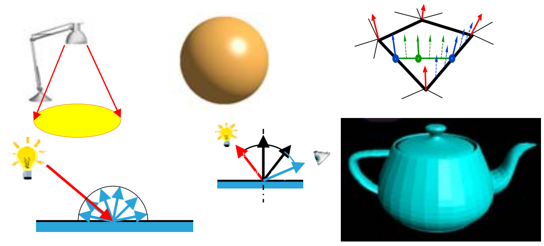
Illumination Models and Surface Rendering Methods

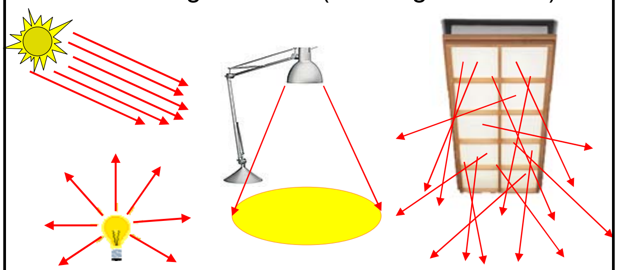


Illumination

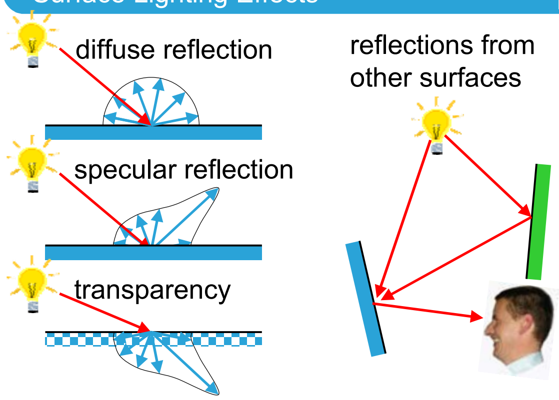
- light sources
- basic model
  - ambient light
  - diffuse shading
  - specular highlights
- shading interpolation
  - Gouraud Shading
  - Phong Shading

Light Sources

- directional light
- point light source (sometimes directional)
- distributed light source (“area light source”)



Surface Lighting Effects



Basic Illumination Models

- empirical models
- lighting calculations
  - surface properties (glossy, matte, opaque,…)
  - background lighting conditions
  - light-source specification
  - reflection, absorption
- ambient light (background light) I_a
  - approximation of global diffuse lighting effects

Ambient Light Reflection

- constant over a surface
- independent of viewing direction
- diffuse-reflection coefficient k_d $(0 \le k_d \le 1)$

$$I_{ambdiff} = k_d I_a$$

Illumination and Shading

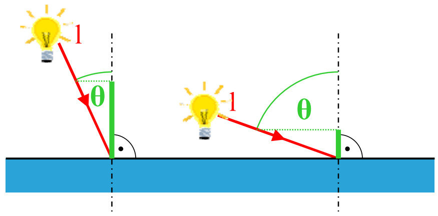
- shaded surfaces generate a spatial impression

the flatter light falls on a surface, the darker it will appear

- therefore:

we need the incident light direction
or
the position of the (point) light source

Lambert's Law



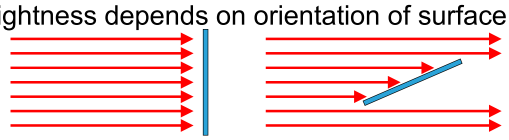
$I = I_l \cdot \cos\theta$

when considering the material:

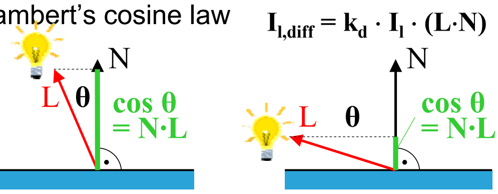
$I = k_d \cdot I_l \cdot \cos\theta$

Lambertian (Diffuse) Reflection

- ideal diffuse reflectors (Lambertian reflectors)
- brightness depends on orientation of surface



- Lambert's cosine law $I_{l,diff} = k_d \cdot I_l \cdot (L{\cdot}N)$



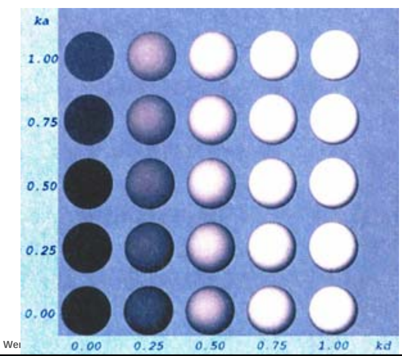
Diffuse Reflection Coefficient

- varying k_d

result for varying values of k_d, $I_a = 0$

Ambient plus Diffuse Reflection

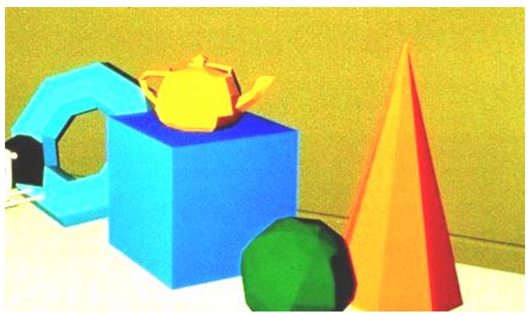
- total diffuse reflection $I_{l,diff} = k_a I_a + k_d I_l (N \cdot L)$



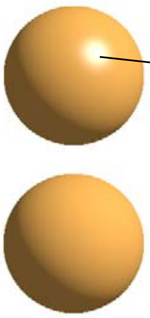
(sometimes k_a for ambient light)

Ambient plus Diffuse Reflection

- total diffuse reflection $I_{l,diff} = k_a I_a + k_d I_l (N \cdot L)$



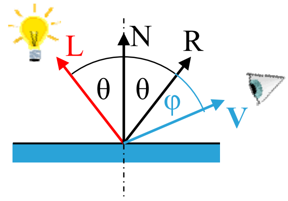
Specular Highlights



this area must be lighter than the shading model calculates, because the light source is reflected directly into the viewer's eye

Specular Reflection Model

- reflection of incident light around specular-reflection angle

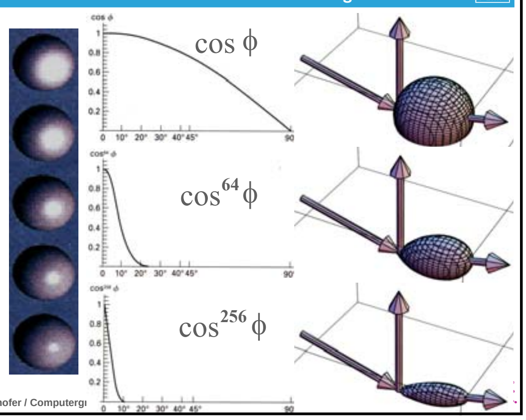


- empirical Phong model

$$I_{l,spec} = k_s \cdot I_l \cdot \cos^{n_s}\varphi$$

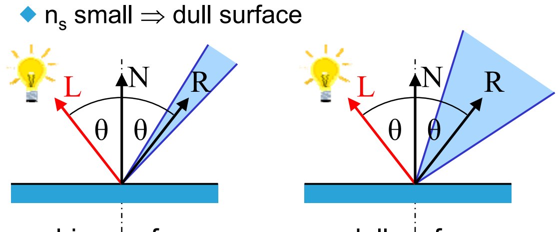
Specular Reflection Coefficient n_s

$\cos^{n_s} \phi$



Specular Reflection Coefficient

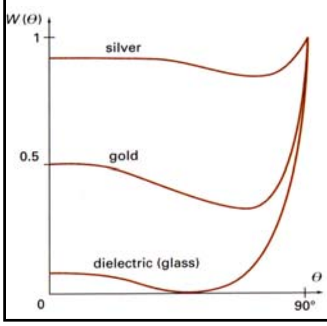
- empirical Phong model $I_{l,spec} = k_s \cdot I_l \cdot \cos^{n_s}\varphi$
  - n_s large ⇒ shiny surface
  - n_s small ⇒ dull surface



Fresnel Specular Reflection Coefficient

- Fresnel's laws of reflection
  - specular reflection coefficient $W(\theta)$

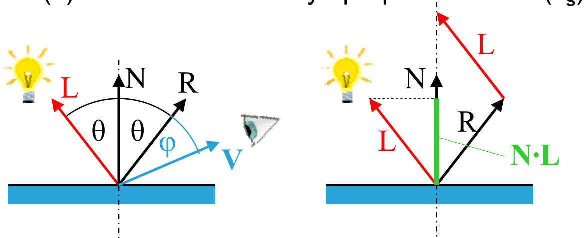
$$I_{l,spec} = W(\theta) I_l \cos^{n_s}\phi$$



specular reflection coefficient as a function of angle of incidence for different materials

Simple Specular Reflection

- $W(\theta) \approx$ constant for many opaque materials (k_s)



$$I_{l,spec} = k_s I_l (V \cdot R)^{n_s}$$

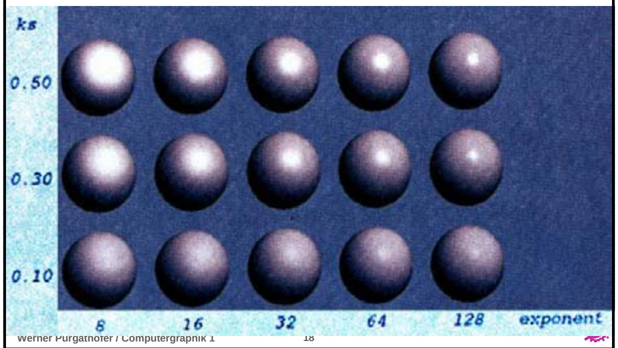
calculation of R:

$$R + L = (2N{\cdot}L)N$$

$$R = (2N{\cdot}L)N - L$$

Specular Reflection Results

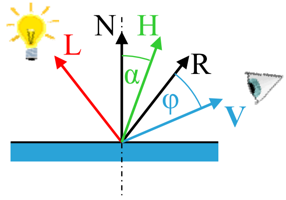
$$I_{l,spec} = k_s I_l (V \cdot R)^{n_s}$$



Simplified Specular Reflection

- simplified Phong model with halfway vector H

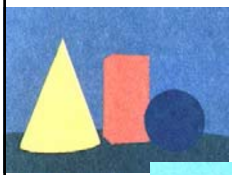
$$I_{spec} = k_s I_l (V \cdot R)^{n_s} \quad \rightarrow \quad I_{spec} = k_s I_l (N \cdot H)^{n_s}$$



$$H = \frac{L+V}{|L+V|}$$

Diffuse and Specular Reflection

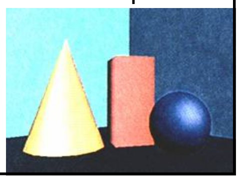
$$I = k_a I_a + \sum_{l=1}^{n} I_l \left[k_d (N \cdot L_l) + k_s (N \cdot H_l)^{n_s} \right]$$



ambient

ambient + diffuse

ambient + diffuse + specular



Other Aspects

- intensity attenuation with distance
- anisotropic light sources (Warn model)
- transparency (Snell's law)
- atmospheric effects
- shadows
- …

Polygon-Rendering Methods

- application of illumination model to polygon rendering
- constant-intensity shading (flat shading)
  - single intensity for each polygon

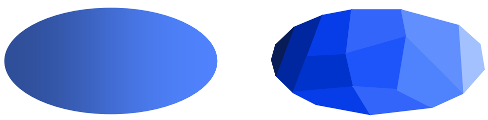
flat ***Gouraud***

Polygon Shading: Interpolation

- the shading of a polygon is not constant, because it normally is only an approximation of the real surface ⇒ **interpolation**

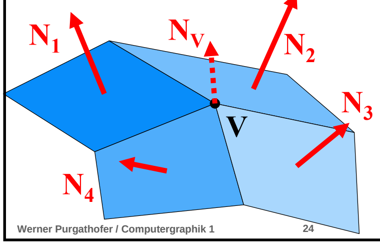
Gouraud shading: intensities

Phong shading: normal vectors



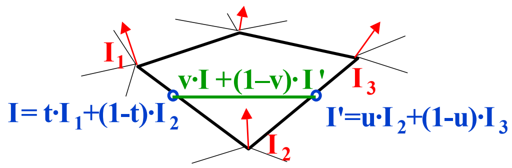
Gouraud Shading Overview

- intensity-interpolation
  - determine average unit normal vector at each polygon vertex
  - apply illumination model to each vertex
  - linearly interpolate vertex intensities



$$N_V = \frac{\sum_{k=1}^{n} N_k}{\left|\sum_{k=1}^{n} N_k\right|}$$

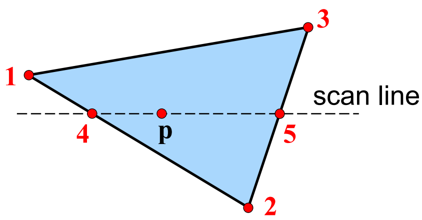
Gouraud Shading



1. find normal vectors at corners and calculate shading (intensities) there: I_i
2. interpolate intensities along the edges linearly: I, I'
3. interpolate intensities along scanlines linearly: I_p

Gouraud Shading

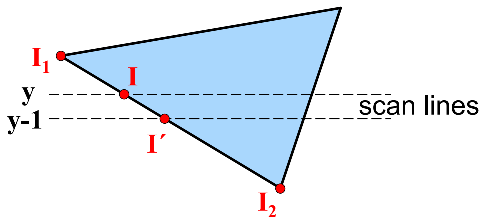
- interpolating intensities



$$I_4 = \frac{y_4 - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_4}{y_1 - y_2} I_2 \qquad I_p = \frac{x_5 - x_p}{x_5 - x_4} I_4 + \frac{x_p - x_4}{x_5 - x_4} I_5$$

Gouraud Shading

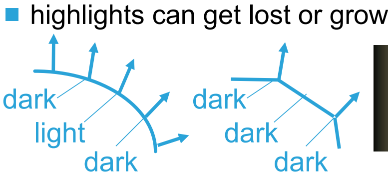
- incremental update



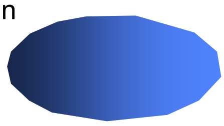
$$I = \frac{y - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y}{y_1 - y_2} I_2 \qquad I' = I + \frac{I_2 - I_1}{y_1 - y_2}$$

Problems of Gouraud Shading

- highlights can get lost or grow



- corners on silhouette remain

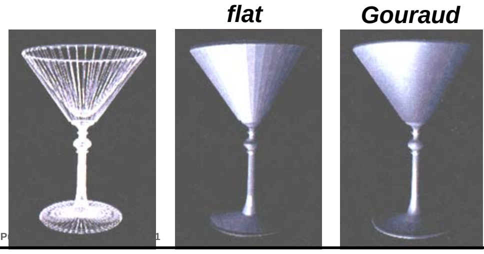


- Mach band effect is visible at some edges

Gouraud Shading Results

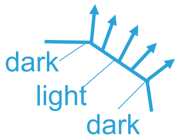
- no intensity discontinuities
- Mach bands due to linear intensity interpolation
- problems with highlights

flat *Gouraud*



Phong Shading

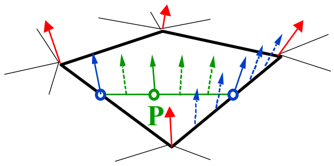
- instead of intensities the normal vectors are interpolated, and for every point the shading calculation is performed separately



Phong Shading Principle

- normal-vector interpolation
  - determine average unit normal vector at each polygon vertex
  - linearly interpolate vertex normals
  - apply illumination model along each scan line

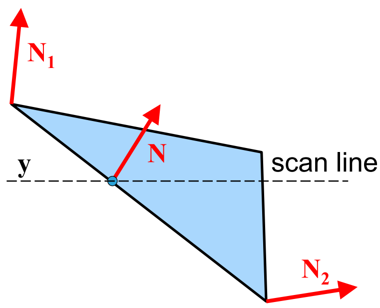
Phong Shading Overview



1. normal vectors at corner points
2. interpolate normal vectors along the edges
3. interpolate normal vectors along scanlines & calculate shading (intensities) for every pixel

Phong Shading Normal Vectors

- normal-vector interpolation



$$\mathbf{N} = \frac{y - y_2}{y_1 - y_2}\mathbf{N}_1 + \frac{y_1 - y}{y_1 - y_2}\mathbf{N}_2$$

Phong Shading

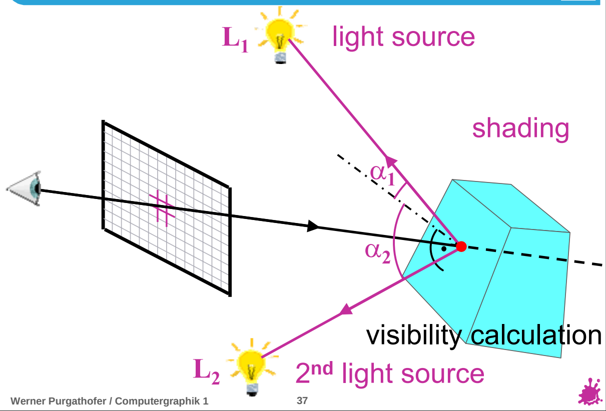
- incremental normal vector update along and between scan lines
- comparison to Gouraud shading
  - better highlights
  - less Mach banding
  - more costly

Flat/Gouraud/Phong Comparison

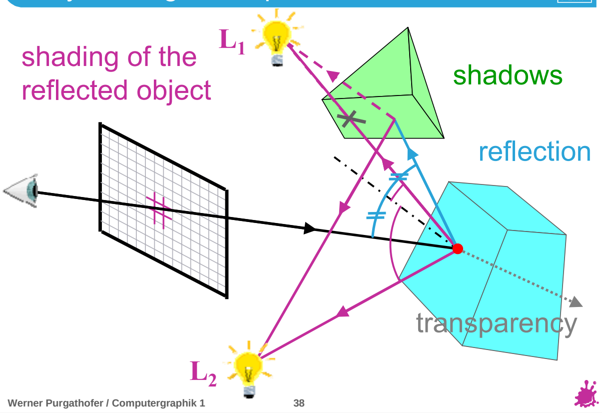
Surface-Rendering Methods

- polygon rendering methods
- **ray-tracing**
- radiosity
- environment mapping
- texture mapping
- bump mapping

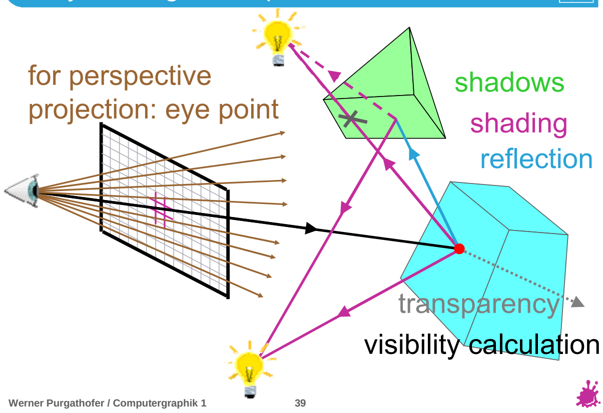
Ray-Tracing Concepts



Ray-Tracing Concepts



Ray-Tracing Concepts



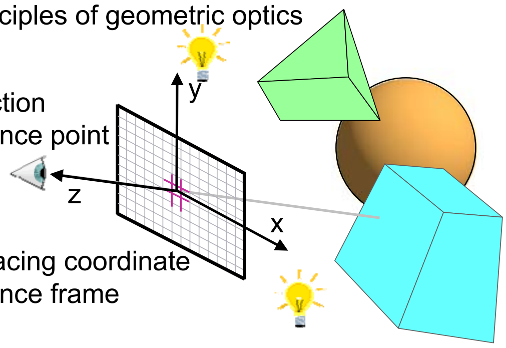
Ray-Tracing Properties

- highly realistic images
- very time consuming
- global reflection, transmission
- visible-surface detection
- shadows
- transparency
- multiple light sources

Ray-Tracing

- principles of geometric optics

projection reference point



ray-tracing coordinate reference frame

primary ray = eyepoint + s·(pixel – eyepoint)

Shading: Diffuse Shading

$I_d = xxx$

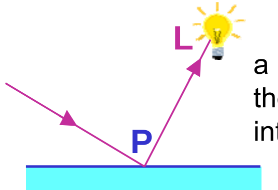
I_d … illumination caused by diffuse shading
xxx … any shading model
(Phong, Blinn, Cook/Torrance,…)

Ray-Tracing: Shadows

ray = intersection point + s · vector to light source

ray = P + s · (L – P)

P …intersection point
L …light source position



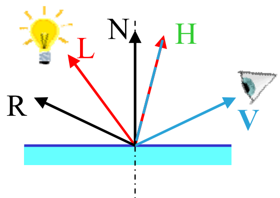
a light source influences the result only if there is no intersection with $0 < s < 1$

Ray-Tracing: Shadows and Shading

- shadow ray along L
- ambient light $k_a I_a$
- diffuse reflection $k_d(N \cdot L)$
- specular reflection $k_s(H \cdot N)^{n_s}$

$I_d = k_d I_a + k_d(N \cdot L) + k_s(H \cdot N)^{n_s}$

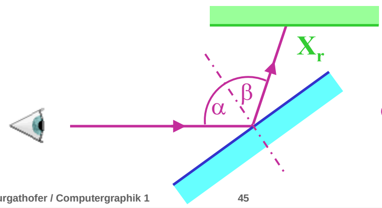
unit vectors at an object surface intersected by an incoming ray from direction V



Ray-Tracing: Reflection

$I_r = k_r \cdot X_r$

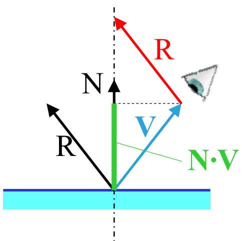
I_r … illumination caused by reflection
k_r … reflection coefficient of the material
X_r … shading in the reflected direction



$\alpha = \beta$

Ray-Tracing: Reflection Ray

- calculation of reflection ray



$R + V = (2N \cdot V)N$

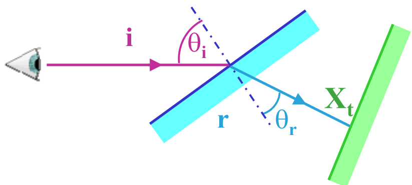
$R = (2N \cdot V)N - V$

if $V = -u$ [book]:
$R = u - (2u \cdot N)N$

Ray-Tracing: Transparency

$I_t = k_t \cdot X_t$

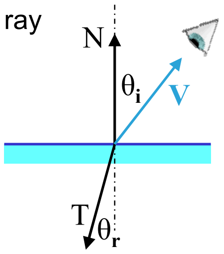
I_t … illumination caused by transparency
k_t … transparency coefficient of the material
X_t … shading in the transparency direction



$\sin\theta_i : \sin\theta_r = \eta_r : \eta_i$

Ray-Tracing: Transparency Ray

- calculation of transparency ray



$$\sin\theta_r = \frac{\eta_i}{\eta_r}\sin\theta_i$$

$$T = -\frac{\eta_i}{\eta_r}V - \left(\cos\theta_r - \frac{\eta_i}{\eta_r}\cos\theta_i\right)N$$

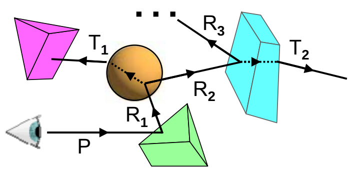
Ray-Tracing: A Complete Shading Method

$$I = I_d + I_r + I_t$$

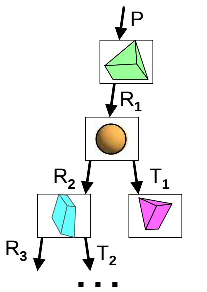
additional requirement: $k_d + k_r + k_t \leq 1$

Ray-Tracing: Rays & Ray Tree

- primary, secondary rays



reflection and refraction ray paths for one pixel

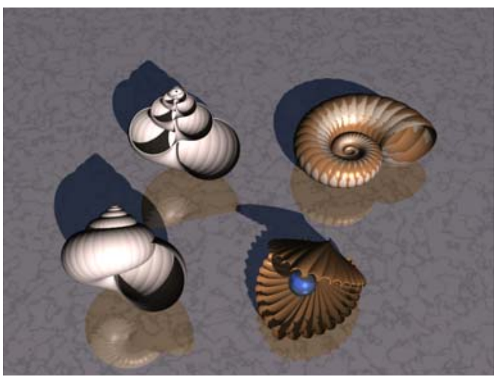


corresponding binary ray-tracing tree

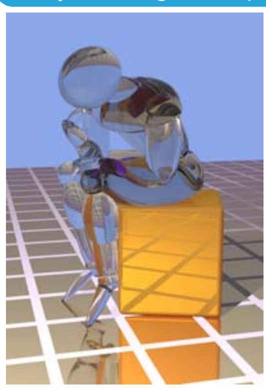
Ray-Tracing: Basic Algorithm

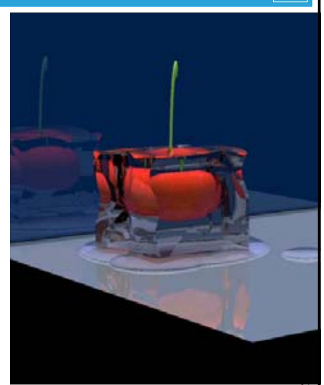
```
FOR all pixels P0 DO
1.trace primary ray eye -> P0
  find closest intersection P
2.FOR all light sources L DO
    trace shadow feeler P -> L
    IF no inters. between P, L
    THEN shading+=influence of L
3.IF surface of P is reflective
  THEN trace secondary ray;
    shading+=influence of refl.
4.IF surface of P is transparent
  THEN trace secondary ray;
    shading+=influence of transp.
```


Ray-Tracing Examples



Ray-Tracing Examples





True Global Illumination Example

Requirements for Object Data

(to use them for ray-tracing)

- intersection calculation ray - object possible
- surface normal calculation possible
  - B-Rep: simple
  - CSG: recursive evaluation

Ray-Surface Intersection

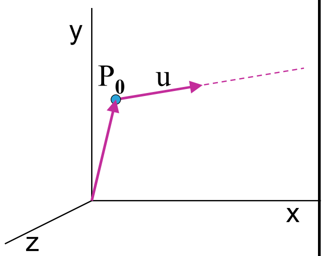
- ray equation

$$P = P_0 + s \cdot u$$

- for primary rays

$$u = \frac{P_{pix} - P_{prp}}{|P_{pix} - P_{prp}|}$$

- for secondary rays $u = R$, $u = T$



describing a ray with an initial-position vector P_0 and unit direction vector u

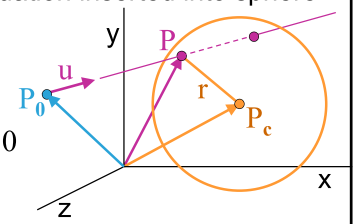
Ray-Sphere Intersection

- parametric ray equation inserted into sphere equation

$$|P - P_c|^2 - r^2 = 0$$

$$|(P_0 + su) - P_c|^2 - r^2 = 0$$

$$\Delta P = P_c - P_0$$



$$s^2 - 2(u \cdot \Delta P)s + (|\Delta P|^2 - r^2) = 0 \qquad (u^2 = 1)$$

$$s = u \cdot \Delta P \pm \sqrt{(u \cdot \Delta P)^2 - |\Delta P|^2 + r^2}$$

Ray-Sphere Intersection

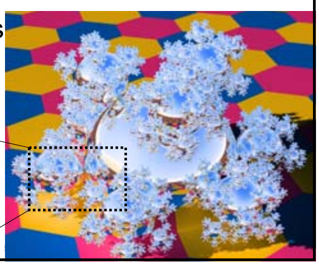
- discriminant negative ⇒ no intersections

$$\mathbf{s = u \cdot \Delta P \pm \sqrt{(u \cdot \Delta P)^2 - |\Delta P|^2 + r^2}}$$

➡ $$\mathbf{s = u \cdot \Delta P \pm \sqrt{r^2 - |\Delta P - (u \cdot \Delta P)u|^2}}$$ because $u^2=1$

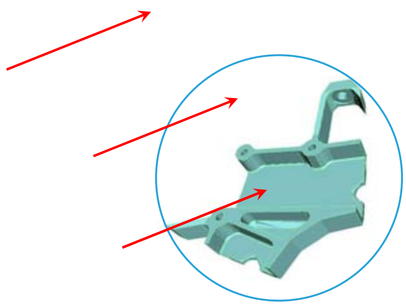
(to avoid roundoff errors when $r^2 << |\Delta P|^2$)

"sphereflake"



Ray-Polyhedron Intersection

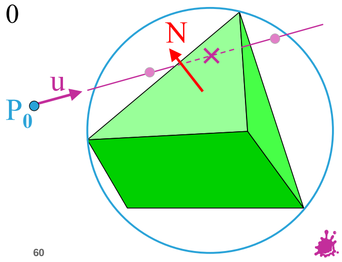
- use ***bounding sphere*** to eliminate easy cases



Ray-Polyhedron Intersection

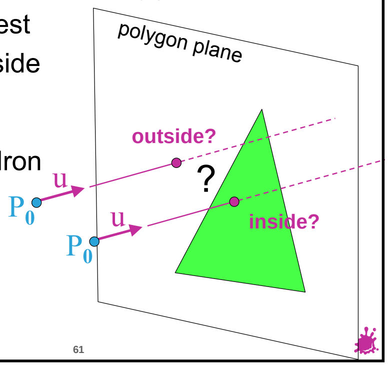
- use bounding sphere to eliminate easy cases
- locate front faces $\quad u \cdot N < 0$
- solving plane equation

$$Ax + By + Cz + D = 0$$
$$N = (A, B, C)$$
$$N \cdot P = -D$$
$$N \cdot (P_0 + su) = -D$$
$$s = -\frac{D + N \cdot P_0}{N \cdot u}$$



Ray-Polyhedron Intersection

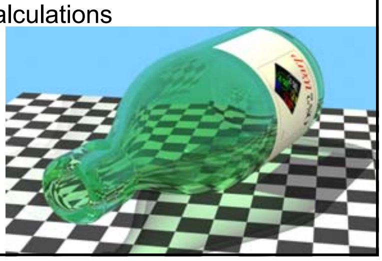
- intersection point inside polygon boundaries?
- inside-outside test
- smallest s to inside point is first intersection point of polyhedron



Ray-Surface Intersection

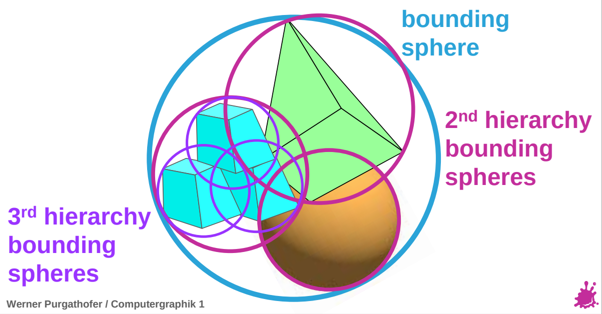
- quadric, spline surfaces:
  - parametric ray equation inserted into surface definition
  - methods like numerical root-finding, incremental calculations

ray-traced scene with NURBS surfaces and multiple reflection / refraction



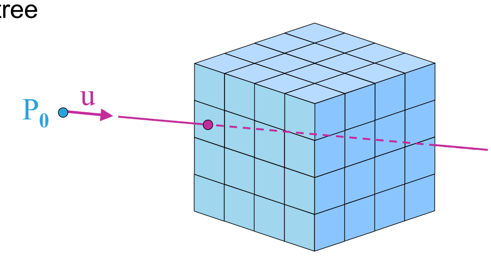
Reducing Object-Intersection Calculations

- bounding volumes
- bounding volume hierarchies



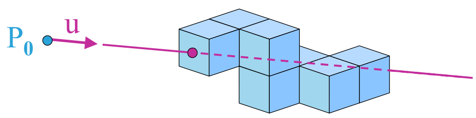
Reducing Object-Intersection Calculations

- space-subdivision methods
  - regular grid
  - octree



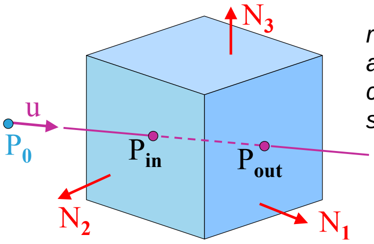
Reducing Object-Intersection Calculations

- space-subdivision methods
  - regular grid
  - octree



Reducing Object-Intersection Calculations

- space-subdivision methods
  - incremental grid traversal
    - 3D Bresenham
    - processing of potential exit faces

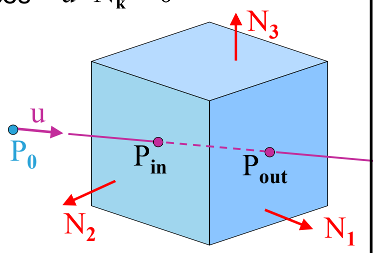


ray traversal through a subregion of a cube enclosing a scene

Incremental Grid Traversal

- ray direction u / ray entry position P_{in}
- potential exit faces $u \cdot N_k > 0$
- normal vectors

$$N_k = \begin{cases} (\pm 1,0,0) \\ (0,\pm 1,0) \\ (0,0,\pm 1) \end{cases}$$



- check signs of components of u

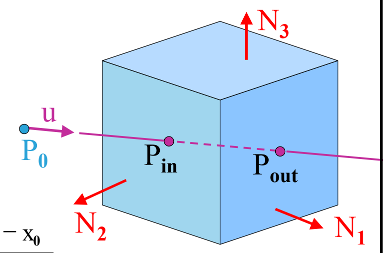
Incremental Grid Traversal

- calculation of exit positions, select smallest s_k

$$P_{out,k} = P_{in} + s_k u$$

$$N_k \cdot P_{out,k} = -D_k$$

$$s_k = \frac{-D_k - N_k \cdot P_{in}}{N_k \cdot u}$$

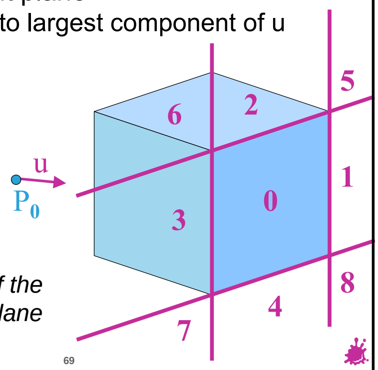


- example:

$$N_k = (1,0,0) \quad s_k = \frac{x_k - x_0}{u_x}$$

Incremental Grid Traversal

- variation: trial exit plane
  - perpendicular to largest component of u
  - exit point in 0
    => done
  - {1, 2, 3, 4}
    => side clear
  - {5, 6, 7, 8}
    => extra calc.



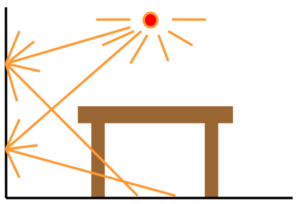
sectors of the trial exit plane

Surface-Rendering Methods

- polygon rendering methods
- ray-tracing
- **radiosity**
- environment mapping
- texture mapping
- bump mapping

Radiosity Method

- describes the physical process of light distribution in a diffuse reflecting environment

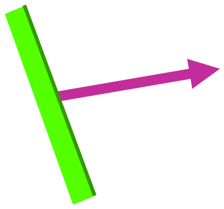


areas that are not illuminated directly are also not completely dark

every object acts as a secondary light source

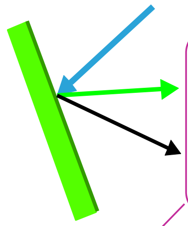
Radiosity

- Radiosity **B** is the „radiant flux per unit area“ that is leaving a surface



Radiosity Equation

incoming light from the environment $\int_{hemi} I(x)\,dx = \int_{hemi} dB$



self emission (only for light sources) E

reflected light from environment $\rho \cdot \int_{hemi} dB$

radiosity of the point $B = E + \rho \cdot \int_{hemi} dB$

Radiosity Equation

- to calculate the light influence between surfaces

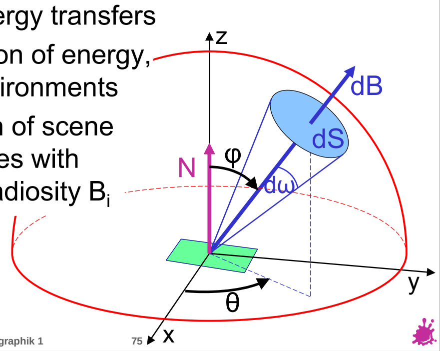
***Radiosity* = total light leaving a surface point**

$$B = E + \rho \cdot \int_{hemi} dB$$

B…radiosity	hemi…half space over point
E…self emission	ρ …reflection coefficient

Radiosity Properties

- diffuse interreflections in a scene
- radiant energy transfers
- conservation of energy, closed environments
- subdivision of scene into patches with constant radiosity B_i



Radiosity: Subdivision into Patches

the scene is discretized into **n "patches"** (plane polygons) P_i, for each of these patches a constant radiosity B_i is assumed:

$$B = E + \rho \cdot \int_{hemi} dB \quad \Rightarrow \quad B_i = E_i + \rho_i \cdot \sum_{j=1}^{n} B_j \cdot F_{ij}$$

ρ_i	diffuse reflection coefficient of patch k
F_{ij}	“formfactor”: describes how much % of the influence on patch i comes from patch j; geometric size

Radiosity Model

$$B_i = E_i + \rho_i \cdot \sum_{j=1}^{n} B_j \cdot F_{ij}$$

- B_i radiosity of patch i
- E_i self-emission of patch i
- $\Sigma B_j F_{ij}$ contribution of other patches
- F_{ij} form factor, defines
  - contribution of B_i on patch j which is equal to
  - contribution of patch j to B_i
- ρ_i reflectivity coefficient of patch i (“*albedo*”)

Radiosity Equation

- solving the radiosity equation

$$B_i = E_i + \rho_i \sum_{j \neq i} B_j F_{ij}$$

$$B_i - \rho_i \sum_{j \neq i} B_j F_{ij} = E_i$$

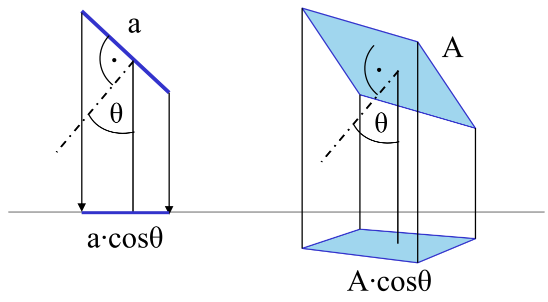
$$\begin{bmatrix} 1 & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1 & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1 \end{bmatrix} \cdot \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix}$$

Radiosity Equation: Form Factors

surface properties | form factors (constants) | radiosities (unknowns) | surface properties

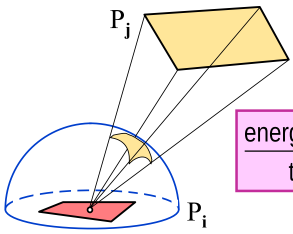
$$\begin{bmatrix} 1 & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1 & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1 \end{bmatrix} \cdot \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix}$$

Projection of a Polygon



Radiosity: Form Factors

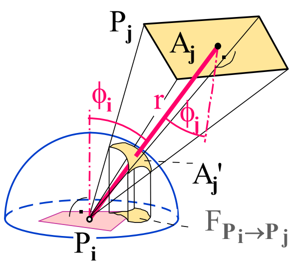
- form factor F_{ij}: contribution of patch P_j to B_i
  = contribution of B_i to patch P_j



$$\frac{\text{energy reaching patch j from patch i}}{\text{total energy leaving patch i}}$$

Radiosity: Form Factors

- form factor F_{ij}: contribution of patch P_j to B_i
  = contribution of B_i to patch P_j



$$F_{ij} = \frac{\cos\phi_i \cos\phi_j A_j}{\pi r^2}$$

because $\sum_{j=1}^{n} F_{ij} = 1$

Radiosity: Form Factors

- form factor F_{ij}: contribution of patch P_j to B_i
  = contribution of B_i to patch P_j

$$F_{ij} = \frac{\cos\phi_i \cos\phi_j A_j}{\pi r^2}$$

- more precisely: form factor is sum over contributions from P_j averaged over area A_i

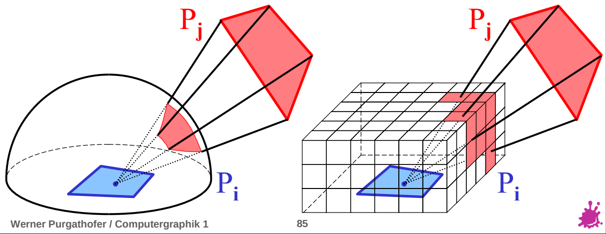
$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\phi_i \cos\phi_j}{\pi r^2} dA_j \, dA_i$$

Radiosity: Form Factors

- form factor properties
  - conservation of energy $\sum_{j=1}^{n} F_{ij} = 1$
  - uniform light reflection $A_i F_{ij} = A_j F_{ji}$
  - no self-incidence $F_{ii} = 0$

Radiosity: Form Factors

- form factor calculation
  - most expensive step in radiosity calculation
  - numerical integration (Monte Carlo methods)
  - *hemicube* approach (replaces hemisphere)



Radiosity Equation

- solving the radiosity equation
  - Gaussian elimination
  - Gauss-Seidel iteration

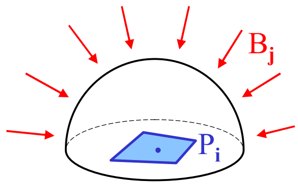
$$\begin{bmatrix} 1 & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1 & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1 \end{bmatrix} \cdot \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix}$$

- very time and storage intensive

Radiosity Equation

- solving the radiosity equation
  - Gauss-Seidel iteration

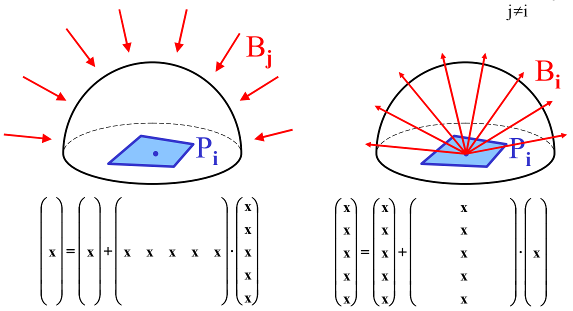
$$B_i^{k+1} = E_i + \rho_i \sum_{j \neq i} B_j^k F_{ij}$$



"gathering"

Radiosity Equation

- "gathering" vs. "*shooting*" $B_i^{k+1} = E_i + \rho_i \sum_{j \neq i} B_j^k F_{ij}$



Progressive Refinement Radiosity (1)

- "shooting"
  - select brightest patch i and distribute its radiosity B_i

$$B_i = E_i + \rho_i \sum_{j \neq i} B_j F_{ij} \Rightarrow \begin{matrix} B_{i \text{ due to } B_j} = \rho_i B_j F_{ij} \\ B_{j \text{ due to } B_i} = \rho_j B_i F_{ji} \end{matrix}$$

$$\Downarrow$$

$$B_{j \text{ due to } B_i} = \rho_j B_i F_{ij} \frac{A_i}{A_j} \Leftarrow A_i F_{ij} = A_j F_{ji}$$

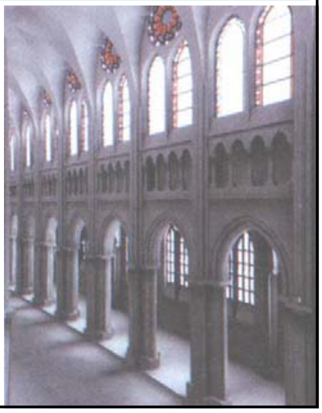
Progressive Refinement Radiosity (2)

[one refinement step]

```
select patch i with highest Ai*ΔBi
for selected patch i {
    set up hemicube
    calculate form factors Fij
}
for each patch j {
    Δrad := ρj*ΔBi*Fij*Ai/Aj
    ΔBj := ΔBj + Δrad
    Bj := Bj + Δrad
}
ΔBi := 0
```


Progressive Refinement Radiosity (3)

- initially $\Delta B_i = B_i = E_i$,
- select patch with highest $\Delta B_i A_i$



cathedral rendered with progressive refinement radiosity

form factors computed with ray-tracing methods

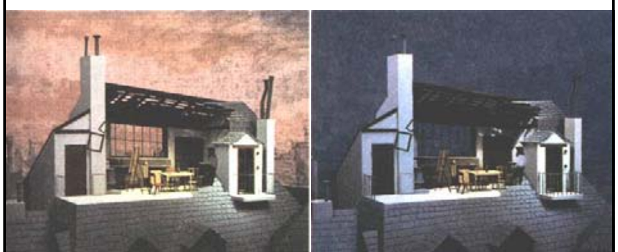
Radiosity Example Images (1)

image of a constructivist museum rendered with progressive refinement radiosity

Radiosity Example Images (2)

stair tower of a building at Cornell University rendered with progressive refinement radiosity

Radiosity Example Images (3)



2 lighting schemes for an opera production:
(left) day view (right) night view

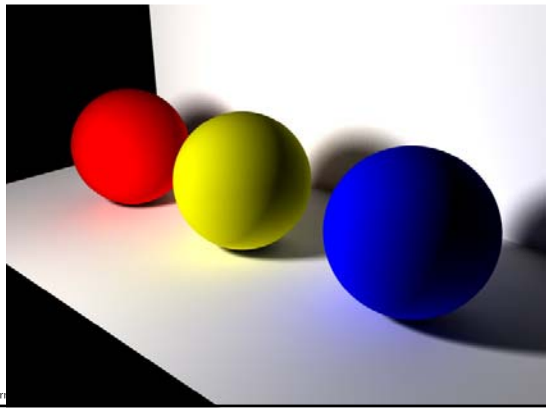
Radiosity Aspects (1)

- radiosity is viewpoint-independent
- needs a rendering step to display
  - polygon rendering
  - Gouraud shading
  - ray-tracing
  - …
- combination with ray-tracing enables
  - reflections
  - shadows
  - …

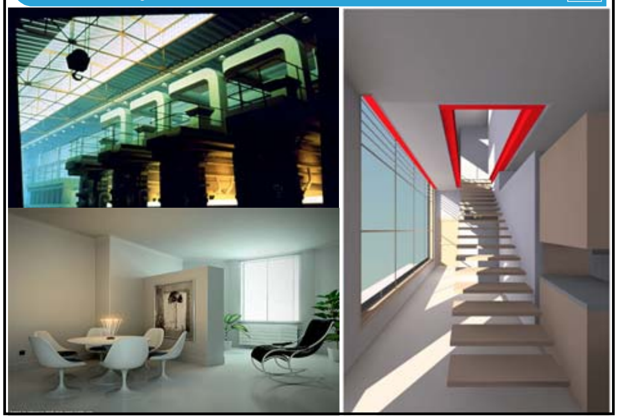
Radiosity Aspects (2)

- hierarchical radiosity
  - to reduce number of form factors
- stochastic methods
  - to calculate form factors
  - to solve radiosity equation system
- path tracing
  - trace light rays (forward tracing!)
  - store effect of light hitting a patch
  - interpolation

Radiosity Results



Radiosity Results

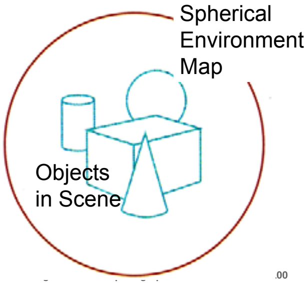


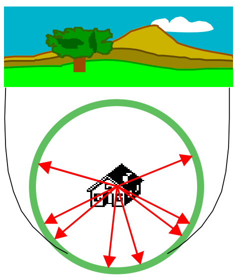
Surface-Rendering Methods

- polygon rendering methods
- ray-tracing
- radiosity
- **environment mapping**
- **texture mapping**
- **bump mapping**

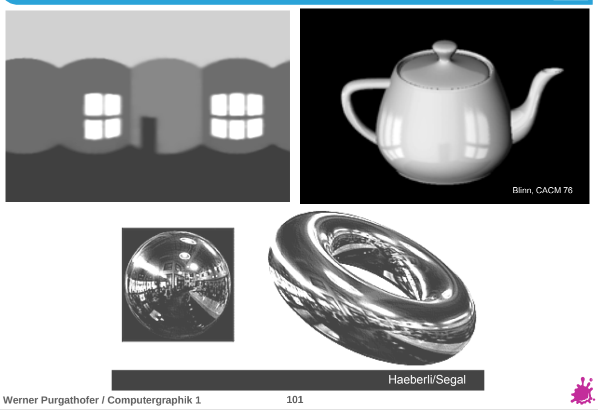
Environment Mapping Principle

- reflection mapping
- defined over surface of an enclosing universe (sphere, cube, cylinder)



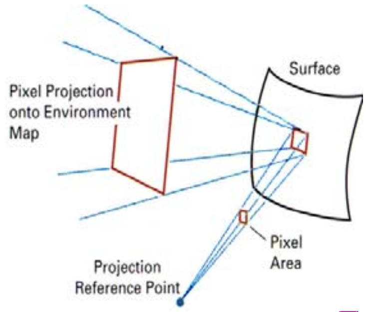


Environment Mapping Example

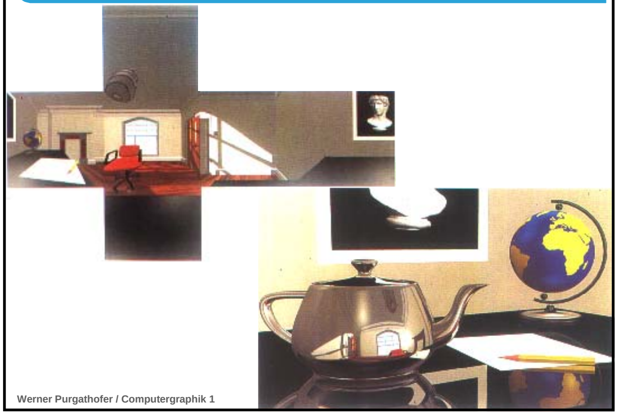


Environment Mapping Calculation

- information in the environment map
  - intensity values for light sources
  - sky
  - background objects
- pixel area
  - projected onto surface
  - reflected onto environment map

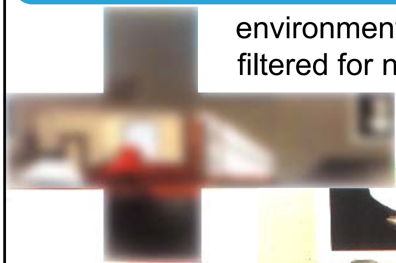


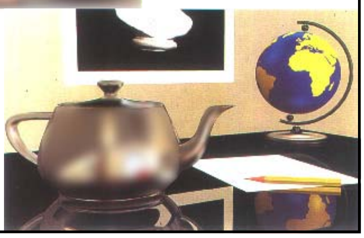
Environment Mapping Example



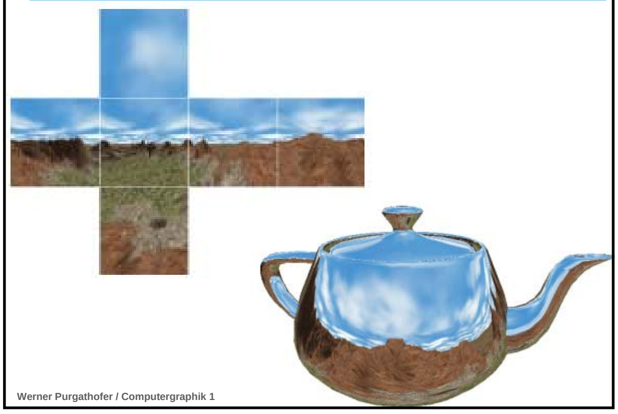
Environment Mapping Filtering

environment maps may be filtered for not so reflective surfaces



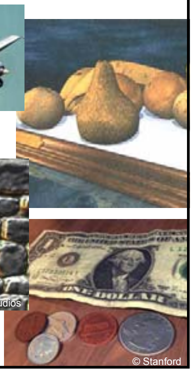


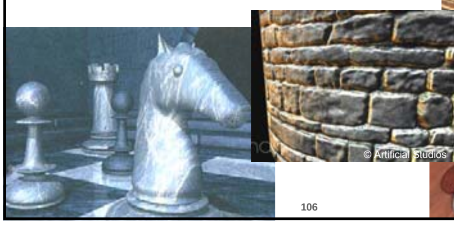
Environment Mapping Example



Adding Surface Detail

- most objects do not have smooth surfaces
  - brick walls
  - gravel roads
  - shag carpets
- surface texture required





Adding Surface Detail

- modeling surface detail with polygons
  - small polygon facets (e.g., checkerboard squares)
  - facets overlaid on surface polygon (parent)
  - parent surface used for visibility calculations
  - facets used for illumination calculations
  - impractical for complicated surface structure

Texture Mapping: Principle

- texture patterns mapped onto surfaces
- texture pattern:
  - raster image
  - or procedure (modifies surface intensities)

Texture Space: (s,t) Array Coordinates → **Object Space:** (u,v) Surface Parameters → **Image Space:** (x,y) Pixel Coordinates

Texture-Surface Transformation; Viewing & Projection Transformation

Texture Mapping: Samples

Texture Mapping: Transformation

- texture mapping
  - texture scanning (s,t)→(x,y)
  - inverse scanning (x,y)→(s,t)

Texture Space: (s,t) Array Coordinates → **Object Space:** (u,v) Surface Parameters → **Image Space:** (x,y) Pixel Coordinates

Texture-Surface Transformation M_T; Viewing & Projection Transformation M_{VP}

- texture-surface transformation

$$u = u(s,t) = a_u s + b_u t + c_u$$
$$v = v(s,t) = a_v s + b_v t + c_v$$

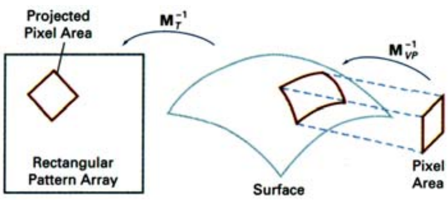
Texture Mapping (dt.)

Textur-Raum: (s,t) Array Koordinaten → **Objekt-Raum:** (u,v) Flächen-Parameter → **Bild-Raum:** (x,y) Pixel Koordinaten

Textur-Objekt Transformation; Viewing & Projektions-Transformation

Texture Mapping: Inverse Transformation

- projecting pixel areas to texture space = = inverse scanning (x,y)→(s,t)



- calculation of M^{-1}_{VP} M^{-1}_{T}
- anti-aliasing with filter operations

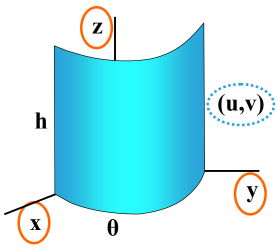
Texture-Mapping: Cylindrical Surface

- M_{VP}

$x^2+y^2=r^2$, $x,y \geq 0$, $0 \leq z \leq h$

$u = \theta$, with $0 \leq \theta \leq \pi/2$
$v = z$ with $0 \leq z \leq h$

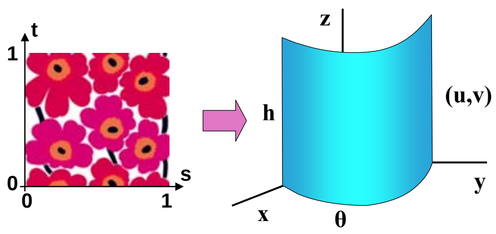
$x = r \cdot \cos u$,
$y = r \cdot \sin u$,
$z = v$



- M^{-1}_{VP}
  - pixel → surface point (x,y,z)
  - (x,y,z) → (u,v): $u = \cos^{-1}(x/r)$, $v = z$

Texture-Mapping: Cylindrical Surface

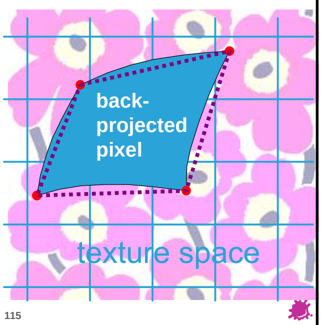
- M_T $\quad u = s \cdot \pi/2, \quad v = t \cdot h$



- M^{-1}_T $\quad s = 2u/\pi, \quad t = v/h$

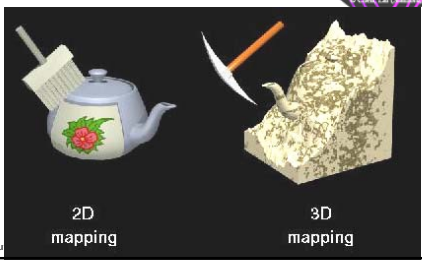
Texture Mapping: Anti-aliasing

- anti-aliasing with filter operations
  - project pixel area into texture space and take average texture value
- speed ups:
  - mip-mapping
  - summed-area table method

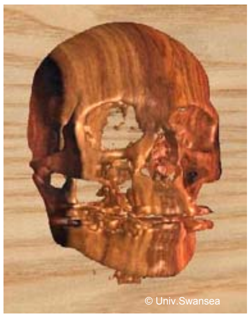


Solid Texturing

- texture defined in 3D
- every position in space has a color
- coherent textures across corners



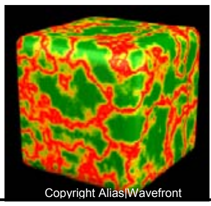
Solid Texturing Examples



examples for application of 3D textures on a scull and a face

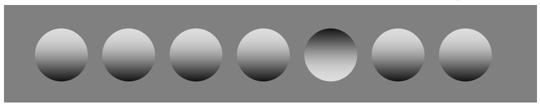
Procedural Texturing

- procedural texture definition
  - texture-function (x,y,z) returns intensity
  - avoid M_T
- 2D (surface texturing) or 3D (solid texturing)
- stochastic variations (noise function)
- examples
  - wood grains
  - marble
  - foam



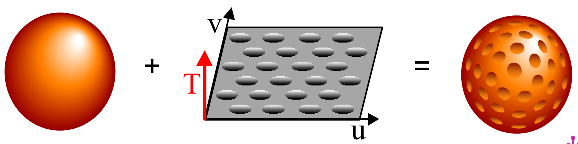
Bump Mapping Principle

bumps are visible because of shading

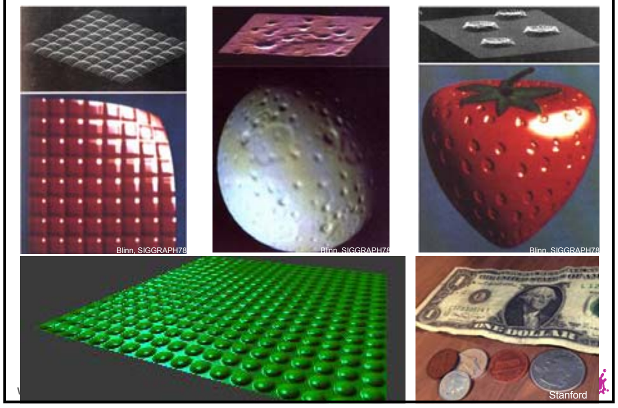


modeling of bumps is very costly.

trick: insert a detail structure T:



Bump Mapping Examples



Bump Mapping Calculation

- surface roughness simulated
  - perturbation function varies surface normal locally
  - bump map b(u,v)

$P(u,v)$ surface point

$N = P_u \times P_v \qquad n = N/|N|$ surface normal

$P'(u,v) = P(u,v) + b(u,v)n$ modified surface point

Bump Mapping Calculation

$$P'(u,v) = P(u,v) + b(u,v)n$$

$$N' = P'_u \times P'_v$$

$$P'_u = \frac{\partial}{\partial u}(P + bn) = P_u + b_u n + b n_u$$

$$P'_u \approx P_u + b_u n, \qquad P'_v \approx P_v + b_v n,$$

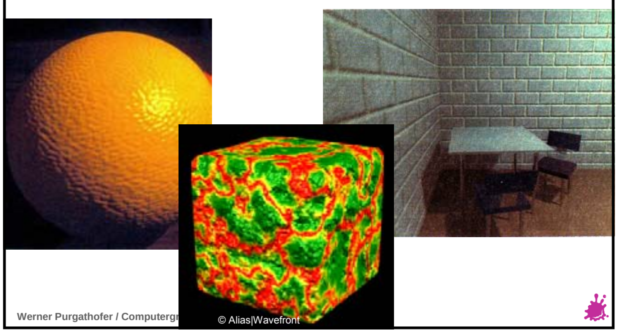
$$N' = P_u \times P_v + b_v(P_u \times n) + b_u(n \times P_v) + b_u b_v(n \times n)$$

$$n \times n = 0$$

$$\mathbf{N' = N + b_v(P_u \times n) + b_u(n \times P_v)}$$

Bump Mapping Representation

- bump map b(u,v) defined as raster image
- b_u, b_v: approximated with finite differences

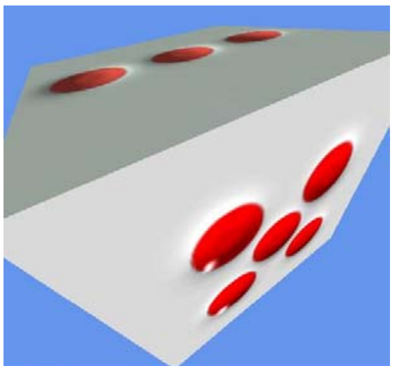


Bump Mapping Problems

- sources of error
  - distortions at grazing angles
  - wrong silhouette (geometry is not changed!)
  - wrong shadows
  - missing shadows of bumps
  - light effects on back side

Bump Mapping: Grazing Angles

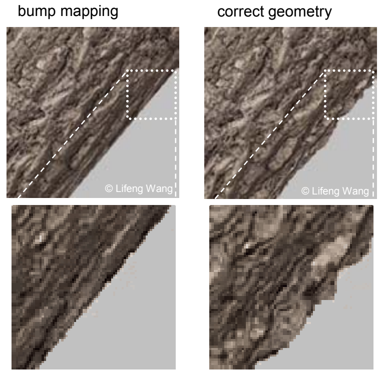
red buttons appear too flat, although they are shaded in 3D



Bump Mapping Problems

- sources of error
  - distortions at grazing angles
  - wrong silhouette (geometry is not changed!)
  - wrong shadows
  - missing shadows of bumps
  - light effects on back side

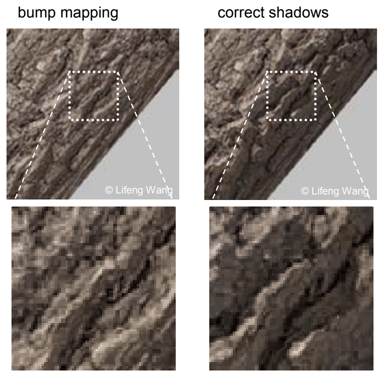
Bump Mapping: Wrong Silhouette



Bump Mapping Problems

- sources of error
  - distortions at grazing angles
  - wrong silhouette (geometry is not changed!)
  - wrong shadows
  - missing shadows of bumps
  - light effects on back side

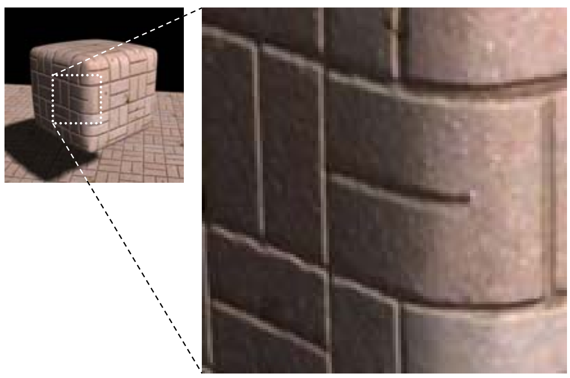
Bump Mapping: Missing Bump Shadows



Bump Mapping Problems

- sources of error
  - distortions at grazing angles
  - wrong silhouette (geometry is not changed!)
  - wrong shadows
  - missing shadows of bumps
  - light effects on back side

Bump Mapping: Back Side Light Effects



Bump Mapping Problems

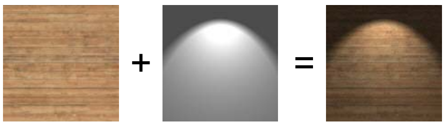
- sources of error
  - distortions at grazing angles
  - wrong silhouette (geometry is not changed!)
  - wrong shadows
  - missing shadows of bumps
  - light effects on back side
- ∃ special algorithms to repair each error

Displacement Mapping

- “correct version of bump mapping”
- surface points are moved from their original position
- outline of object changes
- much harder to implement than bump mapping
  - rare in practice
- latest hardware partially supports it

Multitexturing: Combination of Mappings

- 2 or more textures applied to a surface
- examples:
  - texture + dirt
  - texture + light map
  - texture + bump map
  - photo + annotations
  - …



Multitexturing: Combination of Mappings

bump mapping
& environment mapping
& texture mapping