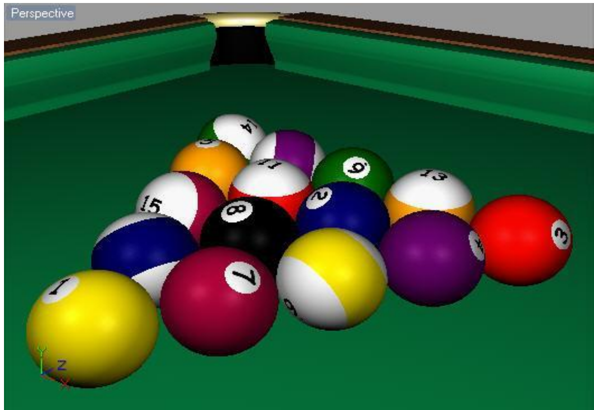
Background

- Rendering methods can be classified into
  - Local Illumination techniques
  - Global Illumination techniques



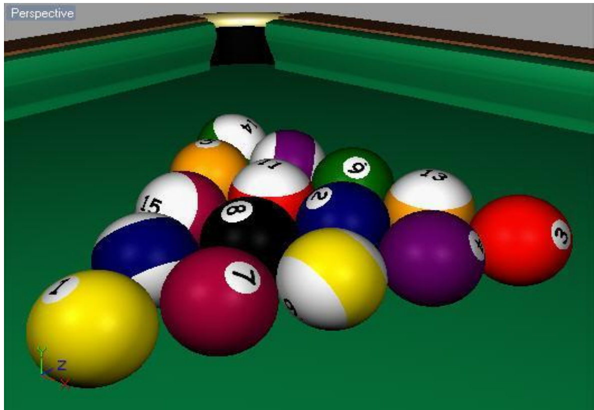
Local Illumination methods

- Considers light sources and surface properties only.
  - Phong Illumination, Phong shading, Gouraud Shading
- Very fast
  - Used for real-time applications such as 3D computer games



Local Illumination : problems

- Images synthesized appear artificial
  - No inter-surface reflections
  - Ambient light is a very simplified model
- Requires the user to add the effects like shadows and mirrors one by one
  - Shadow maps, shadow volume, shadow texture for producing shadows
  - Mirroring by reflecting the world or environment mapping



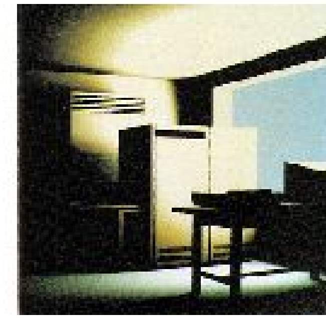
Global Illumination

- In the real-world, light comes from all directions (ambient light)
- This is due to inter-reflections
- Global illumination methods handle such effects

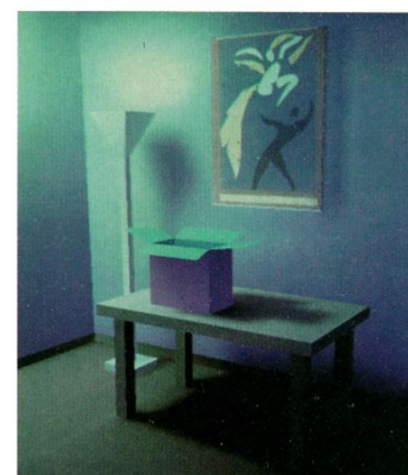
Global Illumination

- Methods
  - Radiosity
  - Monte-Carlo ray tracing, Photon Mapping
- Requires more computation and is slow

The Radiosity Method (85'-)



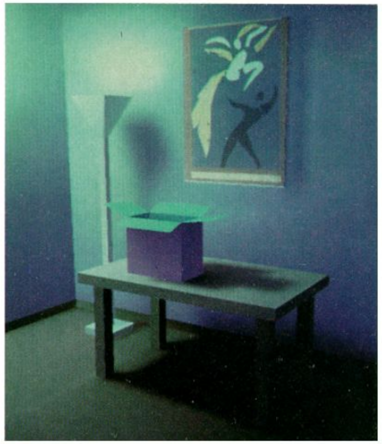
- Based on a method developed by researchers in heat transfer in 1950s
- Applied to computer graphics in the mid 1980s by
  - Michael Cohen
  - Tomoyuki Nishita

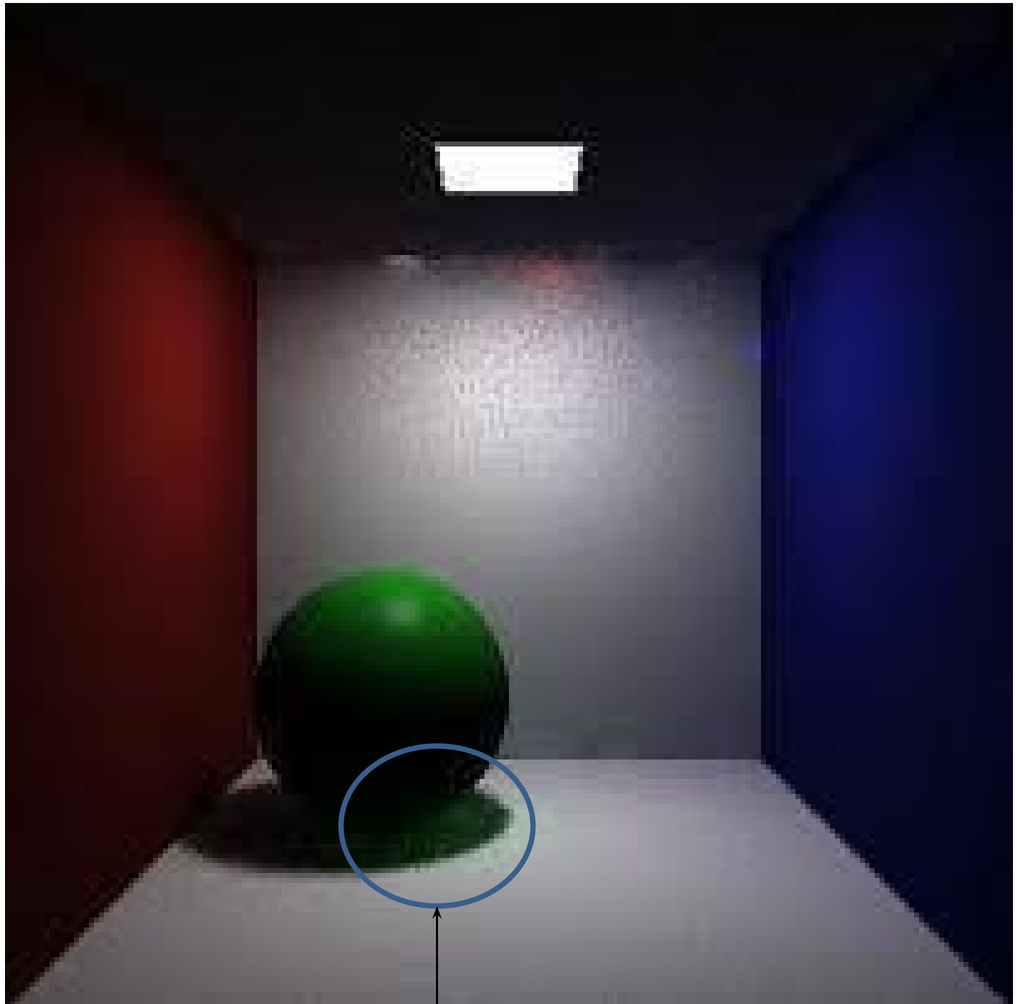


The Radiosity Method (85'-)

- Can simulate inter-surface reflection
- Can produce nice ambient effects
- Can simulate effects such as
  - Soft shadows,
  - color bleeding

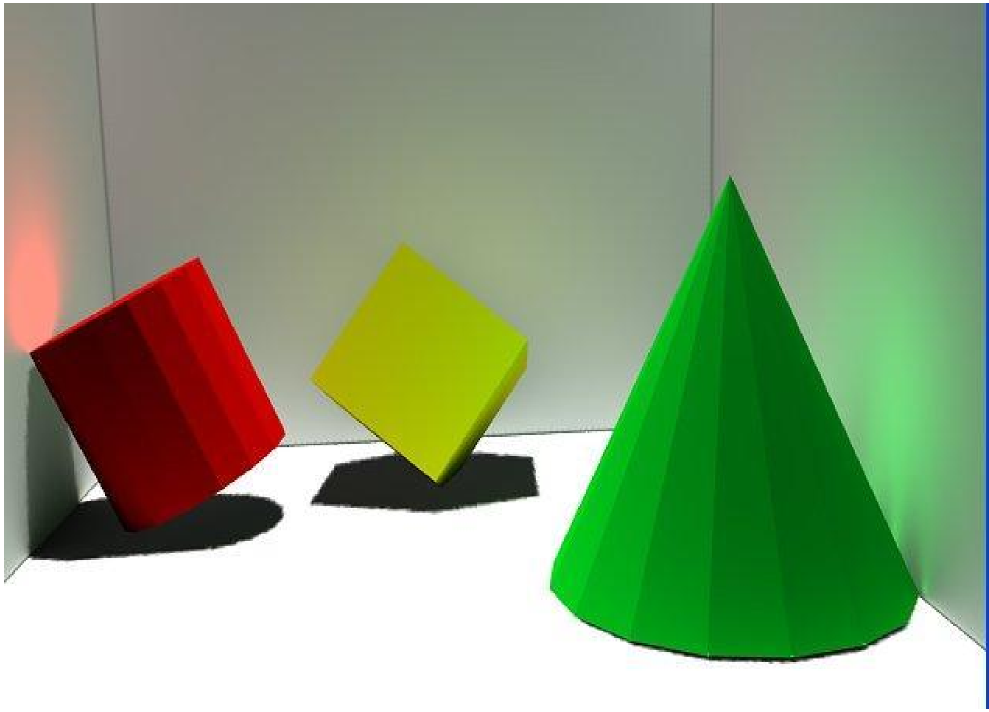
Can only handle diffuse color
→ need to be combined with ray-tracing to handle specular light





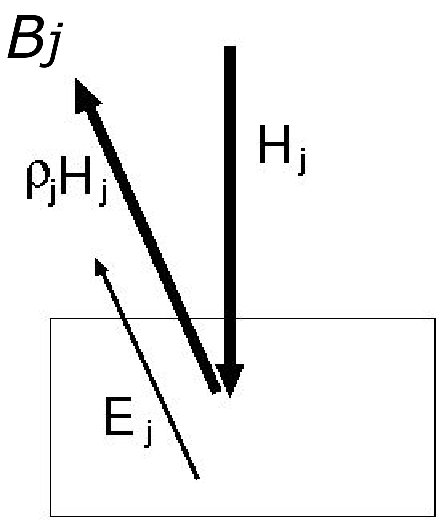
Color bleeding

Color Bleeding



The Radiosity Model

- At each surface in a model the amount of energy that is given off (Radiosity) is comprised of
  - the energy that the surface emits internally (E), plus
  - the amount of energy that is reflected off the surface (ρH)



$$\boxed{B_j = \rho_j H_j + E_j} \qquad \text{Eq. 1}$$

B_j is the radiosity of surface j,

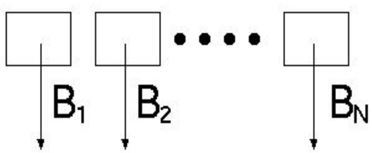
ρ_j is the reflectivity of surface j,

E_j is the energy emitted by surface j.

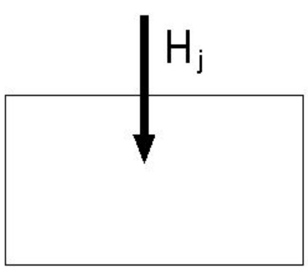
H_j is the energy incident on surface j

The Radiosity Model (2)

- The amount of incident light hitting the surface can be found by summing for all other surfaces the amount of energy that they contribute to this surface



$$\boxed{H_j = \sum_{i=1}^{N} B_i F_{i,j}} \quad \text{Eq. 2}$$



$F_{i,j}$ Form factor

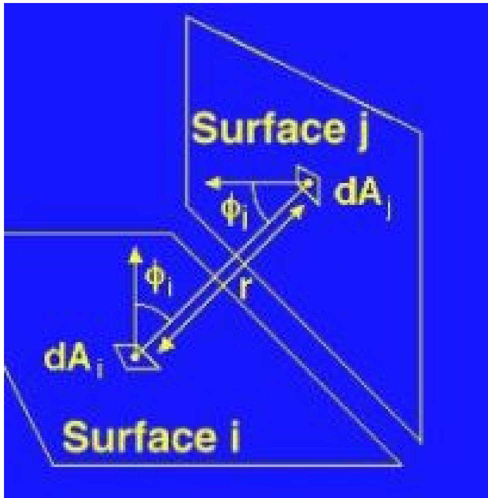
Form Factor (F_{ij})

- The fraction of energy that leaves surface i and lands on surface j
- Between differential areas, it is

$$\frac{\cos\phi_i \cos\phi_j}{\pi |r|^2} dA_i dA_j$$

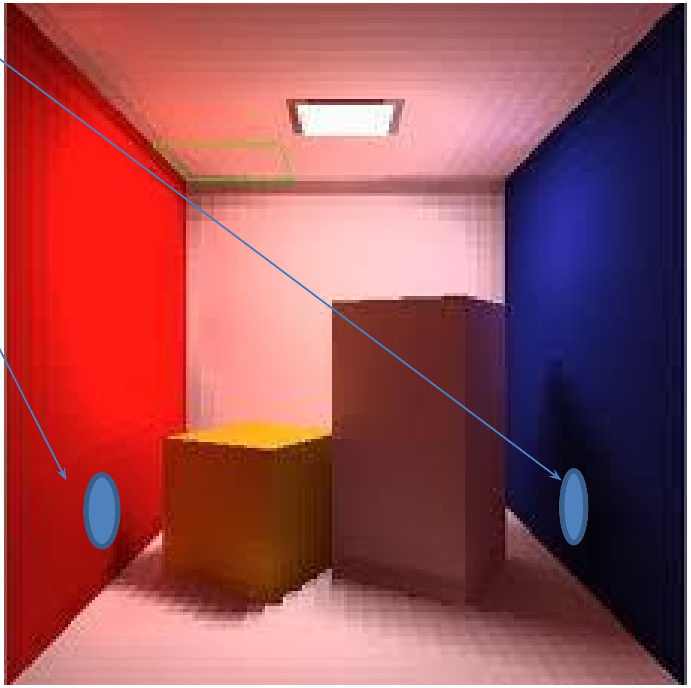
- The overall form factor between i and j is

$$F_{ij} = \sum_i \sum_j \frac{\cos\phi_i \cos\phi_j}{\pi |r|^2} dA_i dA_j$$



Form Factor (2)

- Also need to take into account occlusions
- The form factor for those faces which are hidden from each other must be zero



The Radiosity Matrix

The radic :

$$B_j = E_j + \rho_j \sum_{i=1}^{N} B_i F_{i,j}$$

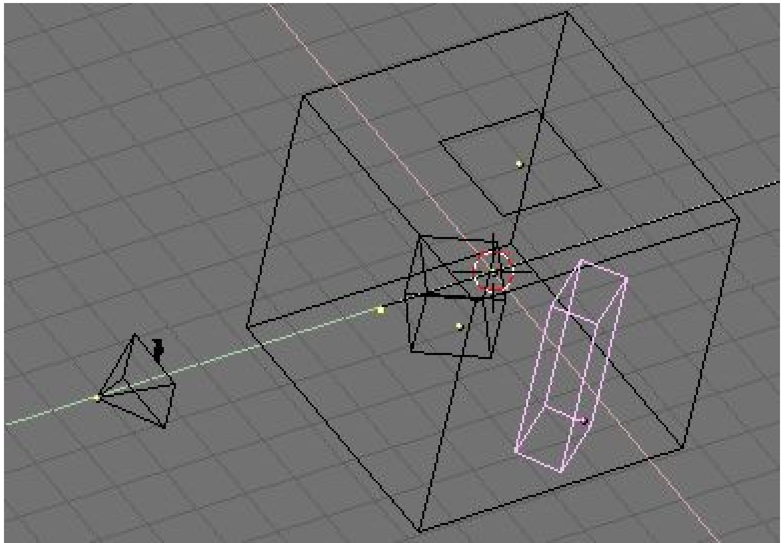
The derived radiosity equations form a set of N linear equations in N unknowns. This leads nicely to a matrix solution:

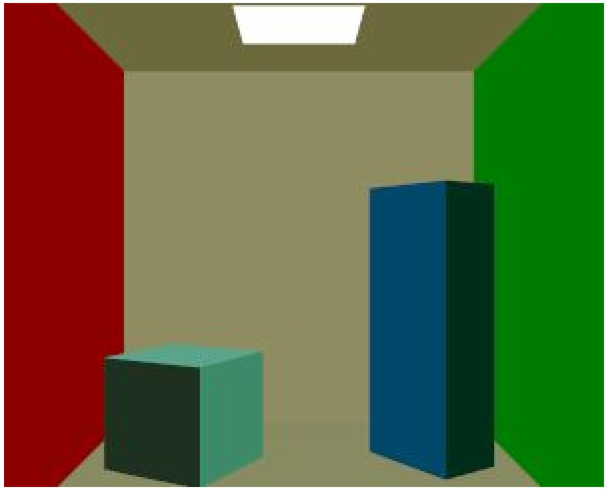
$$\begin{pmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \dots & -\rho_1 F_{1N} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \dots & -\rho_2 F_{2N} \\ \vdots & \vdots & \dots & \vdots \\ -\rho_N F_{N1} & -\rho_N F_{N2} & \dots & 1-\rho_N F_{NN} \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_N \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_N \end{pmatrix}$$

Solve for Bi – Use methods like Gauss-Seidal

Radiosity Steps:

1. Generate Model (set up the scene)
2. Compute Form Factors and set the Radiosity Matrix
3. Solve the linear system
4. Render the scene





Radiosity Steps:

1. Generate Model (set up the scene)
2. Compute Form Factors and set the Radiosity Matrix
3. Solve the linear system
4. Render the scene

$$\boxed{\begin{pmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \dots & -\rho_1 F_{1N} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \dots & -\rho_2 F_{2N} \\ \vdots & \vdots & \dots & \vdots \\ -\rho_N F_{N1} & -\rho_N F_{N2} & \dots & 1-\rho_N F_{NN} \end{pmatrix}} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_N \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_N \end{pmatrix}$$

Radiosity Steps:

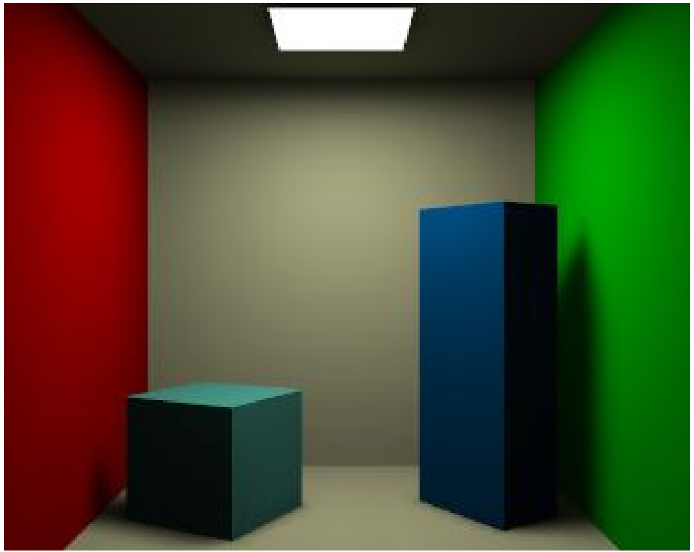
1. Generate Model (set up the scene)
2. Compute Form Factors and set the Radiosity Matrix
3. Solve the linear system
4. Render the scene

$$\begin{pmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \dots & -\rho_1 F_{1N} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \dots & -\rho_2 F_{2N} \\ \vdots & \vdots & \dots & \vdots \\ -\rho_N F_{N1} & -\rho_N F_{N2} & \dots & 1-\rho_N F_{NN} \end{pmatrix} \boxed{\begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_N \end{pmatrix}} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_N \end{pmatrix}$$

Radiosity Steps:

1. Generate Model (set up the scene)
2. Compute Form Factors and set the Radiosity Matrix
3. Solve the linear system
4. Render the scene

$$\begin{pmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \ldots & -\rho_1 F_{1N} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \ldots & -\rho_2 F_{2N} \\ \vdots & \vdots & \ldots & \vdots \\ -\rho_N F_{N1} & -\rho_N F_{N2} & \ldots & 1-\rho_N F_{NN} \end{pmatrix} \boxed{\begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_N \end{pmatrix}} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_N \end{pmatrix}$$



Radiosity Steps:

1. Generate Model
2. Compute Form Factors and set the Radiosity Matrix
3. Solve the linear system
4. Render the scene

$$\begin{pmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \dots & -\rho_1 F_{1N} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \dots & -\rho_2 F_{2N} \\ \vdots & \vdots & \dots & \vdots \\ -\rho_N F_{N1} & -\rho_N F_{N2} & \dots & 1-\rho_N F_{NN} \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_N \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_N \end{pmatrix}$$

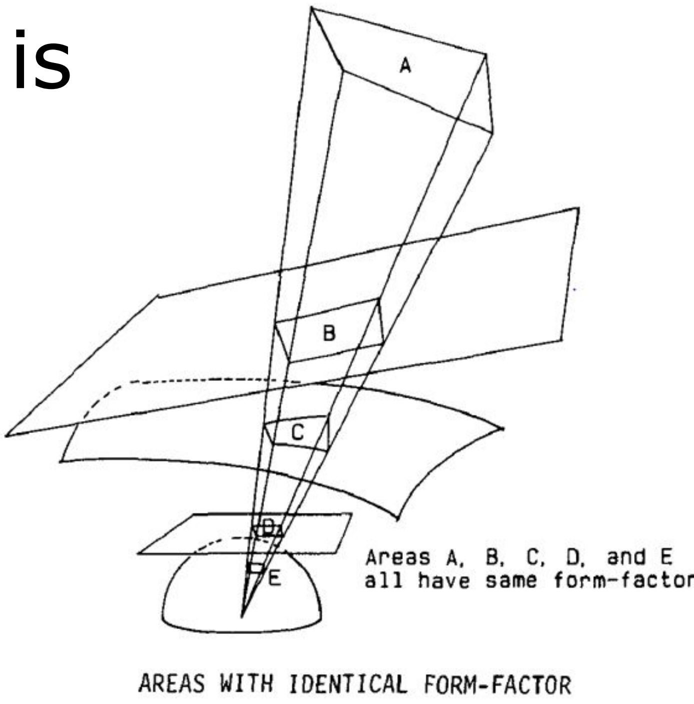
- Where do we resume from if objects are moved?
- Where do we resume from if the lighting is changed?
- Where do we resume from if the reflectance parameters of the scene are modified?
- Where do we resume from if the view point changes?

Radiosity Features:

- Very costly
- The faces must be subdivided into small patches to reduce the artefacts
- The computational cost for calculating the form factors is expensive
  - Quadratic to the number of patches
- Solving for B_i is also very costly
  - Cubic to the number of patches
- Cannot handle specular light

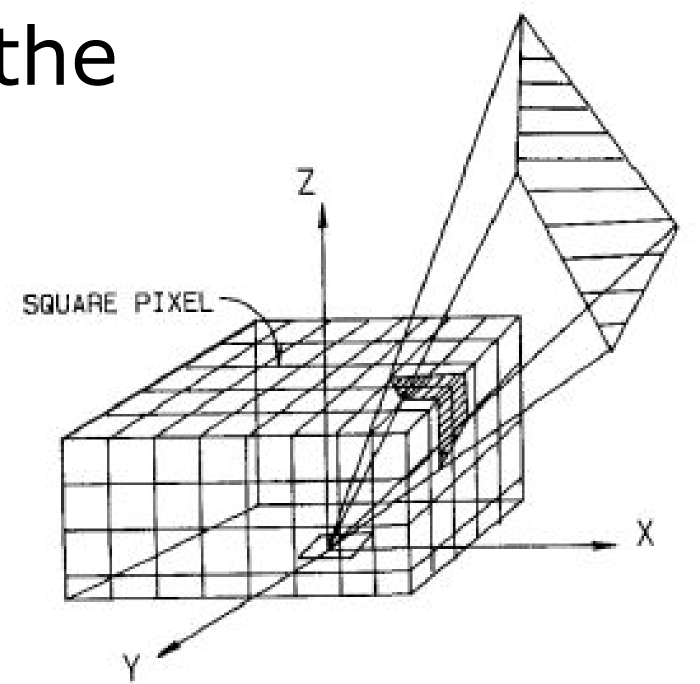
Hemicube: by Michael Cohen'85

- Accelerates the computation of the form factor
- The form factor for the right four faces with respect to a small patch in the bottom is the same
- Then, we can project the patches onto a hemicube



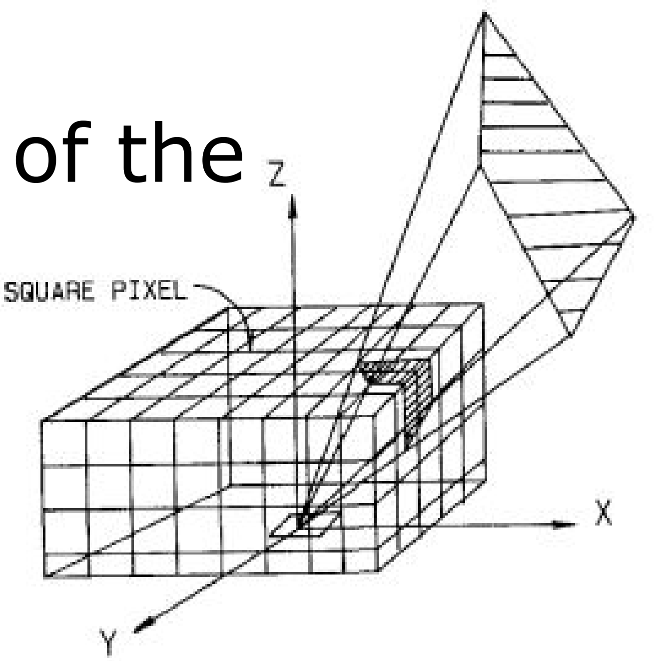
Hemicube (2)

- Prepare a hemicube around the patch i
- Project those polygons you want to compute the form factor with patch i onto the hemicube
- Then, compute the form factor between them



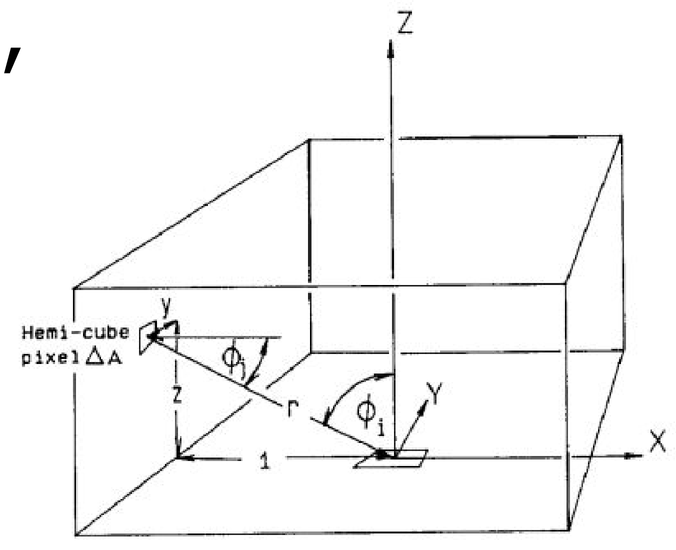
Hemicube (3)

- This can be done by perspective projection
- We can use the Z-buffer algorithm to find the closest polygon
  - Handling the occlusion
  - Setting the form factor of the pairs that occlude each other to zero



Hemicube (4)

- The form factor between each pixel of the hemicube and the patch at the origin can be pre-computed and saved in a table
- Only 1/8 of all are needed, thanks to symmetry



Summary

- Bump maps can be used to increase the reality without increasing the resolution of the meshes
- Global illumination methods simulate inter-reflectance
- Radiosity can simulate diffuse inter-reflectance
- The form factor computation can be accelerated by hemi-cube.

Readings

- Blinn, "Simulation of Wrinkled Surfaces", Computer Graphics, (Proc. Siggraph), Vol. 12, No. 3, August 1978, pp. 286-292.
- Real-time Rendering, Chapter 5,1-5.2
- http://www.blacksmith-studios.dk/projects/downloads/tangent_matrix_derivation.php
- Cohen et al., The hemi-cube: a radiosity solution for complex environments, SIGGRAPH '85