Chapter 10

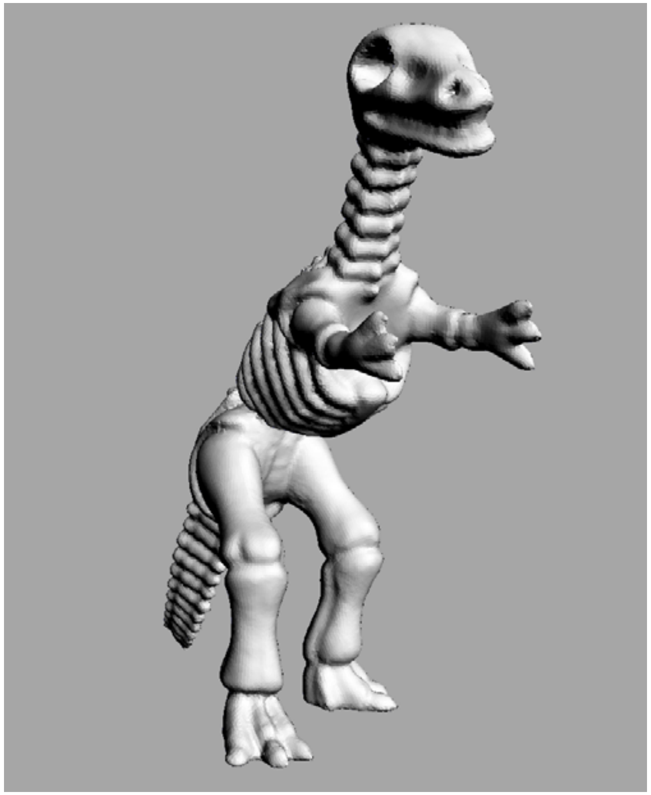
Illumination Models and Surface-Rendering Methods

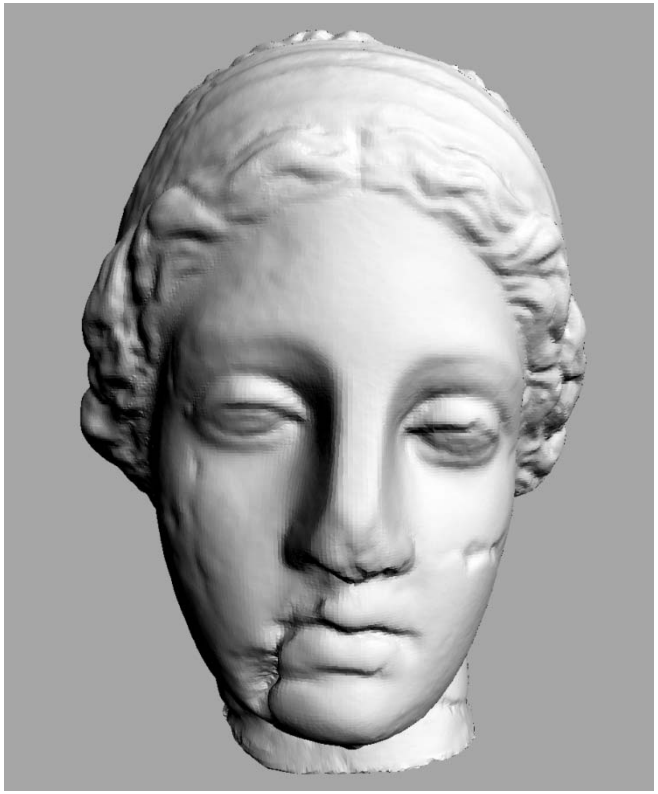
10.1 Overview

For a realistic display of a scene the lighting effects should appear naturally. An **illumination model**, also called a **lighting model**, is used to compute the color of an illuminated position on the surface of an object. The **shader model** then determines when this color information is computed by applying the illumination model to identify the pixel colors for all projected positions in the scene. Different approaches are available that calculate the color for each pixel individually or interpolate between pixels in the vicinity. This chapter will introduce the basic concepts that are necessary for achieving different lighting effects.

10.1 Overview

Using correct lighting improves the three-dimensional impression.





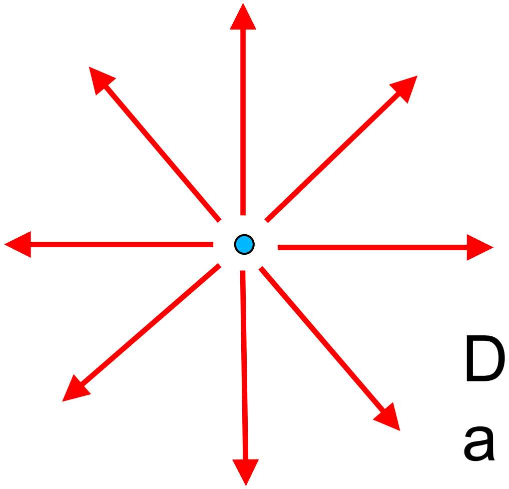
10.1 Overview

In particular, this chapter covers the following:

- Light sources
- Shading models
- Polygon rendering methods
- Ray-tracing methods
- Radiosity lighting model
- Texture mapping
- OpenGL illumination and surface-rendering
- OpenGL texture functions

10.2 Light Sources

Often in computer graphics, light sources are assumed as simple points. This is basically due to the fact that it is easier to do all the necessary lighting computations with point light sources.



Diverging ray paths from a point light source

10.2 Light Sources

Radial intensity attenuation

As the radial intensity from a light source travels outwards its amplitude at any distance d_l from the source is attenuated quadratically:

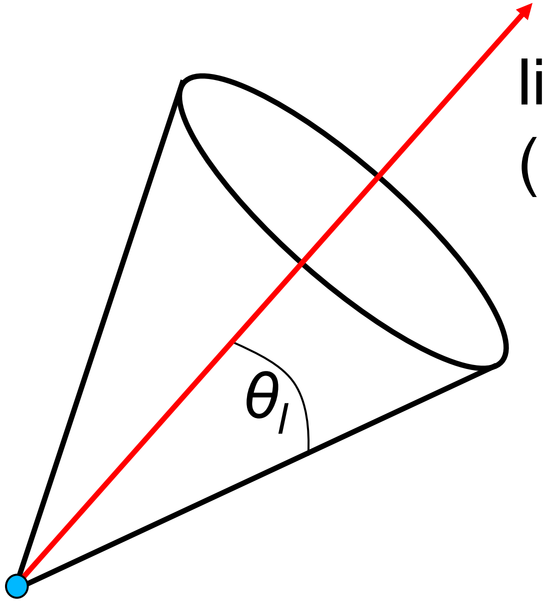
$$f_{RadAtten}(d_l) = \frac{1}{a_0 + a_1 d_l + a_2 d_l^2}$$

For light sources, that are located very far away from the objects of the scene it is safe to assume that there is no attenuation, i.e. $f_{RadAtten} = 1.0$.

10.2 Light Sources

Directional light sources

By limited the direction of the light source, directional light sources, or spotlights, can be created. Then, only objects that divert at a maximal angle θ_l are lit by the light source.

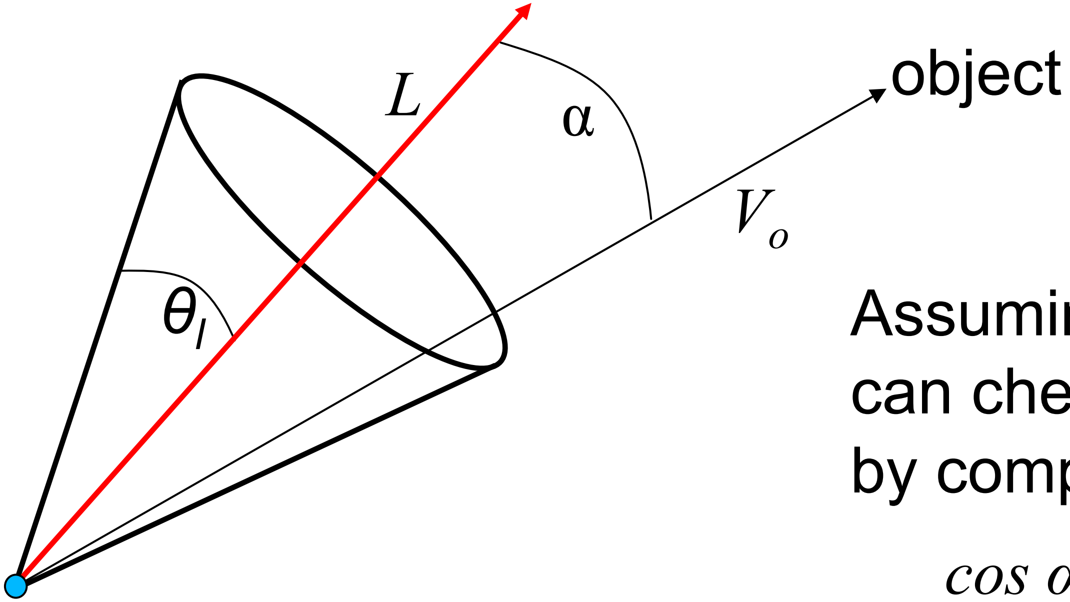


light direction vector L (normalized)

10.2 Light Sources

Directional light sources (continued)

To determine if an object is lit by the light source, we need to compare the angle between the light direction vector and the vector to the object, e.g. its vertices.



Assuming that $0<\theta_l\leq 90°$ we can check if an object is lit by comparing the angles:

$$\cos\alpha = L \cdot V_o \geq \cos\theta_l$$

10.2 Light Sources

Directional light sources (continued)

In analogy to radial attenuation, we can introduce an attenuating effect for spotlights as well based on the angle:

$$f_{AngAtten} = \begin{cases} 1.0 & \text{if source is not a spotlight} \\ 0.0 & \text{object is outside the spotlight} \\ (L \cdot V_o)^{a_l} & \text{otherwise} \end{cases}$$

10.2 Shading model

Illumination model

For determining the intensity (color) of a pixel, which results from the projection of an object (for example a polygon), **illumination models** are used.

An illumination model describes how to compute a intensity (color) of a point within the scene depending on the incoming light from different light sources. The computation is usually done in object space.

In many illumination models, the intensity (color) of a point depends on the incoming direct light from light sources and indirect light approximating reflection of light from surrounding objects.

10.2 Shading model

Shading model

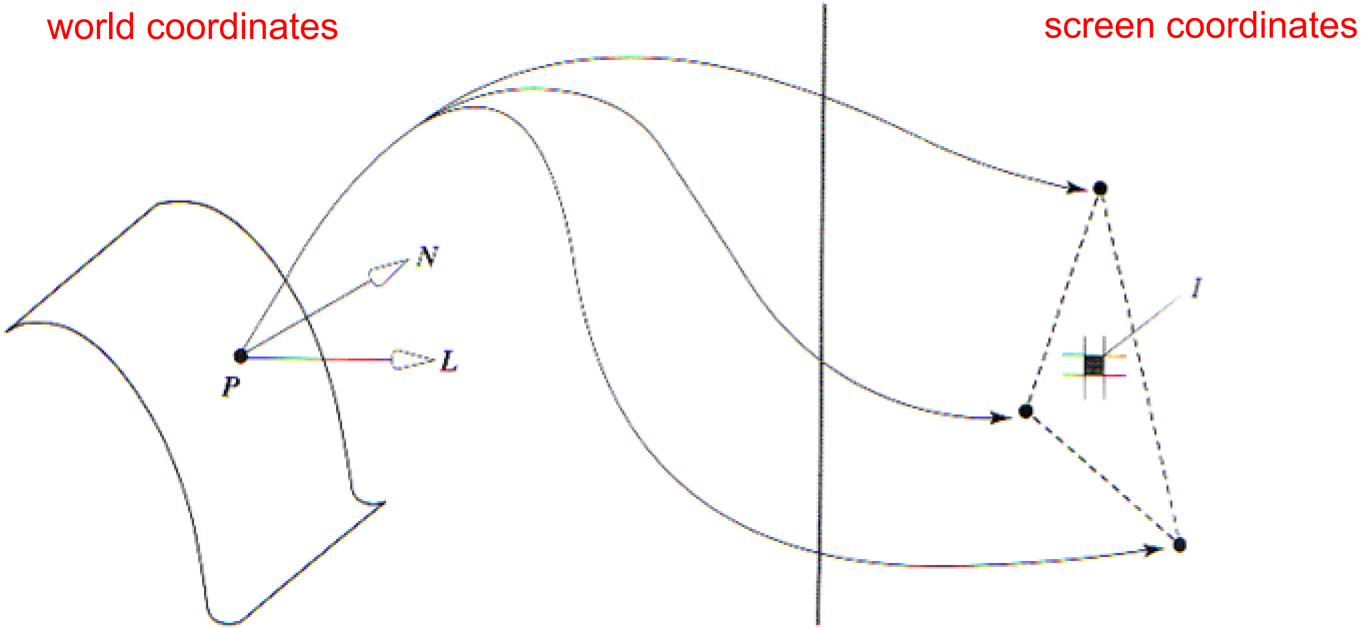
Two different approaches for applying the illumination model are possible to determine all pixels of the resulting image. The illumination model can be applied to

- Each projected position individually
- Certain projected position; the pixels in between are then interpolated

→ interpolating shading techniques, e.g. flat shading, Gouraud shading, or Phong shading

10.2 Shading model

Interpolating shading techniques



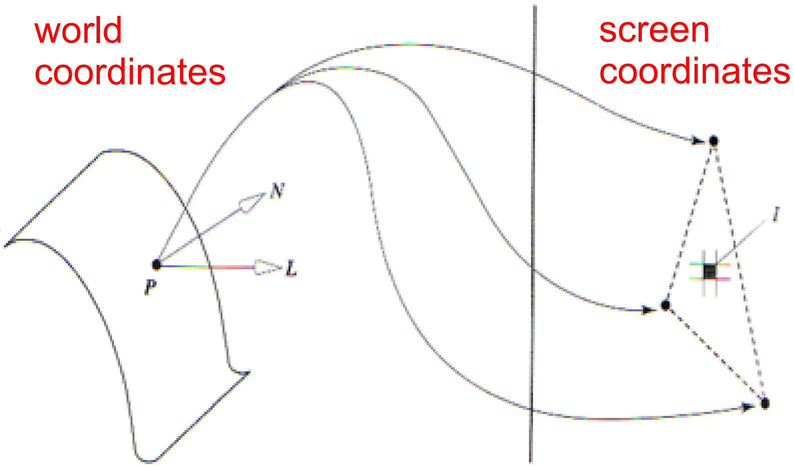
illumination model:
The intensity of a point on a surface of an object is computed

interpolating shading algorithm:
Interpolates pixel intensities by interpolating intensities of poylgon vertices

10.2 Shading model

Interpolating shading techniques (continued)

Does this state a problem?


world coordinates

screen coordinates

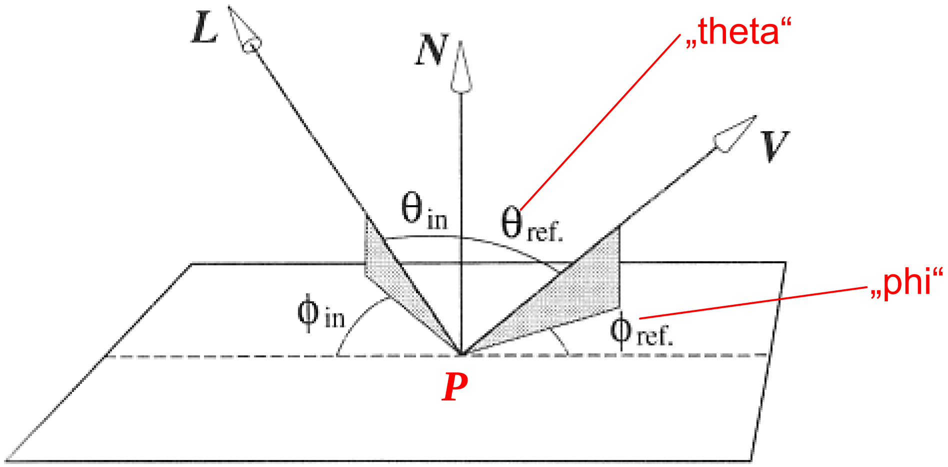
- Lighting of the scene is done in object space (world coordinates)
- Interpolation of intensity values is done in image space
- Projections generally are not affine transformations

→ By using an interpolating scheme (e.g. linear interpolation) we use “incorrect” ratios with respect to the world coordinate system

Despite being mathematically incorrect, this shading model achieves fast and acceptable results

10.2 Shading model

Geometry



P point on the object’s surface

N surface normal vector at P, normalized

L vector pointing from P towards a point light source, normalized

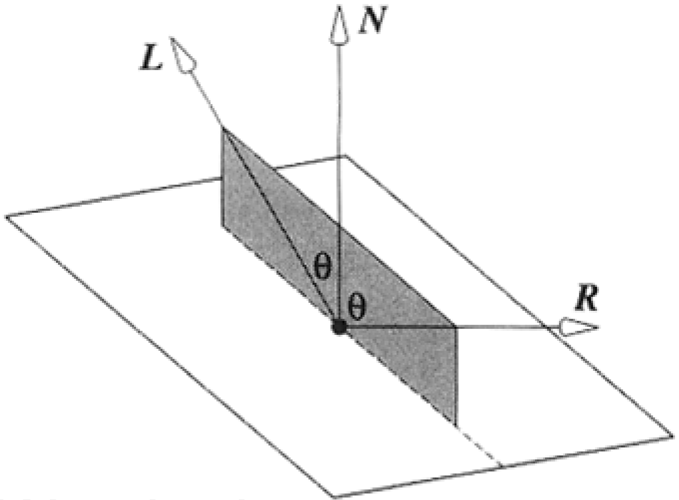
V vector pointing from P towards the view point (eye), normalized

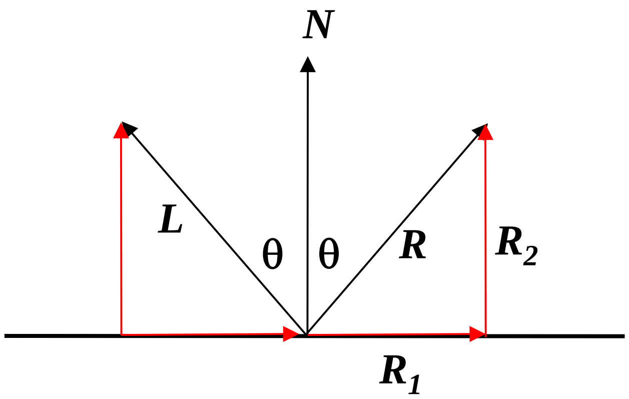
ϕ_i, θ_l (local) spherical coordinates (of L and V)

10.2 Shading model

Specular Reflection

R: vector of the reflected ray, normalized





We obtain:

L and R are located in the same

Plane and $\theta = \theta_{in} = \theta_{ref}$

$$R = R_2 + R_1$$
$$= 2R_2 - L$$
$$= 2(L \cdot N) \cdot N - L$$

10.2 Shading model

We first consider the most common illumination model: the **Phong illumination model**.

Careful: this model is based on empirical results without any physical meaning but with good and practical results!

The model simulates the following physical reflective properties:

a) Perfect/full specular reflection

A light ray is completely reflected without any scattering according to the laws of reflection.

Surface: ideal mirror (does not exist in reality)

10.2 Shading model

Simulated physical reflective properties (continued)

b) Partly specular reflection

The light ray is split up so that a reflective conus occurs with the full specular reflection as its main extent.

Surface: imperfect mirror, rough surface; a surface element is composed of microscopically small ideal mirrors which are leveled slightly differently.

10.2 Shading model

Simulated physical reflective properties (continued)

c) Perfect/full diffuse reflection

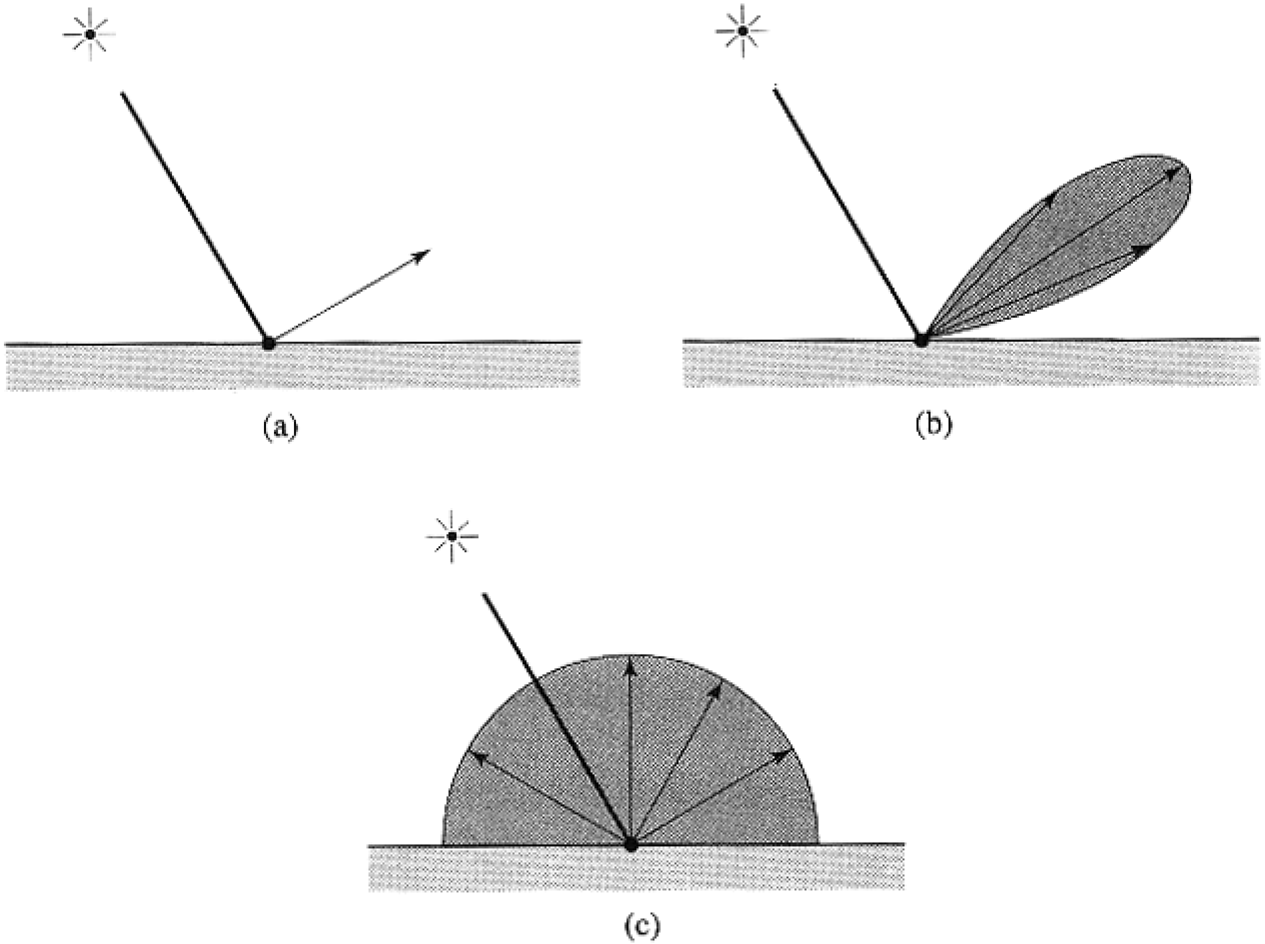
The light ray is perfectly scattered, i.e. with the same intensity in all direction

Surface: ideal, matt surface (does not exist in reality); approximates e.g. fine layer of powder

The Phong illumination model then composes the reflected light linearly out of these three components:

reflected light = diffuse component + specular component
+ ambient light

10.2 Shading model



10.2 Shading model

Ambient light

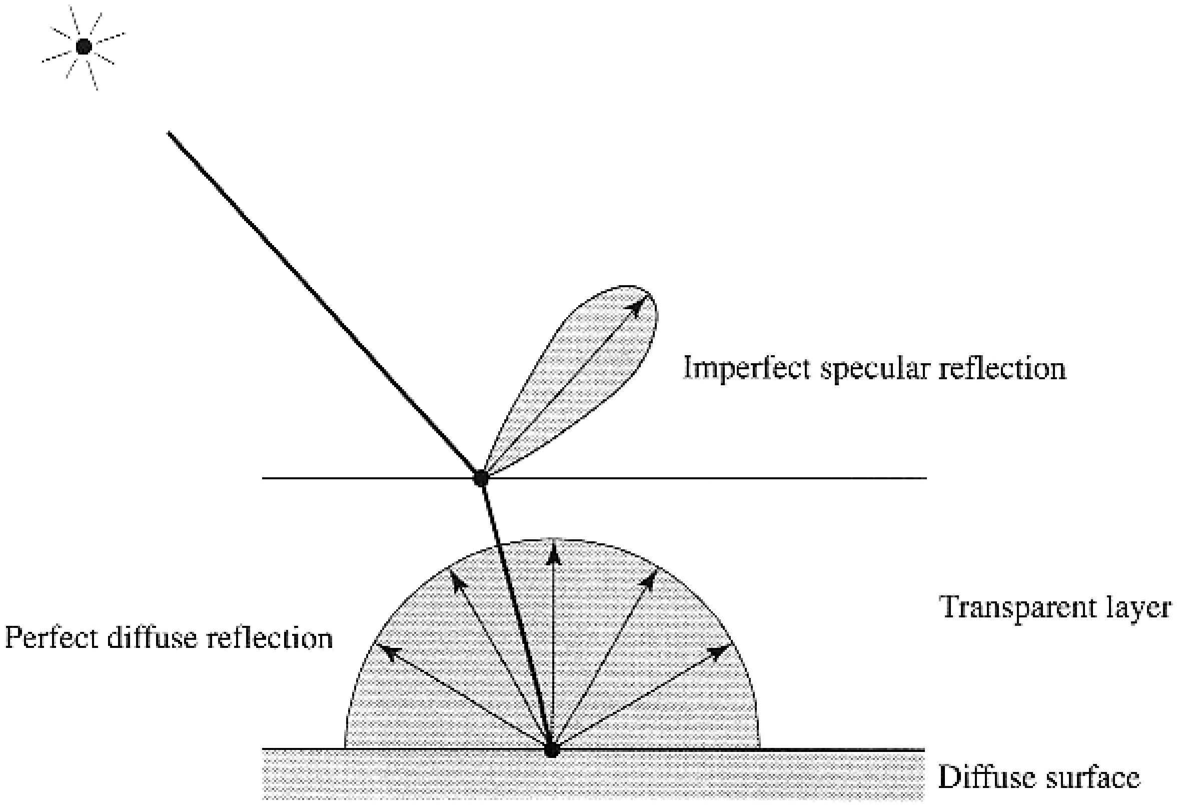
The ambient component is usually defined as constant and simulates the global and indirect lighting. This is necessary because some objects are not hit by any light rays and would end up as being black. In reality, these objects would be lit by indirect light reflected by surrounding objects. Hence, ambient light is just a mean to cope for the missing indirect lighting.

What kind of surfaces can be simulated with this model?

The linear combination of different components (diffuse and specular) resemble, for example, polished surfaces.

10.2 Shading model

Polished surfaces



10.2 Shading model

The mathematical model (without color information)

$$I = k_d I_d + k_s I_s + k_a I_a$$

The physical properties of the surface are determined by the ratios between the different components. The constants always add up to one:

$$k_d + k_s + k_a = 1$$

Diffuse reflection, i.e. the term $k_d I_d$

$I_d = I_i \cdot cos\ (\theta)$ with

I_i intensity of the incoming light

θ angle between normal N and light vector L

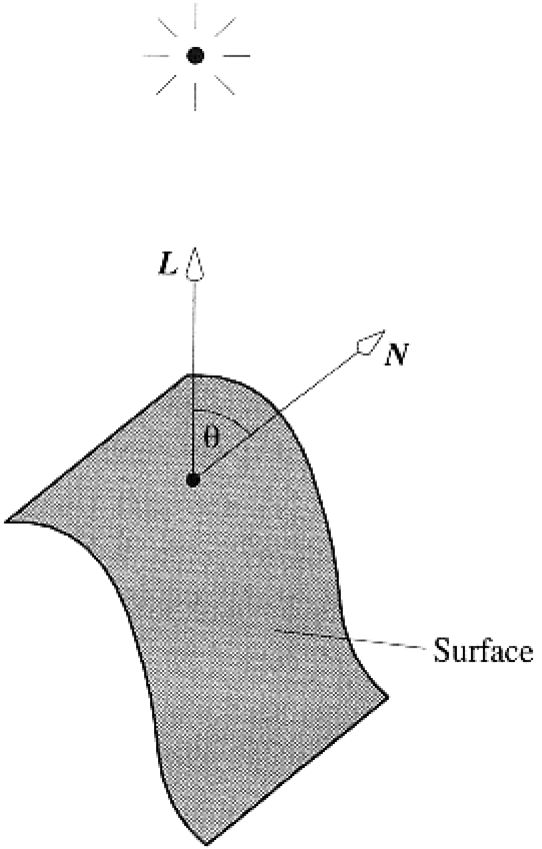
10.2 Shading model

Diffuse reflection (continued)

$$I_d = I_i (L \cdot N)$$

The diffuse component of the Phong model emulates **Lambert's cosine law**:

Ideal diffuse (matt) surfaces reflect the light at an intensity (in all directions equally) identical to the cosine between surface normal and light vector



10.2 Shading model

Specular reflection

From a physical point of view, the specular reflection forms an image of the light source “smeared” across the surface. This is usually called a highlight.

A highlight can only be seen by the viewer if her/his viewing direction V is close to the direction of the reflection R. This can be simulated by:

$I_s = I_i \cos^n (\Omega)$ with

Ω angle between V and R

n simulates degree of perfection of the surface

($n \rightarrow \infty$ simulates a perfect mirror, i.e. only reflects in direction of R)

10.2 Shading model

Specular reflection (continued)

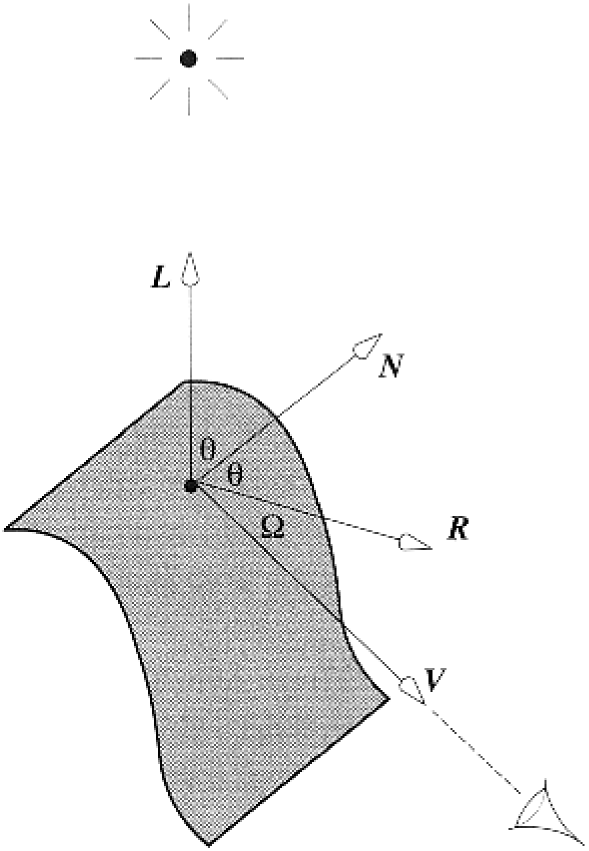
$$I_s = I_i\,(R \cdot V)^n$$

Comments:

For different L we always get (except its direction R) the same reflection cone.

This does **not** concur with the real relation between reflection and direction of the light vector.

Major drawback of this model!

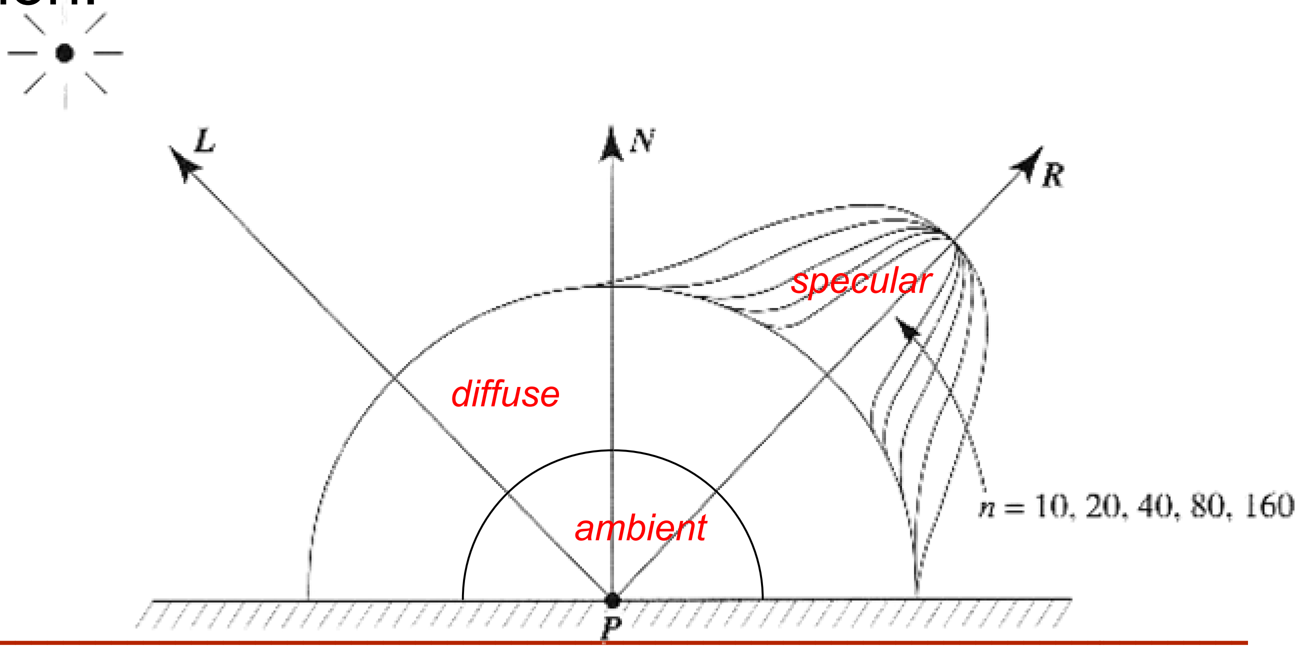


10.2 Shading model

Entire mathematical model in detail:

$$
\begin{aligned}
I &= k_d I_d + k_s I_s + k_a I_a \\
&= I_i \, (k_d \, (L \cdot N) + k_s (R \cdot V)^n) + k_a I_a
\end{aligned}
$$

As a 2-D section:

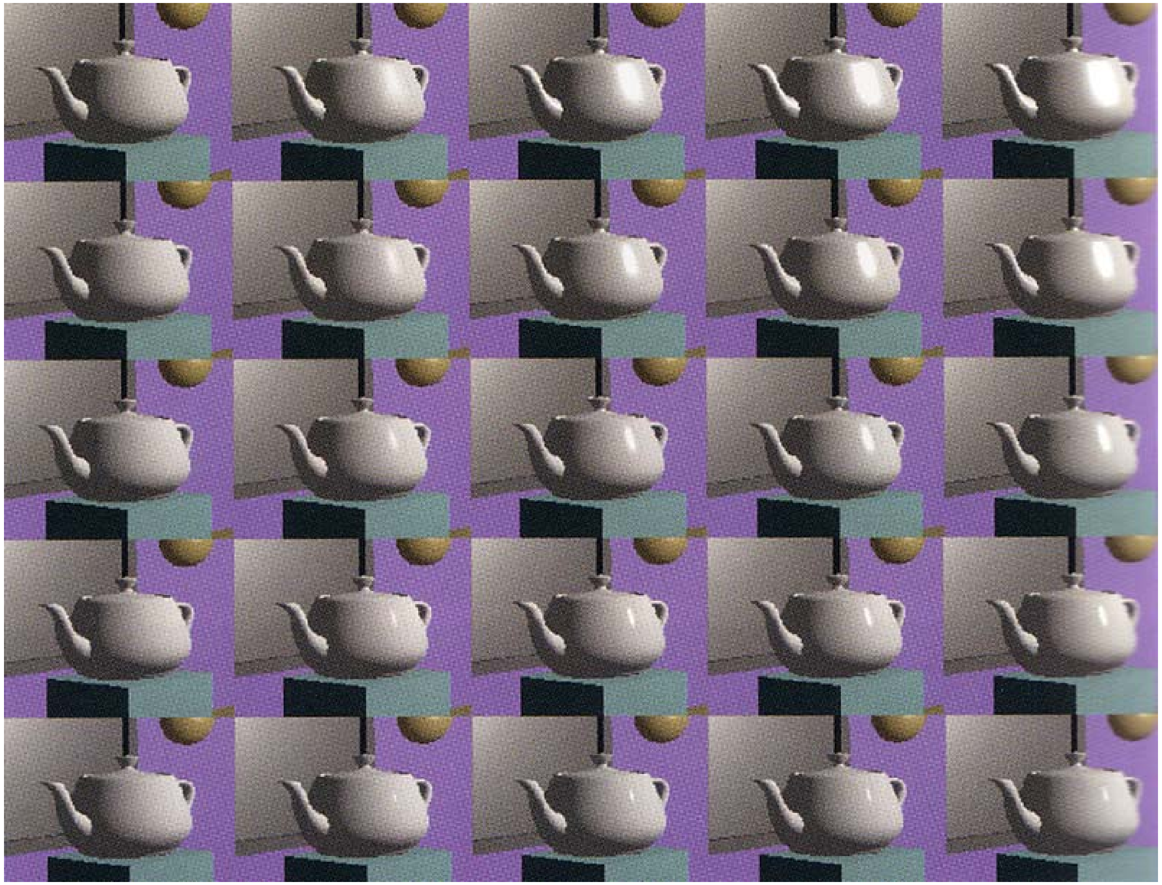


10.2 Shading model

Example:

increasing k_s

k_a,k_d constant



increasing n

10.2 Shading model

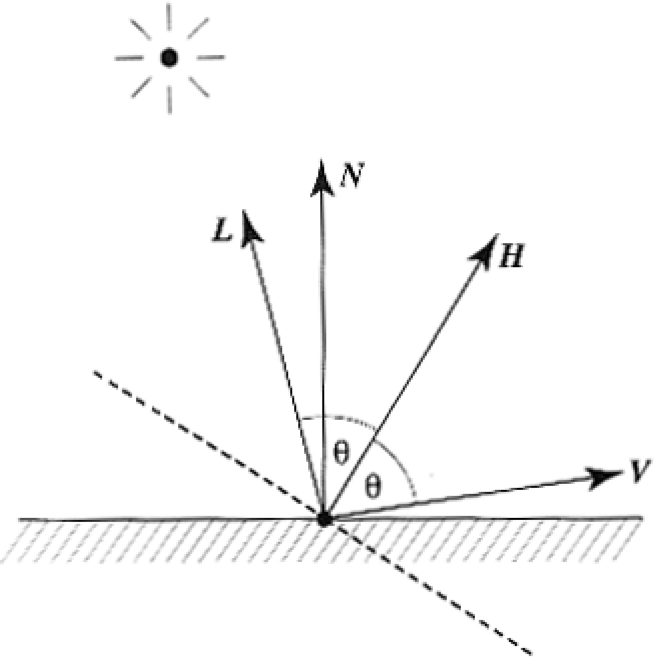
Comment:

If the view point is sufficiently, e.g. infinitely, far away from the light source we can replace the reflection vector R by a constant vector H: $H = (L+V)/\|L+V\|$.

Then we can use $N{\cdot}H$ instead of $R{\cdot}V$, which differs from $R{\cdot}V$ but has "similar" properties.

Hence, we get:

$$I = I_i\,(k_d\,(L \cdot N) + k_s(N{\cdot}H)^n) + k_a I_a$$



10.2 Shading model

The mathematical model: (including color)

$$I_r = I_i\,(k_{dr}\,(L \cdot N) + k_{sr}(N \cdot H)^n) + k_{ar} I_a$$

$$I_g = I_i\,(k_{dg}\,(L \cdot N) + k_{sg}(N \cdot H)^n) + k_{ag} I_a$$

$$I_b = I_i\,(k_{db}\,(L \cdot N) + k_{sb}(N \cdot H)^n) + k_{ab} I_a$$

where

k_{dr}, k_{dg}, k_{db} model the color of the object

k_{sr}, k_{sg}, k_{sb} model the color of the light source

(for white light: $k_{sr,} = k_{sg,} = k_{sb}$

k_{ar}, k_{ag}, k_{ab} model the color of the background light

10.2 Shading model

Comments:

Main deficiencies of the model:

- Two-way reflections and mirroring of surfaces are described insufficiently by the ambient term
- Surfaces appear like plastic, for example, metal cannot be modeled exactly

→ physically based shading models, that try to simulate the BRDFs (reflection function, see next slide) correctly

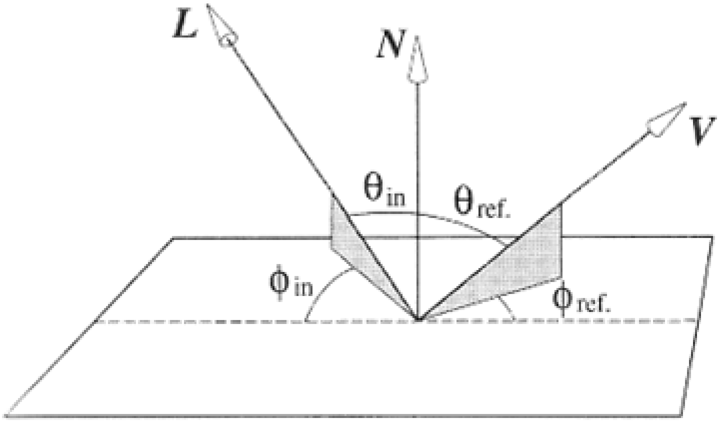
10.2 Shading model

BRDF (bi-directional reflection distribution function):

In general, the reflected light emitted from a point on a surface can be described by a BRDF. The name BRDF specifically stresses the dependence of the light reflected in an arbitrary direction on the direction of the incoming light.

If all directions L and V are known, the correlation between intensities are described by a BRDF:

$$f(\theta_{in}, \Phi_{in}, \theta_{ref}, \Phi_{ref}) = f(L, V)$$



10.2 Shading model

BRDF (continued)

In practice, incoming light enters from more than one direction at a specific point on the surface.

The entire reflected light then has to be computed by integrating across the hemisphere to cover for all possible directions of incoming light.

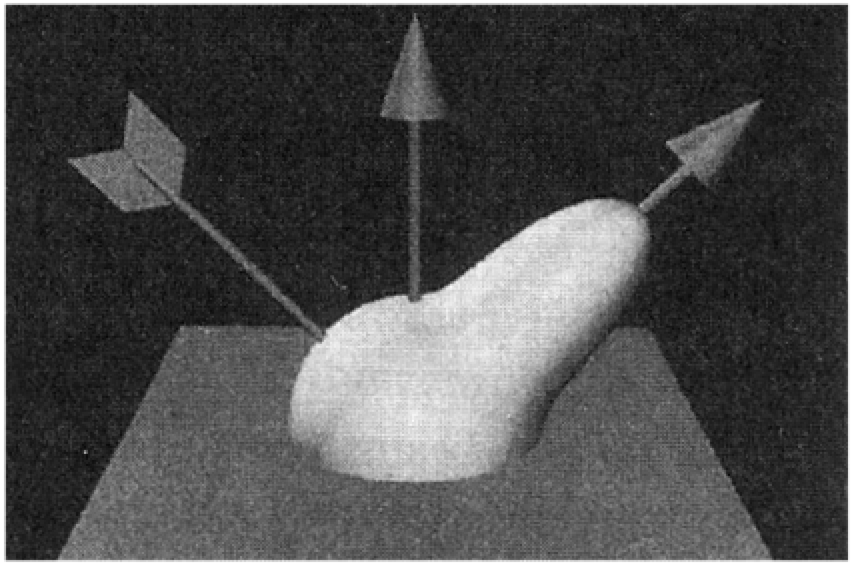
Questions:

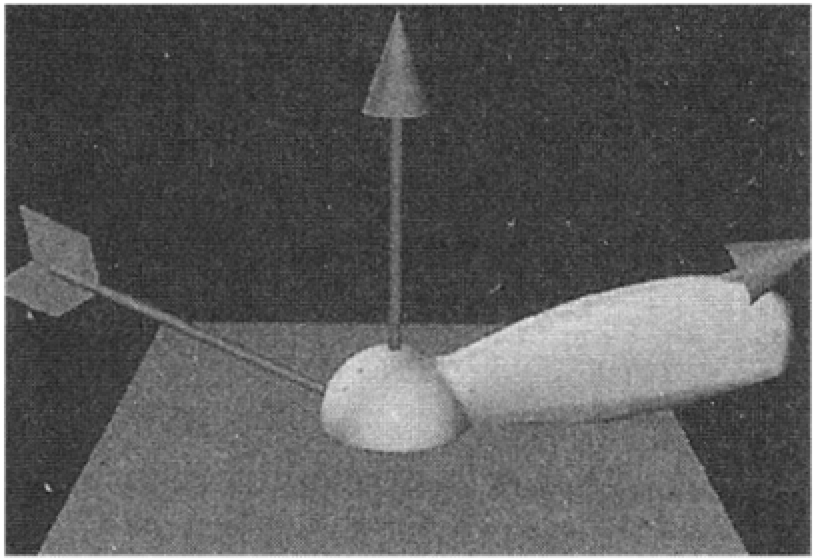
- How to determine BRDFs?
  - → e.g. measuring, modeling
- What resolution is necessary to represent BRDFs?
  - → heuristics if there are no closed form representations
- How can BRDFs be stored and processed efficiently?
  - → e.g. matrices

10.2 Shading model

BRDF (continued)

Representation of BRDFs for two different directions of incoming light (modeled after Blinn (1977):





10.2 Shading model

Disadvantages of “entirely” local shading models:

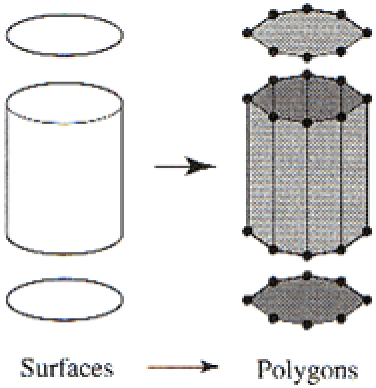
- Represent ideal case of a single object in a scene that is lit by a single point light source
- Only consider direct lighting
- Interaction with other objects is not modeled (i.e. no indirect lighting, no shadows)

→ global shading techniques

10.3 Polygon rendering methods

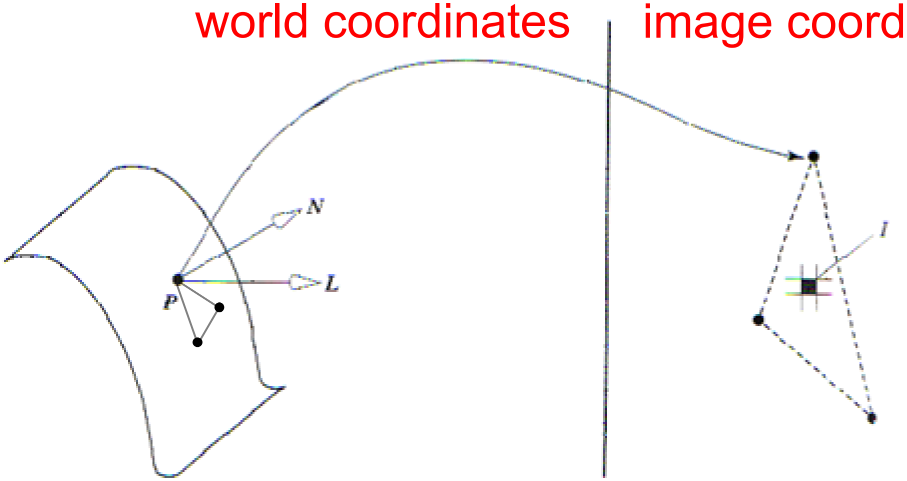
How can the evaluation of a specific illumination model for an object be used to determine the light intensities of the points on the object’s surface?

we assume a polygonal object representation consisting of several faces



world coordinates

image coordinates



There is a difference between (three-dimensional) object space and (two-dimensional) image space!

10.3 Polygon rendering methods

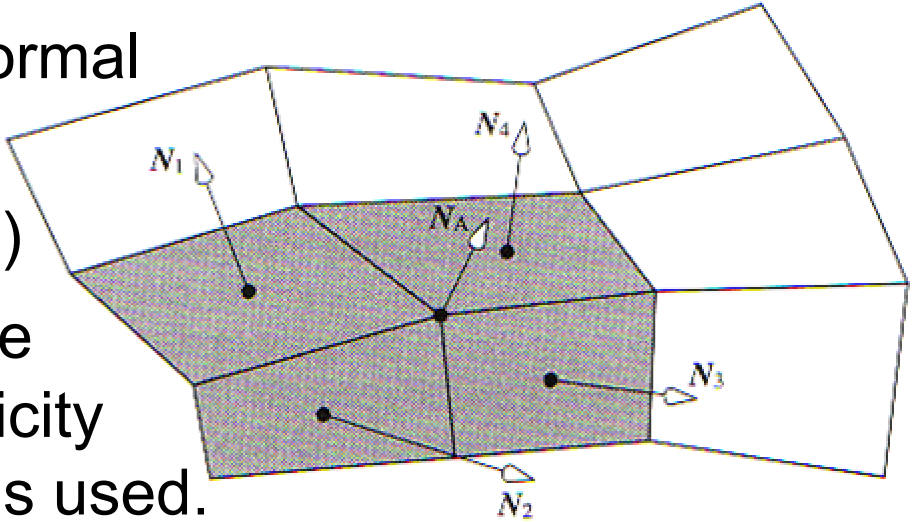
Flat shading

For each polygon/face, the illumination model is applied exactly once for a designated point on the surface. The resulting intensity is then used for all other points on that surface.

The illumination model requires a polygon normal or surface normal in object space.

(for example $N_1, N_2, N_3, N_4, \ldots$)

As designated point, often the center of gravity or, for simplicity reasons, one of the vertices is used.

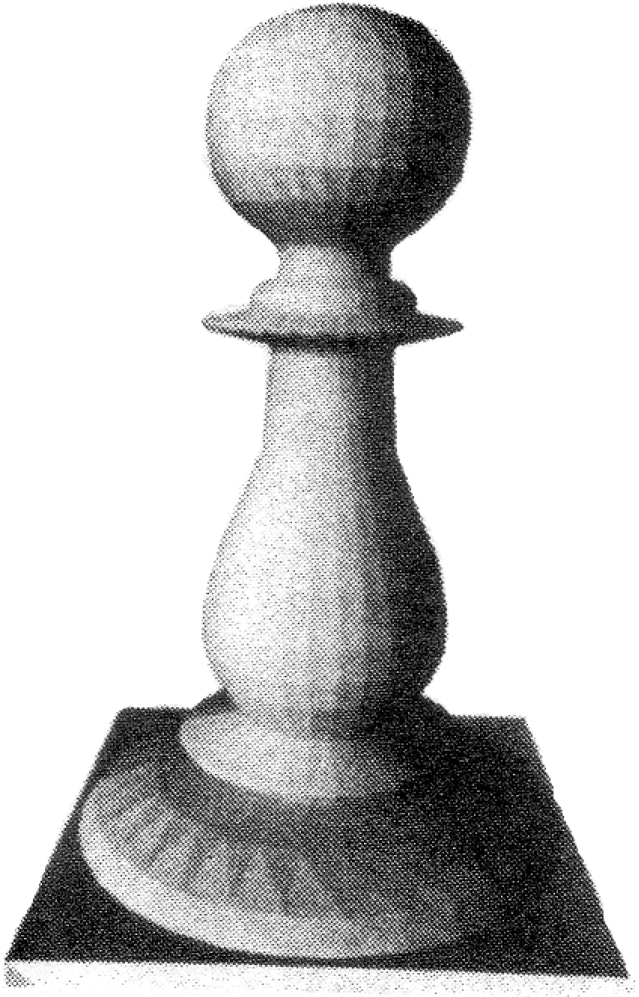


10.3 Polygon rendering methods

Flat shading (continued)

Comments:

- Simple, cost-effective method; interpolation is not necessary
- Edges within polygonal networks remain visible, i.e. faces are visible; non-continuous intensity distribution across edges
- Can be used for pre-views, sketches, visualization of the polygonal resolution.



10.3 Polygon rendering methods

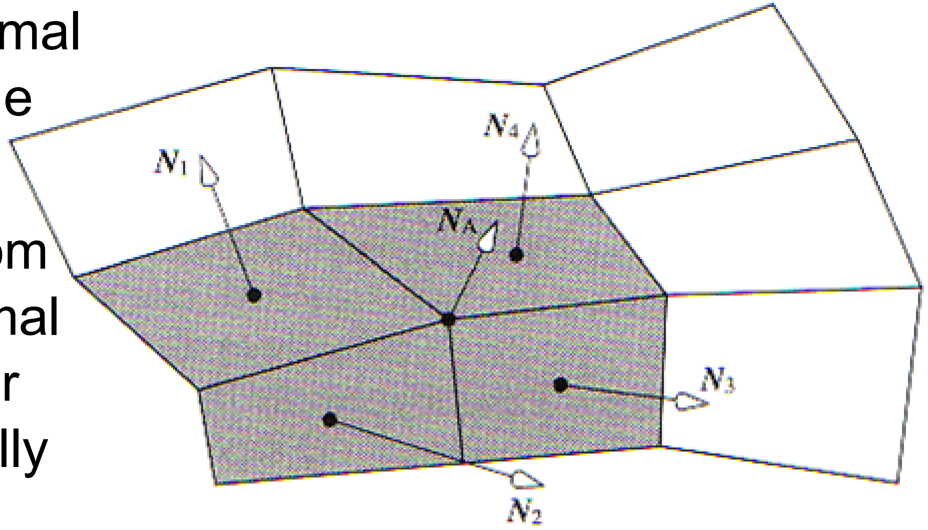
Gouraud and Phong shading

Both methods use interpolation techniques in order to smooth out or eliminate the visibility of the (virtual) edges.

(the polygonal network represents an approximation of a curved surface)

For the illumination model, the normal vectors at the shared vertices of the polygons are used (e.g. N_A,…)

A vertex normal can be derived from the (weighted) average of the normal vectors of the attached polygons or determined from the object originally represented by the polygonal network.



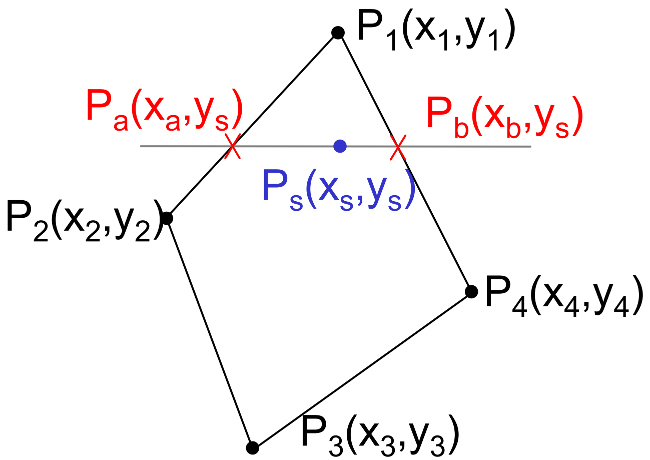
10.3 Polygon rendering methods

Gouraud and Phong shading (continued)

Both methods use (bi-)linear interpolation in the image space:

Values inside (and on the edge of) a polygon are computed from the values at the vertices (generally determined in object space) by using linear interpolation within the image space.

Efficient implementation work incrementally following the scan line.



10.3 Polygon rendering methods

Gouraud and Phong shading (continued)

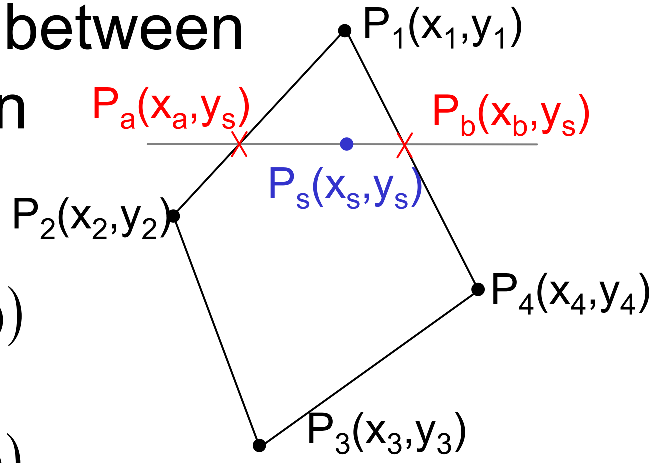
1. Determine values $V(P_1), V(P_2), V(P_3), V(P_4)$
2. Determine the intersections P_a, P_b between scan-line and edges of the polygon
3. Determine $V(P_a)$ and $V(P_b)$:

$$V(P_a) = \frac{1}{y_2 - y_1}\left(V(P_1)(y_2 - y_s) + V(P_2)(y_s - y_1)\right)$$

$$V(P_b) = \frac{1}{y_4 - y_1}\left(V(P_1)(y_4 - y_s) + V(P_4)(y_s - y_1)\right)$$

4. Determine $V(P_s)$:

$$V(P_s) = \frac{1}{x_b - x_a}\left(V(P_a)(x_b - x_s) + V(P_b)(x_s - x_a)\right)$$

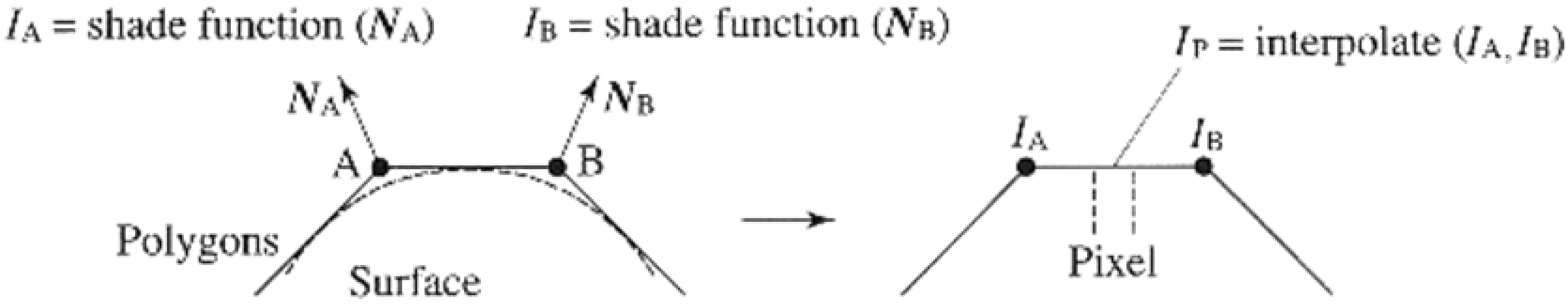


10.3 Polygon rendering methods

Gouraud shading

The illumination model is only used for evaluating the intensities at the vertices of the polygons using the normal vectors of those vertices.

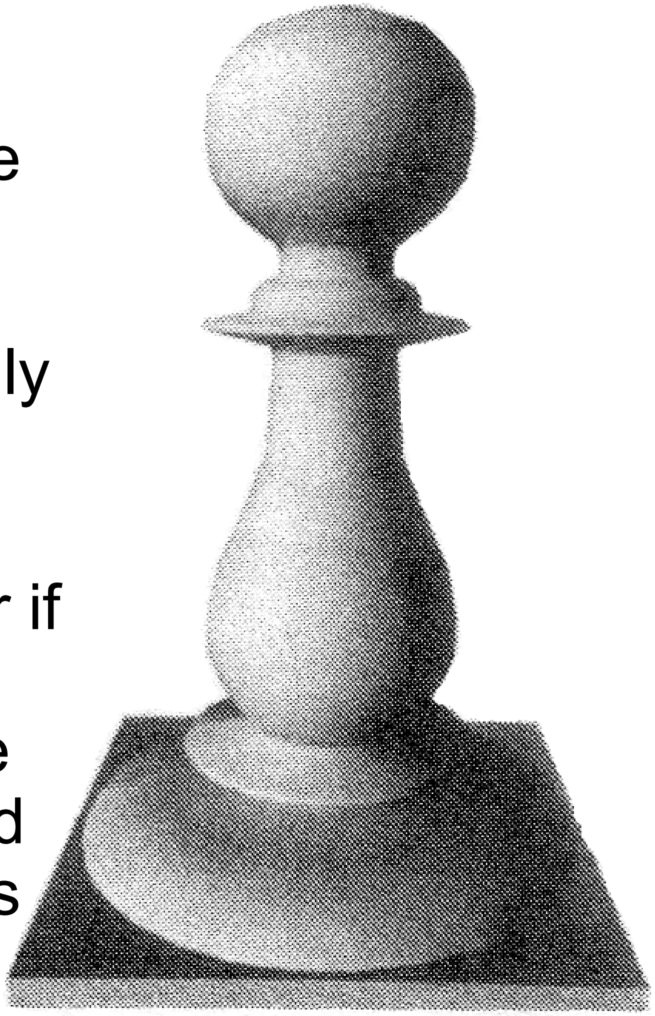
Using interpolation, the intensity values at the (projected) points inside the polygon are computed.


I_A = shade function (N_A)
I_B = shade function (N_B)
I_P = interpolate (I_A, I_B)
N_A
N_B
A
B
Polygons
Surface
I_A
I_B
Pixel

10.3 Polygon rendering methods

Gouraud shading (continued)

- Edges of the polygonal network are smoothed and the intensity distribution across the edge is continuous, however not necessarily smooth.
- Method cannot generate highlights appropriately: these can only occur if the view vector is very close to the direction of reflection; however, the illumination model is only evaluated at the vertices and hence may miss the highlights



10.3 Polygon rendering methods

Gouraud shading (continued)

- Highlights are either skipped or appear shaped like a polygon (instead of round)
- Often used: combination of Gouraud shading and exclusively diffuse reflective component

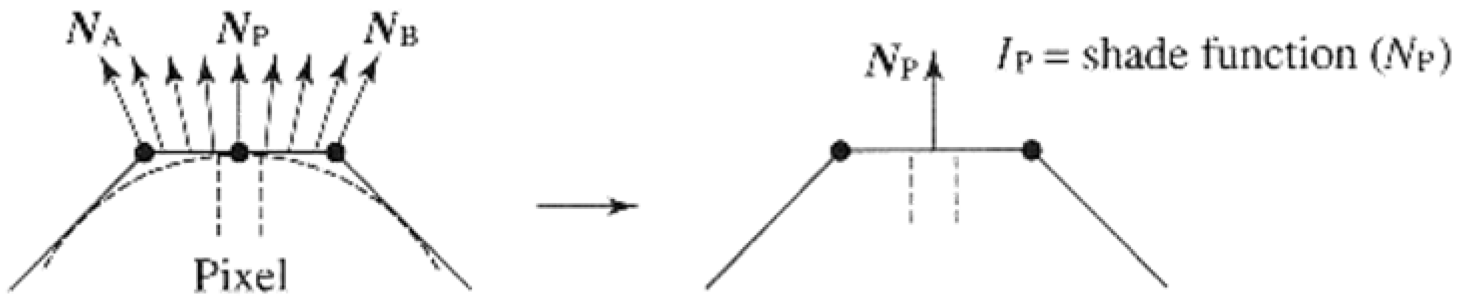
Comment:

Gouraud shading is one of the standard shading methods used by today’s graphics hardware

10.3 Polygon rendering methods

Phong shading

The illumination model is evaluated for every projected point of the polygonal surface. The surface normal at each projected point is computed by interpolating the normals at the vertices.



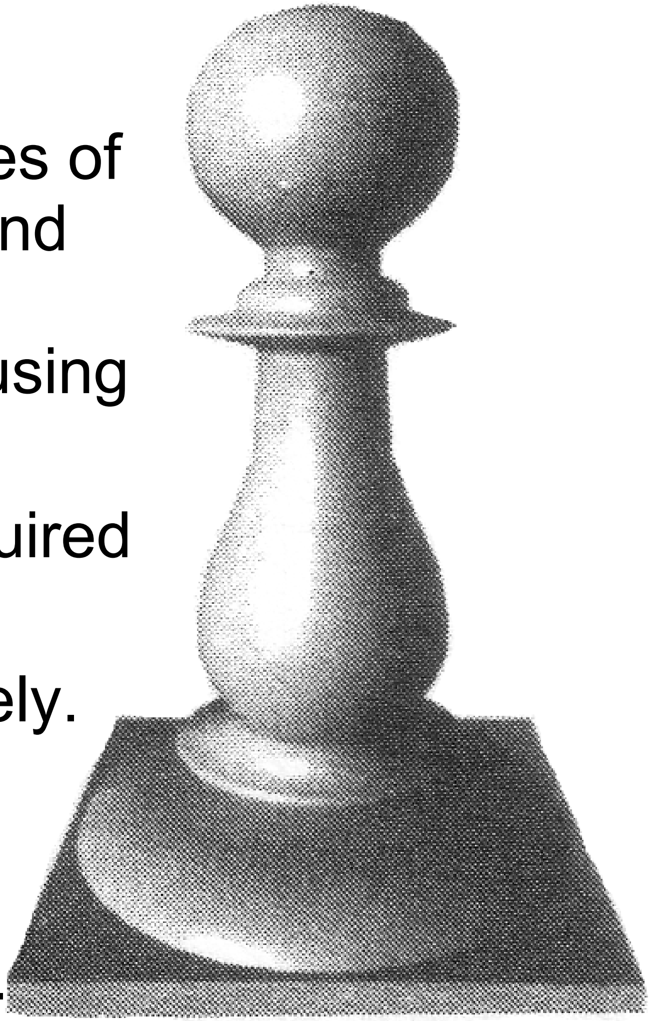
10.3 Polygon rendering methods

Phong shading

- Intensity distribution across the edges of the polygon network is continuous and smooth; the appearance of curved surfaces is approximated very well using the interpolated normal vectors
- Much more computational effort required compared to Gouraud shading.
- Highlights are represented adequately.

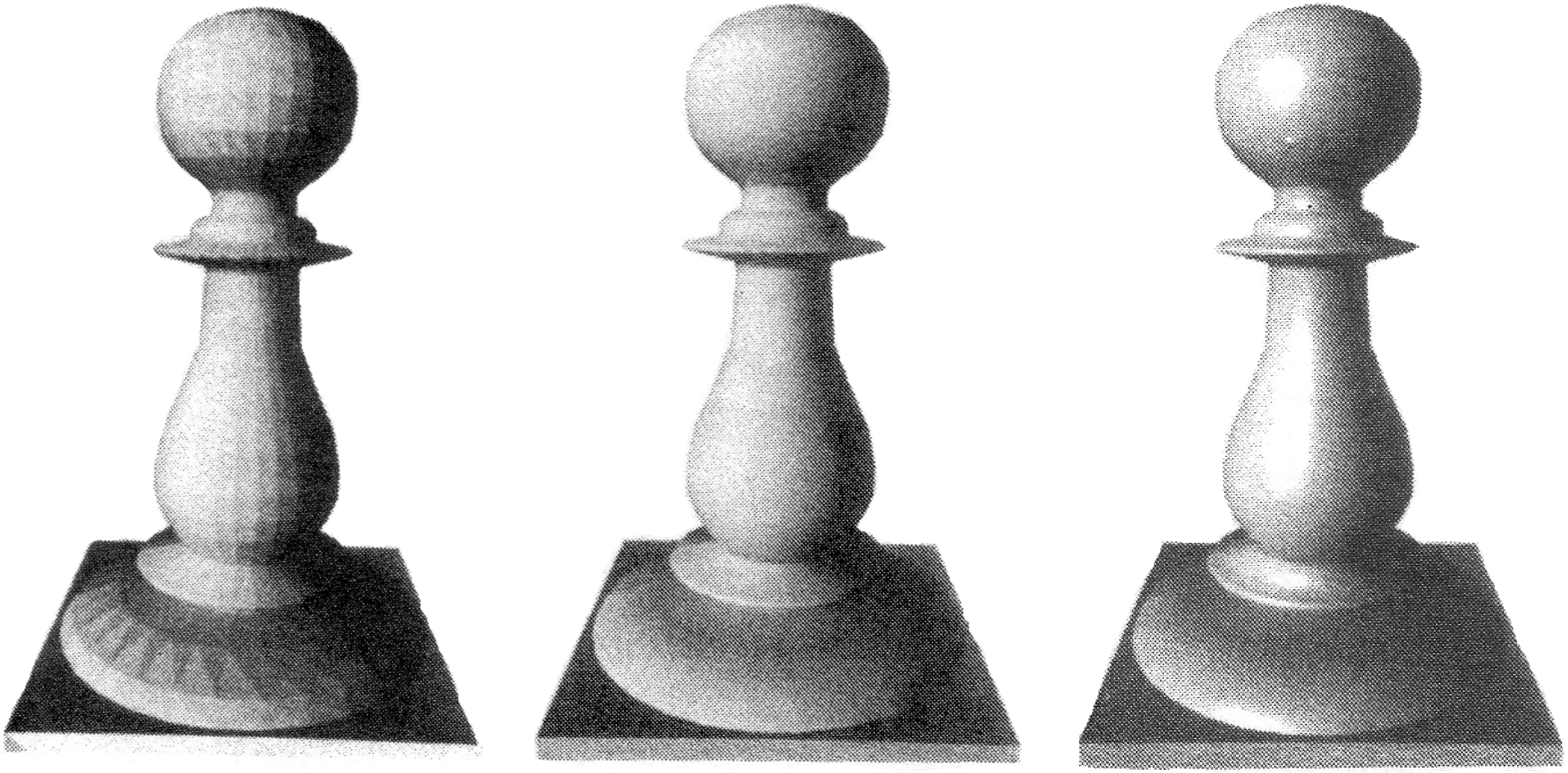
Comment:

Phong shading is supported by current high-end graphics hardware.



10.3 Polygon rendering methods

Flat, Gouraud and Phong shading in comparison



10.3 Polygon rendering methods

Comments:

What do we need to do to ensure that polygon edges that are explicitly supposed to appear as edges when using Gouraud or Phong shading?

- Vertices of the polygon that are part of such feature edges have to be stored separately and with different normal vectors.
- Strong connection and dependence between shading method and polygonization or triangulation of the object (feature recognition)

10.3 Polygon rendering methods

Before the perceived images are transmitted to the brain, the cells in the human eye pre-process the intensity values.

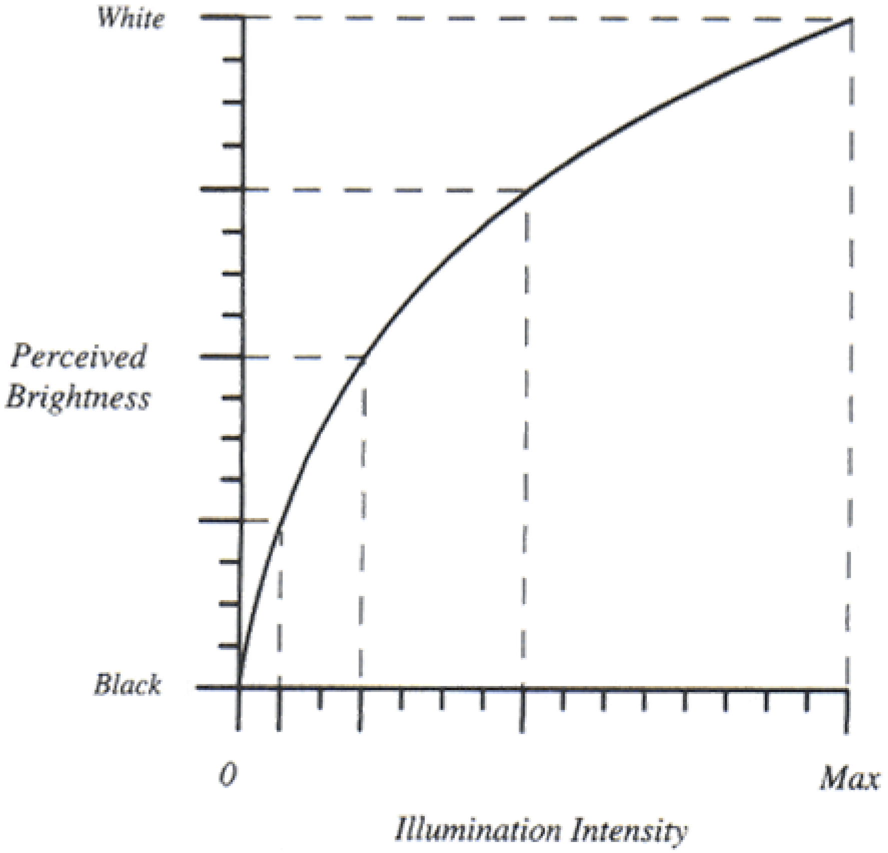
How do the light receptors in the eye react to different light intensities?

Lechner's law

The relation between the light entering the eye and the light intensity perceived by the eye is not linear but approximately logarithmic.

10.3 Polygon rendering methods

Lechner's law (continued)



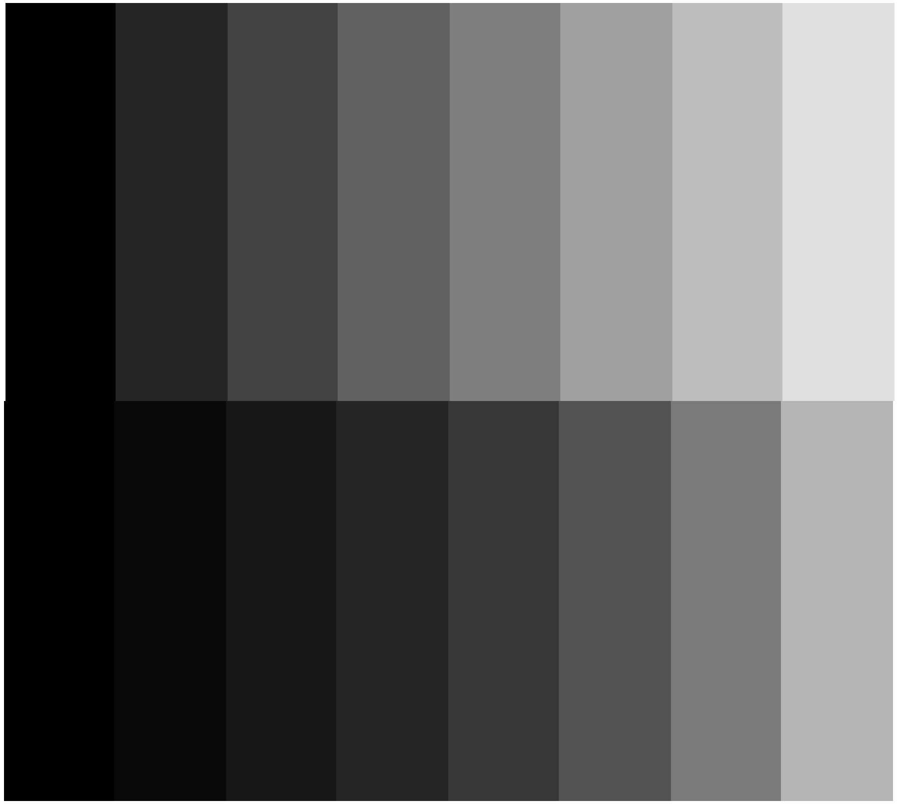
Implication:

Small changes to the intensity in dark areas are perceived better than the exact same change in intensity in brighter areas.

10.3 Polygon rendering methods

Lechner’s law (continued)

Intensity progression / color progression



Intensity increase of equidistant steps of 12.5% with respect to incoming light (0% to 100%

- Jump in intensities in darker areas appears significantly larger than in lighter areas
- Great difference between perceived jumps in intensities

Increase in intensities in equidistant steps with respect to perceived intensities

- Perception of almost equidistant jumps in intensity

10.3 Polygon rendering methods

Mach band effect

The interaction of the light receptors in the human eye emphasize “sharp” changes in intensity.

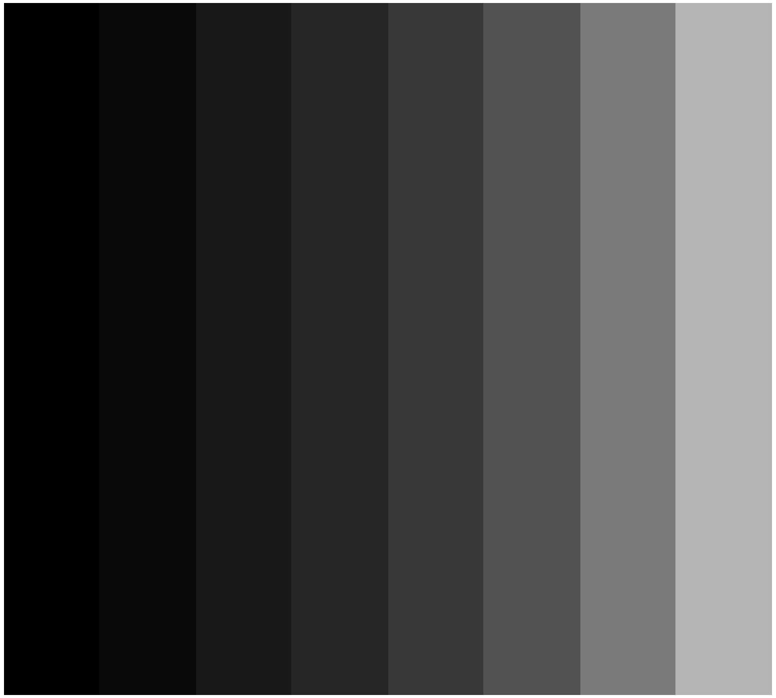
As soon as the eye detects such changes in incoming intensity, it adds overshoot and undershoot to the received intensity which amplify the difference.

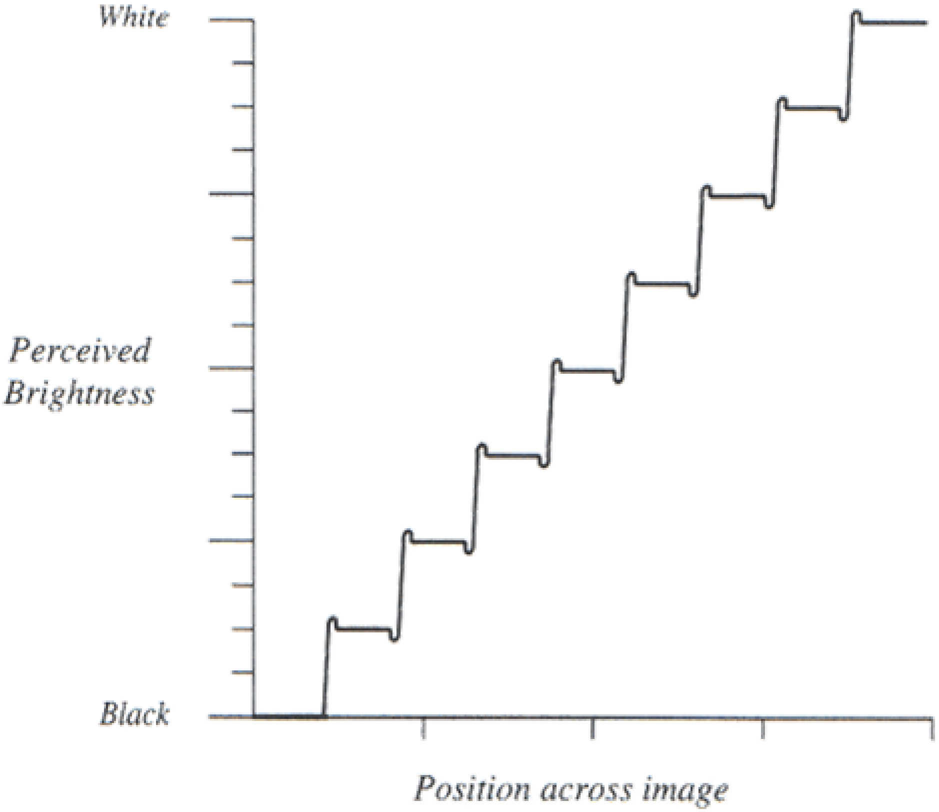
This sub-conscious mechanism of highlighting edges between different intensities helps our visual perception to automatically sharpen contours (edge detection).

10.3 Polygon rendering methods

Mach band effect (continued)

Example:





10.3 Polygon rendering methods

Mach band effect (continued)

When rendering, the automatic detection of changes in intensity of the human eye is rather disturbing and can only be reduced by generating transitions between different intensities as smooth as possible.

Flat shading:

non-continuous transitions in intensity, very strong Mach band effect

Gouraud shading:

continuous change in intensity; nevertheless still Mach band effects depending on the polygonization

Phong shading:

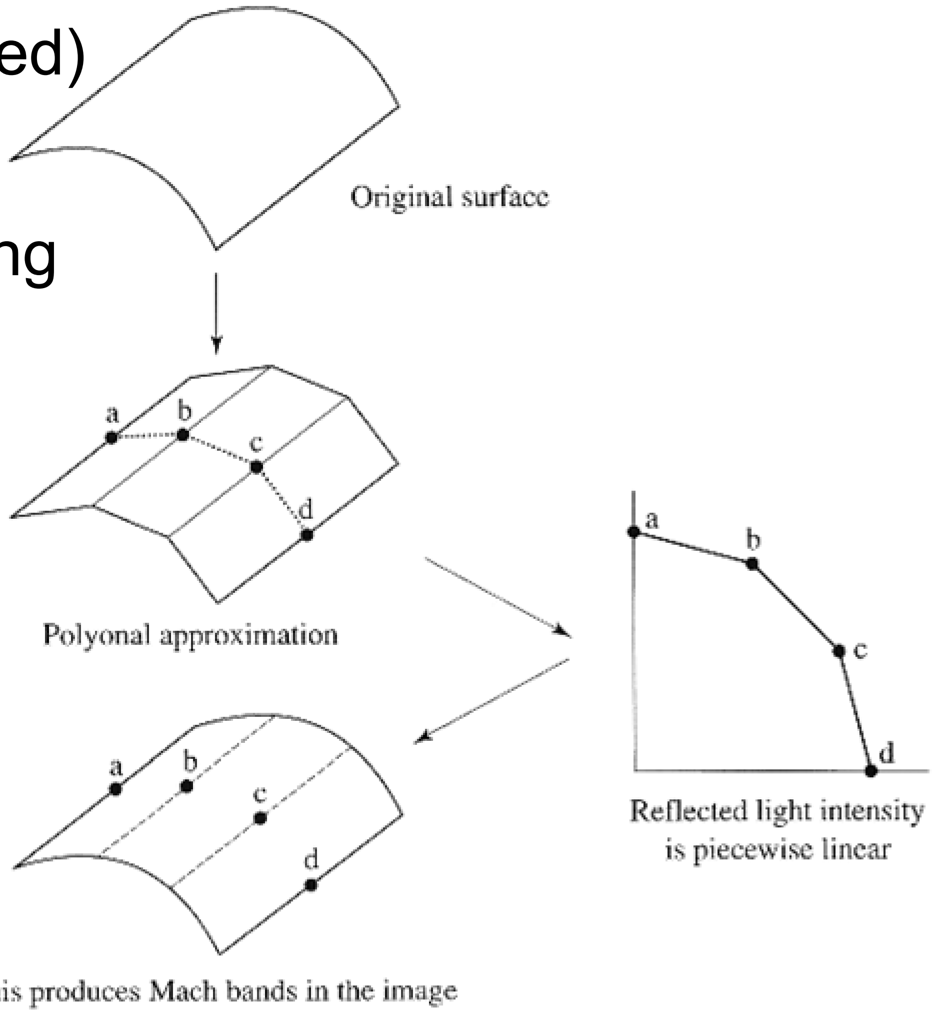
smooth transition between intensities reduce Mach band effect significantly

10.3 Polygon rendering methods

Mach band effect (continued)

Mach band effect occurring

when using Gouraud shading

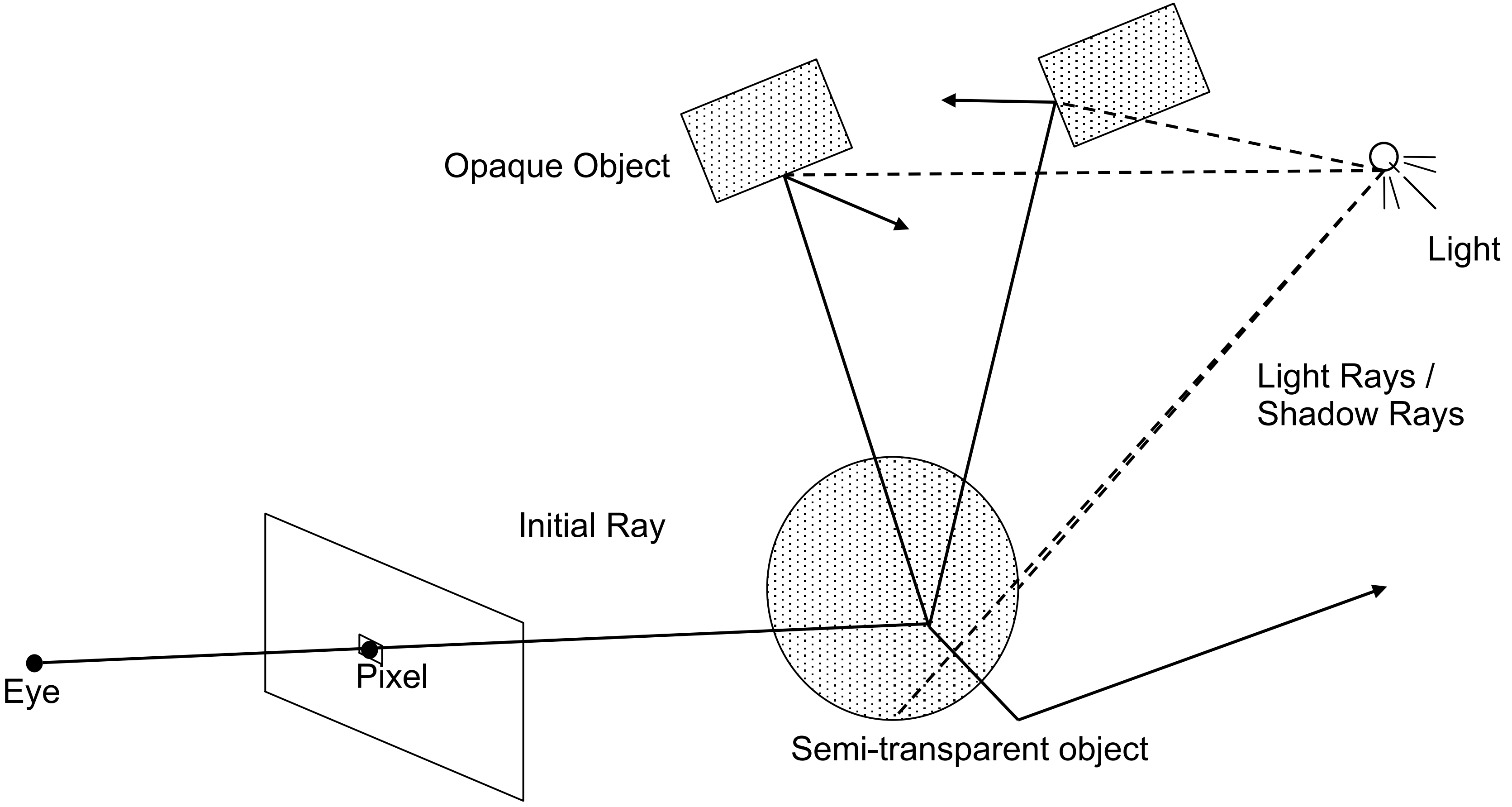

Original surface
a
b
c
d
Polyonal approximation
a
b
c
d
Reflected light intensity is piecewise linear
a
b
c
d
This produces Mach bands in the image

10.4 Ray-tracing

Principle

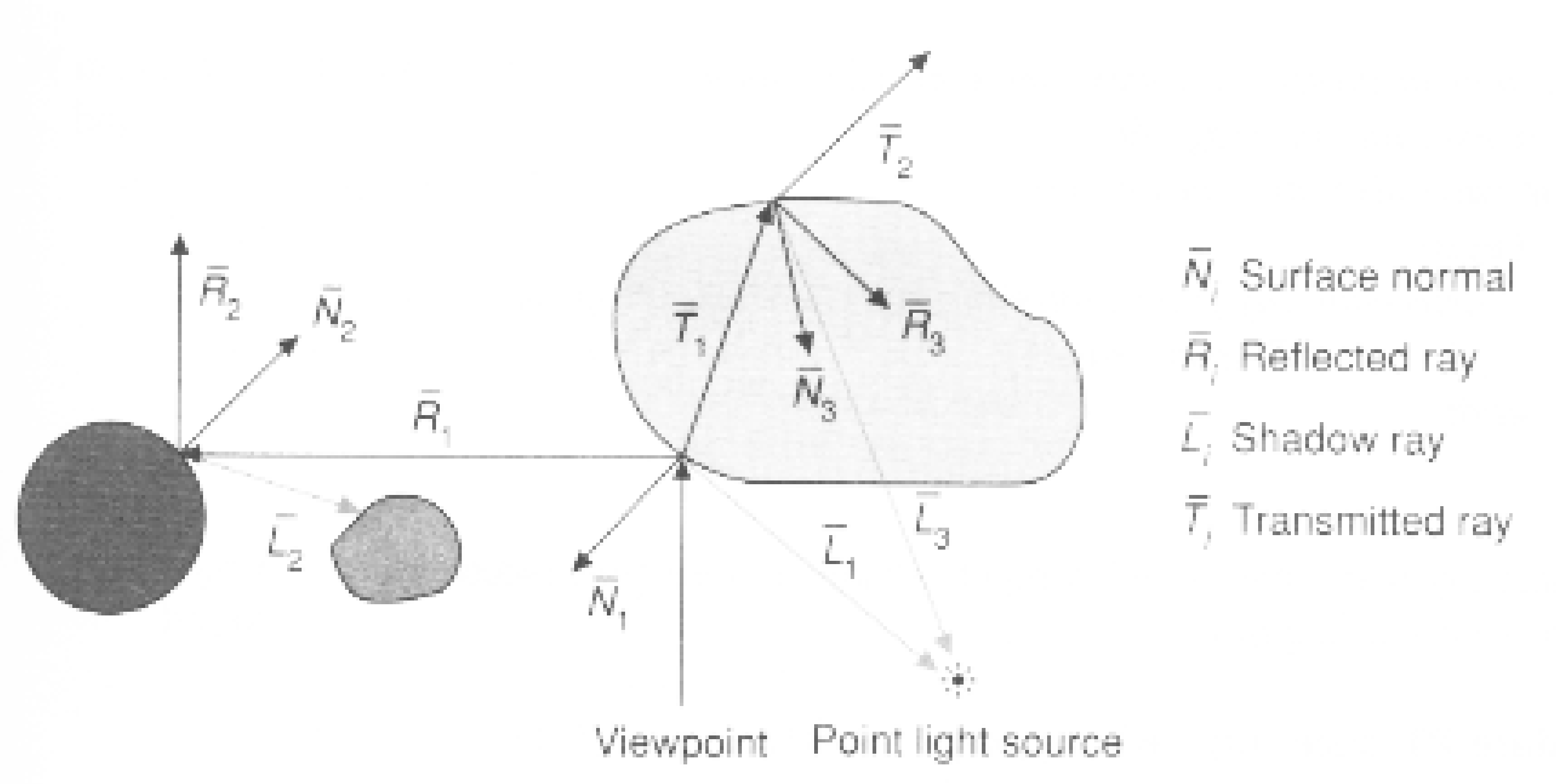
- Backward ray-tracing: since most light rays do not hit the eye it requires less computational effort to trace the light rays backwards starting at the eye and then end at the individual light sources and other surfaces.
- The rays are traced for every pixel within the image place into the scene; at every intersection with an object, the direct (local model) as well as the reflected and refracted light components are determined.
- The resulting bifurcations of the rays implicitly describe a tree.
- Ray-tracing is specifically suitable for modeling of directed light (many mirror-like or transparent objects)

10.4 Ray-tracing



10.4 Ray-tracing

Recursively tracing of rays

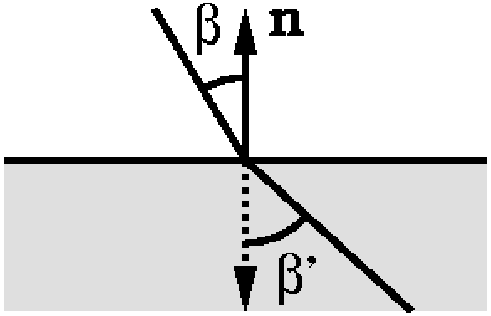


10.4 Ray-tracing

Recursively tracing rays (continued)

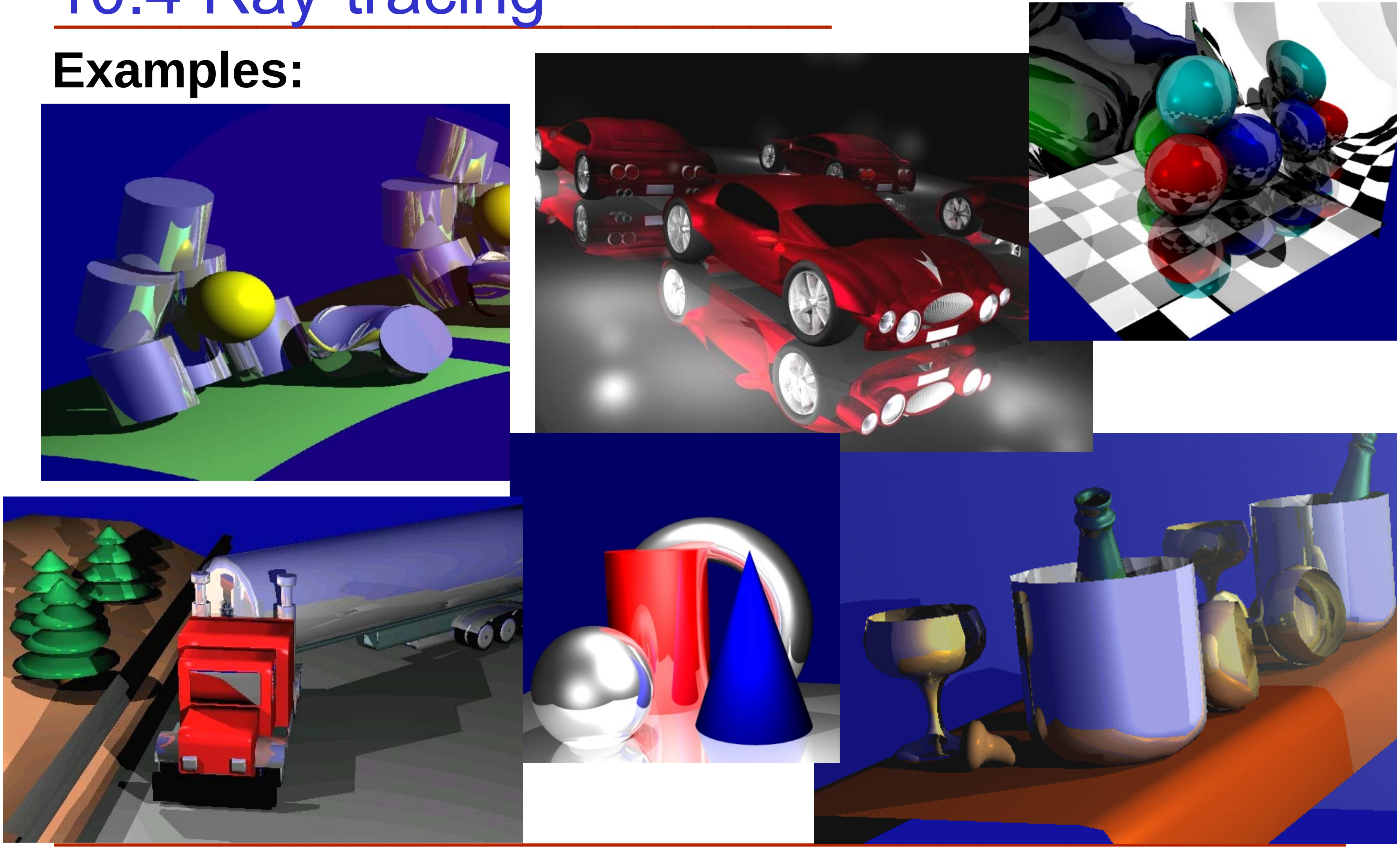
Refraction occurs at the common boundary between media with different densities, e.g. air and water. The angle of refraction β’ is proportional to the angle of incidence β. If the ray enters a more dense medium the angle is going to decrease.

Surfaces can (partly) reflect and refract at the same time. However, if the angle of refraction β’ is greater than 90° the ray is only reflected.



10.4 Ray-tracing

Examples:



10.4 Ray-tracing

Recursively tracing rays (continued)

The ray is “terminated” if

- The reflected or refracted ray does not intersect any more objects.
- A maximal depth of the tree is reached.
- The contribution in intensity (color) value of a continued ray becomes too low.

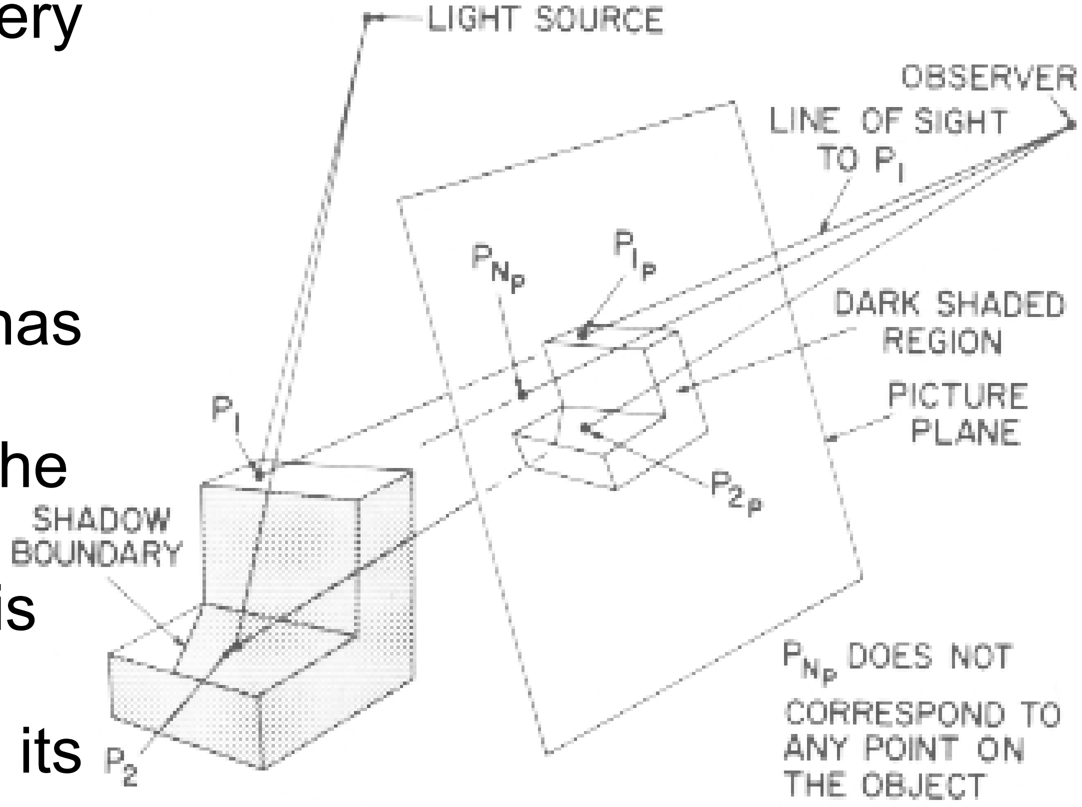
Comment:

The computational effort of this approach depends heavily on the complexity of the scene. Space subdivision techniques, such as octrees, can result in significant speed-ups when using ray-tracing.

10.4 Ray-tracing

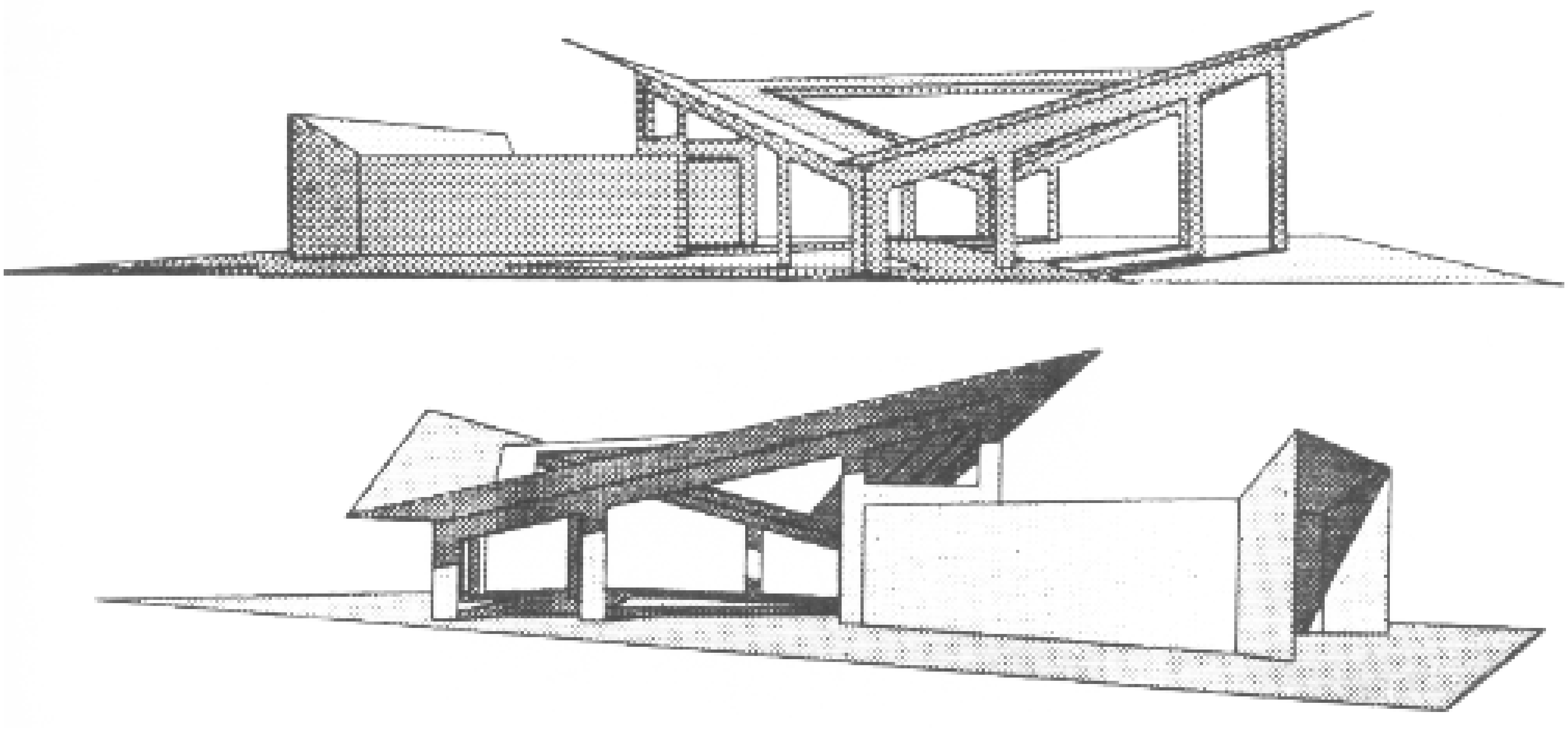
Shadows

- Follow a ray from every intersection with an object to all light sources.
- If one of those rays has to pass through an object on its way to the light source then the point of intersection is not lit by this light source, hence it is in its shadow.



10.4 Ray-tracing

Shadows (continued)

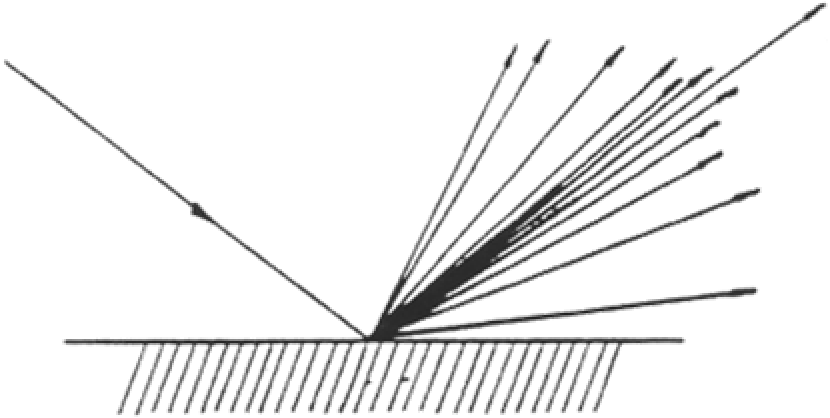


10.4 Ray-tracing

Distributed ray-tracing

In reality, there is no perfect mirror, since no mirror is absolutely planar and reflects purely (100%).

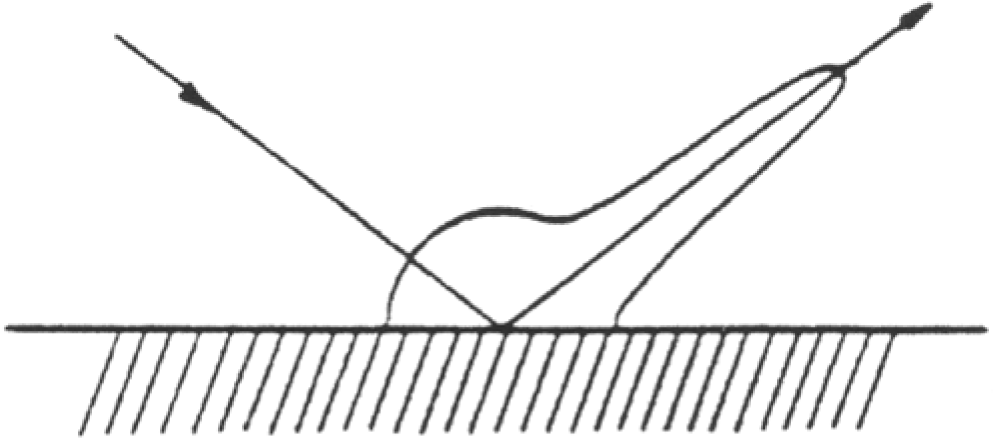
Distributed ray-tracing allows the generation of realistic blur effects for ray-tracing. Instead of using a single reflected ray several rays are traced and the resulting intensities averaged.



10.4 Ray-tracing

Distributed ray-tracing (continued)

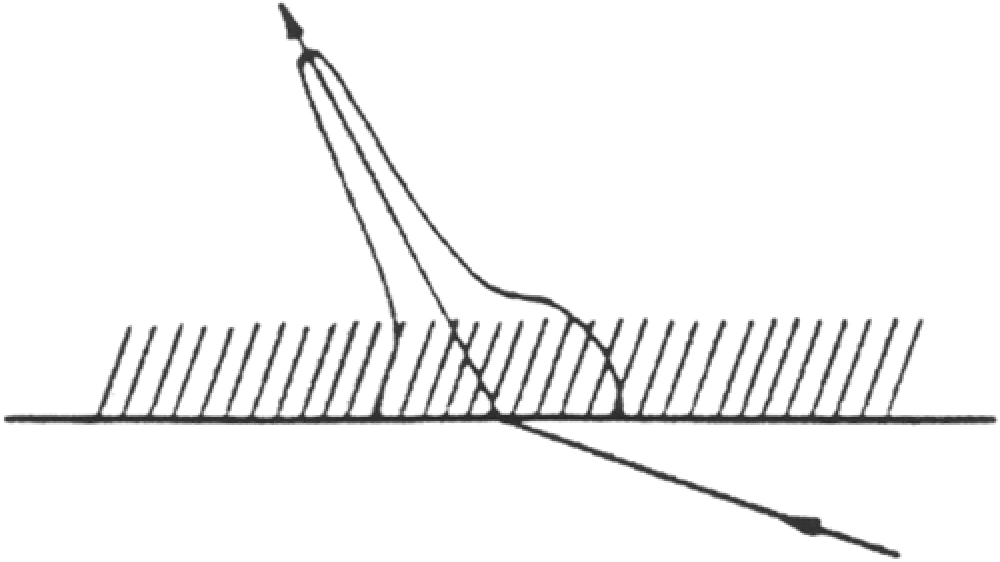
Most of those rays follow more or less the reflected direction while a few brake with this behavior resulting in a bulb-shaped distribution.



10.4 Ray-tracing

Distributed ray-tracing (continued)

Similarly, a refracted ray is spread out using the same pattern.

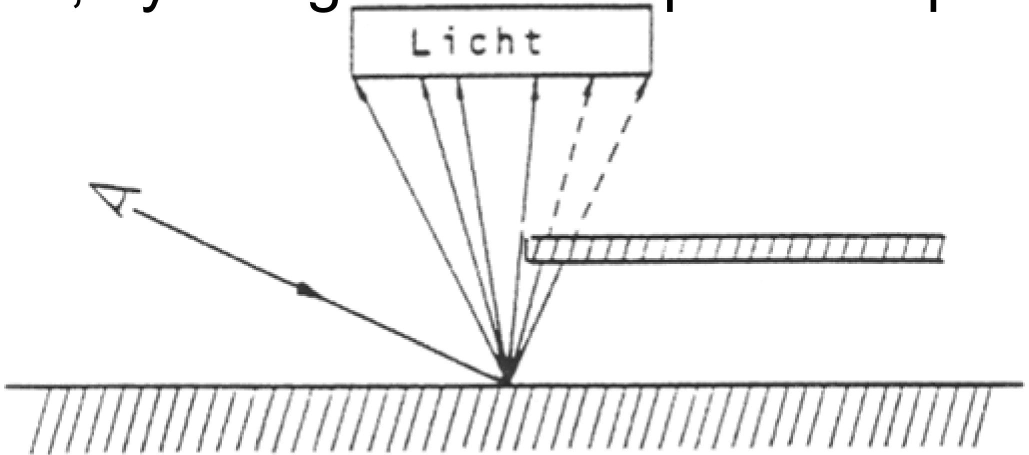


Using a stochastic distribution across all possible directions of reflection and refraction and then averaging, we get a realistic approximation.

10.4 Ray-tracing

Distributed ray-tracing – larger light sources

An additional increase in “realism” can be achieved by allowing light sources not to be just point-shaped. A larger light emitting surface area can be modeled, for example, by using numerous point-shaped light sources.

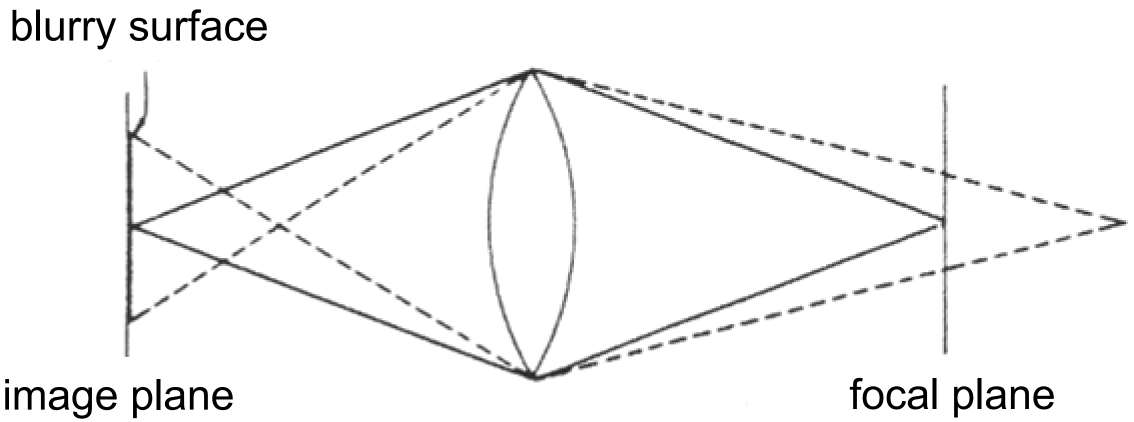


By deploying suitable stochastic distributions of rays, realistic soft-shadows can be achieved.

10.4 Ray-tracing

Distributed ray-tracing – modeling apertures

Photorealistic images result from simulation of the aperture of a camera.

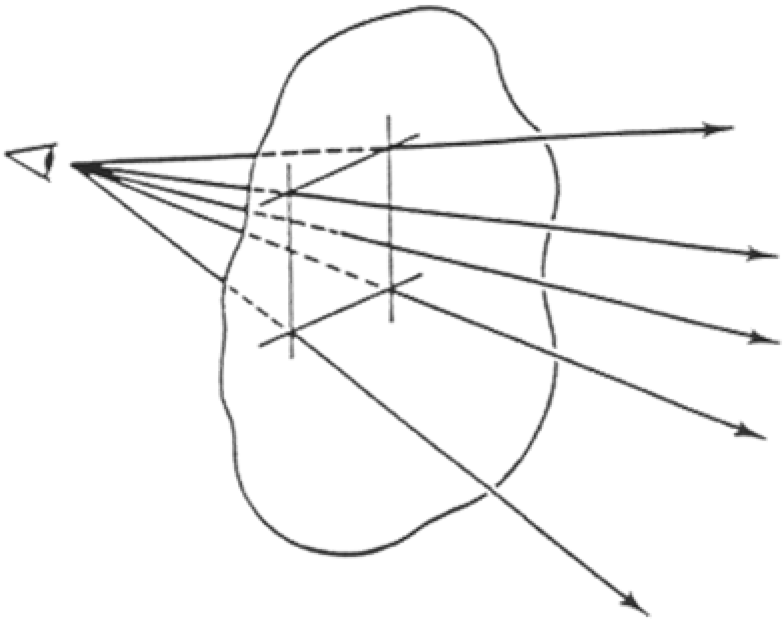


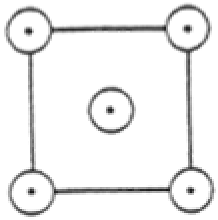
An object outside the focal plane appears blurry. This can be achieved by correctly calculating the refraction of the lens using a stochastic distribution of the rays across the surface of the lens.

10.4 Ray-tracing

Adaptive super-sampling

In order to avoid aliasing artifacts, several rays can be traced through a single pixel and the results averaged.



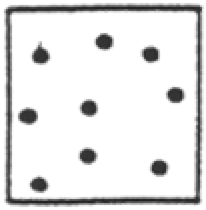


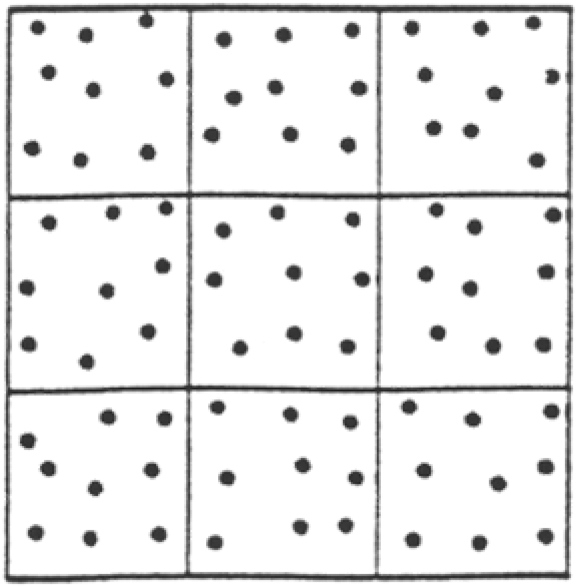
Example: four rays traced that cross each vertex and the center of the pixel

10.4 Ray-tracing

Stochastic ray-tracing

Instead of using a fixed distribution we can use stochastic methods, for example, random points for super-sampling:





10.4 Ray-tracing

Properties

- + The physical properties of lighting is modeled very well
- + Very suitable for high reflective surface, such as mirrors
- + The visibility problem is solved automatically
- + Great realism

- – Not quite suitable for diffuse reflection
- – High computation effort required
- – Computation of the intersections expensive
- – Sensitive to numerical errors

10.5 Radiosity

Radiosity techniques compute the **energy transfer** based on **diffuse radiation** between different surface components (e.g. polygons, triangles, …) of a scene. By incorporating diffuse reflections from all different objects within the scene, we get a quite different effect in terms of ambient lighting. This can be particularly important, for example for interior designers.

The correlation between different objects are described by integral equations, which are approximated to find the solution. The total emitted energy (**radiosity**) is then used as input for the rendering method. In order to integrate specular reflection, radiosity can be combined with ray-tracing.

10.5 Radiosity

- Radiosity incorporates the propagation of light while observing the energetic equilibrium in a closed system.
- For each surface, the emitted and reflected amount of light is considered at all other surfaces.
- For computing the amount of incoming light at a surface we need:
  - The complete geometric information of the configuration of all emitting, reflecting, and transparent objects.
  - The characteristics with respect to their light emitting properties of all objects.

10.5 Radiosity

Let S be a three-dimensional scene consisting of different surface elements

$$S = \{ S_i \}.$$

and E the emitted power per surface

$$E(x) \quad [\text{ W/m}^2]$$

in every point x of S. The surfaces with $E(x) \neq 0$ are light sources.

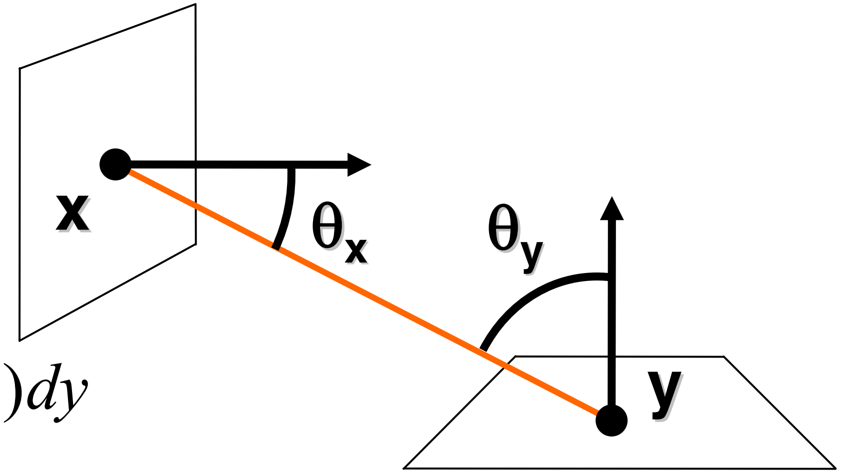
We then are interested in the radiosity function

$$B(x) \quad [\text{W/m}^2]$$

which defines the (diffuse) reflected energy per surface. This can then be used to determine color values and render the scene.

10.5 Radiosity

Energy transport



$$B(x) = E(x) + \rho(x) \int_S F(x,y) B(y)\, dy$$

$\rho(x) \in [0,1]$ Reflectivity of the surface S at $\mathbf{x}$

$$F(x,y) = V(x,y) \frac{\cos\theta_x \cos\theta_y}{\pi \|x-y\|^2}$$

← Angle of incidence and reflection

← Attenuation

Visibility (0 or 1)

Normalization

10.5 Radiosity

If the equation describing the energy transport is discretized, for example using the B-spline basis functions to express the functions *E(x)* and *B(x)*

$$E(x) = \sum_i e_i B_i(x) \qquad B(x) = \sum_i b_i B_i(x)$$

we can derive from the integral equation

$$B(x) = E(x) + \rho(x) \int_S F(x, y) B(y) dy$$

a system of linear equations

$$b_i = e_i + \rho_i \sum_j f_{ij} b_j$$

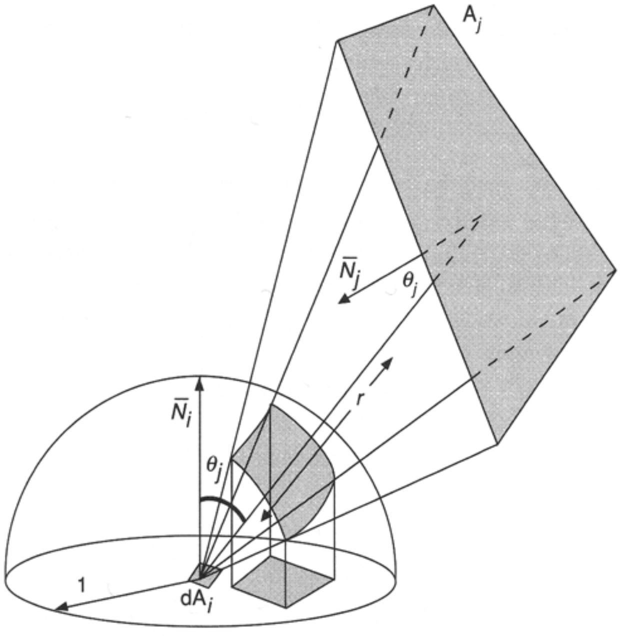
10.5 Radiosity

The coefficients e_i of the emitting function $E(x)$ and the reflectivity coefficients ρ_i of the individual surfaces segments S_i are known. The ρ_i describe the material properties, which determine which part of the orthogonally incoming light is reflected.

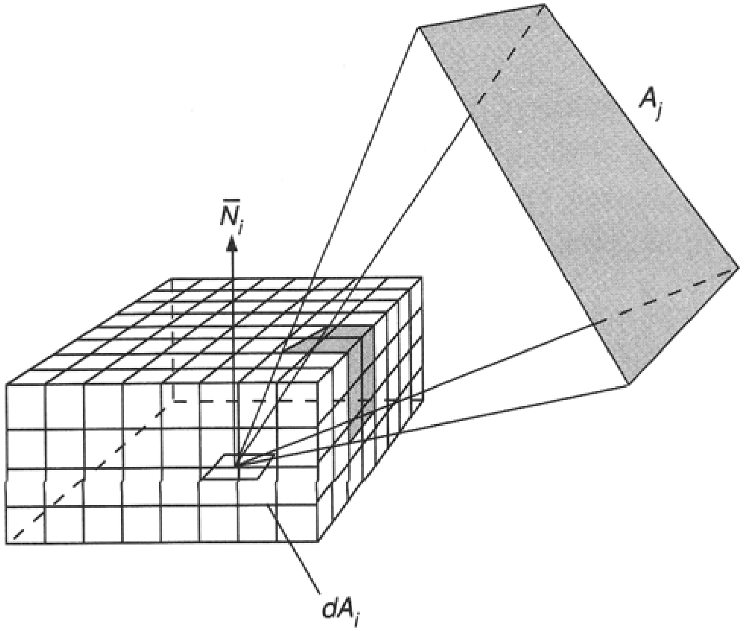
The form factors f_{ij} determine the portion of the energy that is transported from S_j to S_i and depend only on the geometry of the scene. The form factors can be computed using numerical integration across the hemisphere observing the size, direction, and mutual visibility of the surface segments S_j and S_i.

10.5 Radiosity

Computation of the form factors using numerical integration for a differential surface dA_i:



Hemisphere, Nusselt ´81



Hemicube, Cohen ´85

10.5 Radiosity

Computation of the form factors

The form factors between the differential surfaces dA_j and dA_j of the surface segments S_j and S_i result in:

$$dF_{ij} = v_{ij} \frac{\cos\theta_i \cos\theta_j}{\pi \cdot r^2} dA_j$$

v_{ij}: visibility

(1 if dA_j visible from dA_i, 0 otherwise)

The form factors of S_j to S_i equal to

$$f_{ij} = \int_{S_i}\int_{S_j} v_{ij} \frac{\cos\theta_i \cos\theta_j}{\pi \cdot r^2} dA_j dA_i$$

10.5 Radiosity

The resulting system of linear equations can then be described as:

$$b_i - \rho_i \sum_j f_{ij} b_j = e_i$$

Or in matrix form:

$$\begin{pmatrix} 1-\rho_1 f_{11} & -\rho_1 f_{12} & \cdots & -\rho_1 f_{1n} \\ -\rho_2 f_{21} & 1-\rho_2 f_{22} & \cdots & -\rho_2 f_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_n f_{n1} & -\rho_n f_{n2} & \cdots & 1-\rho_n f_{nn} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{pmatrix}$$

This system is usually solved separately for the different frequencies of light (e.g. RGB).

10.5 Radiosity

Rendering a scene

- Computation of the radiosity values b_i for all surface segments S_i
- Projection of the scene and determination of the visibility
- Computation of the color values for each pixel

Comments:

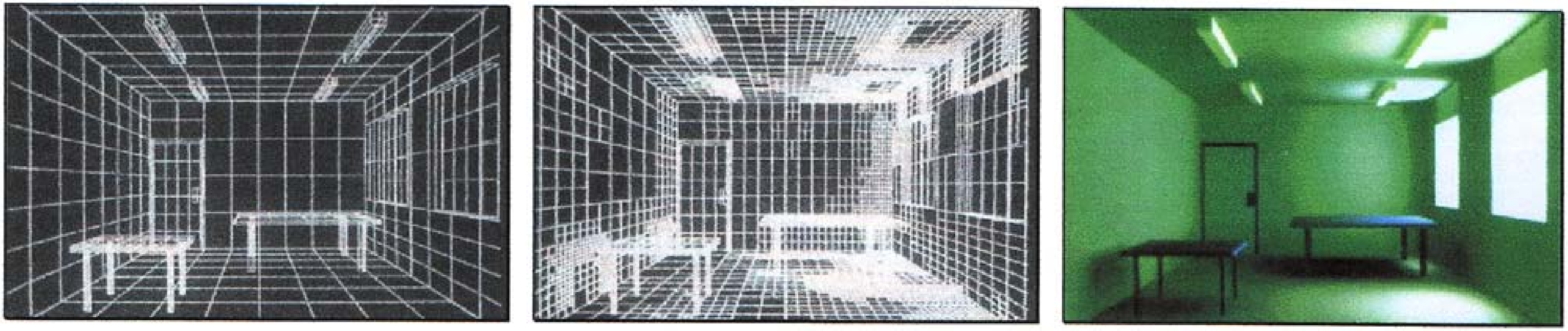
- For different views, only the steps two and three need to be repeated
- Step three can be accelerated by interpolating along scan-lines
- For step one, the form factors f_{ij} need to be calculated before the system of linear equations can be solved.

10.5 Radiosity

Sub-division of a scene

The finer the sub-division of the scene into surface segments the better the results. However, the number of form factors increase quadratically and the size of the system of linear equations increases linearly with an increasing number of surface segments.

In addition, approximation of the integral equation using the system of linear equations only works for constant radiosity per surface segment. Sub-division is required for critical (non-constant) areas.



10.6 Texture mapping

Motivation

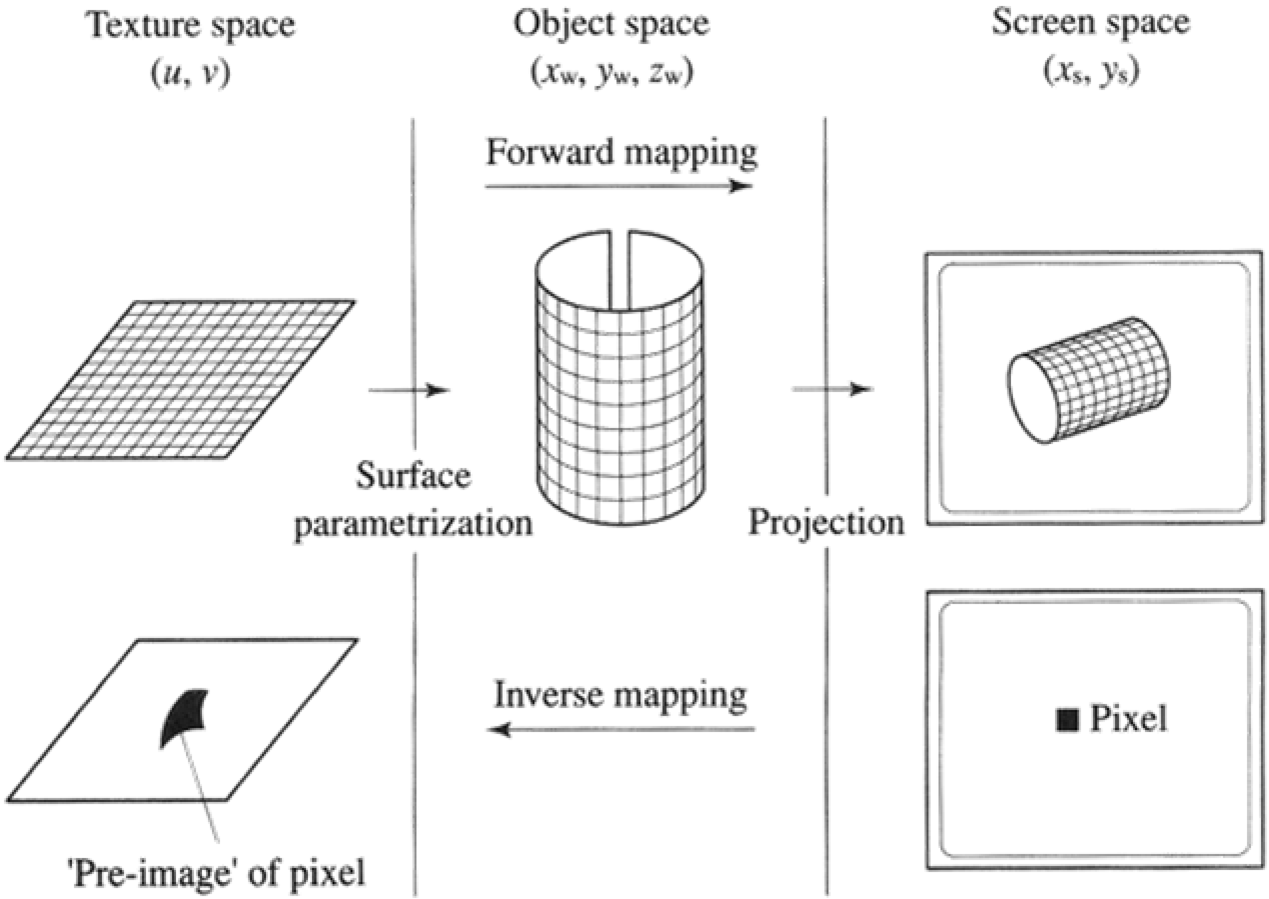
So far, all surfaces (polygonal objects or free-form surfaces) were modeled as idealized, smooth objects – in contrast to real-world surfaces with lots of detail.

The explicit modeling of surface details is too costly for rendering and is therefore simulated using different mapping techniques.

In the beginning, plain **texture mapping** [Catmull 1974] was introduced, which projected two-dimensional structures and patterns (**textures**, consisting of texture elements, **texels**) onto the surfaces of the objects. Several variations were then developed on top of this technique.

10.6 Texture mapping

Principle



10.6 Texture mapping

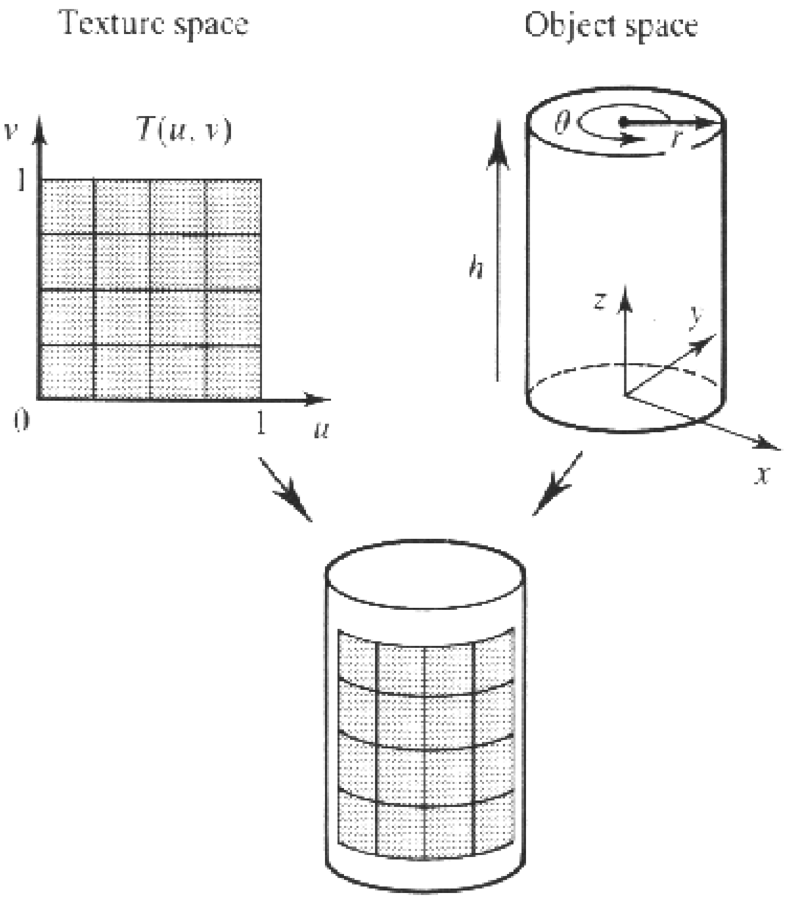
Comments:

- We usually distinguish between two different approaches: **forward** and **inverse mapping**
- In practice, it proved useful to split up the mapping process into two steps (for example for the forward mapping):
  1. First, the texture is projected onto an interim object using a simple mapping → "s-mapping"

     Rectangles, cubes, cylinders, or spheres are often used
  2. Then, the texture is mapped onto the object that is to be texturized → "o-mapping"

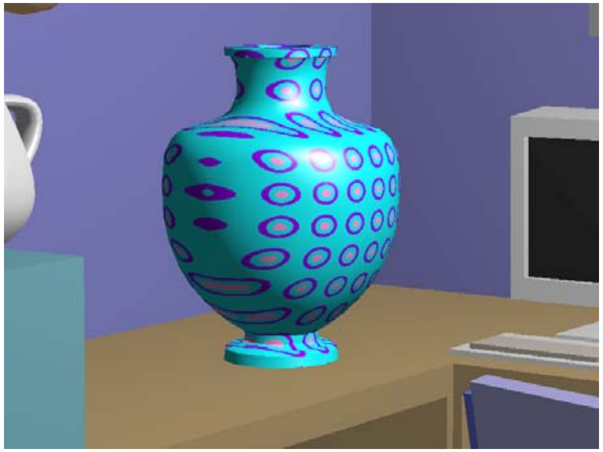
10.6 Texture mapping

Example: interim objects



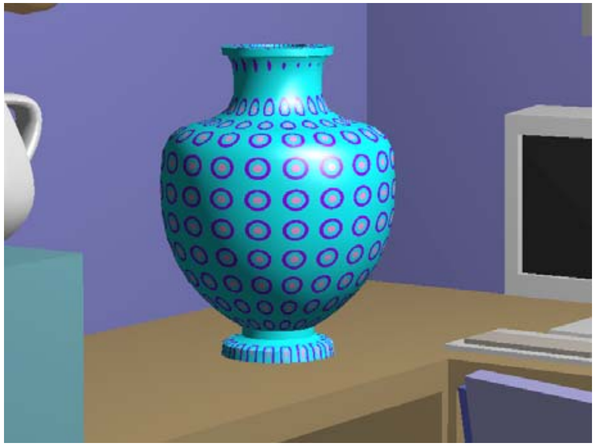
10.6 Texture mapping

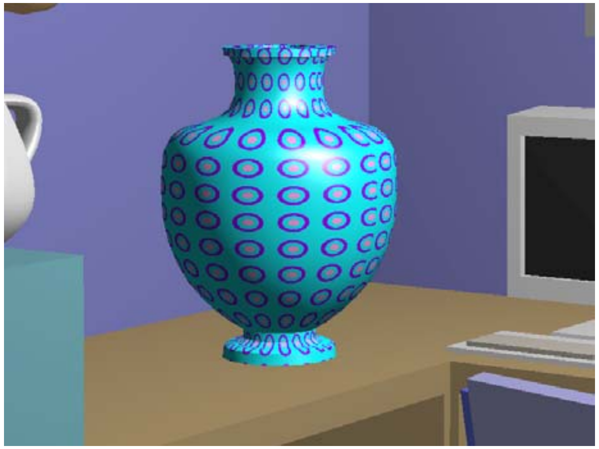
Example: interim objects



Planar

Sphere



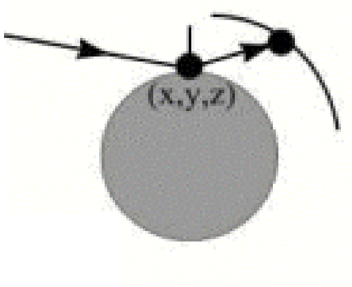


Cylinder

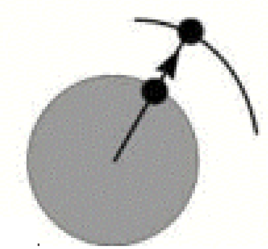
10.6 Texture mapping

Approaches to o-mapping

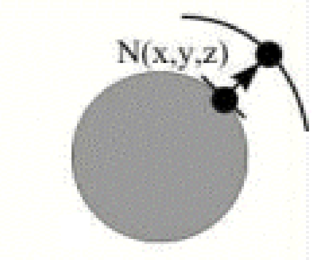
1. Reflective ray



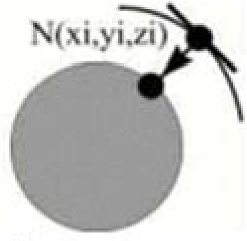
2. Object center



3. Normal vector

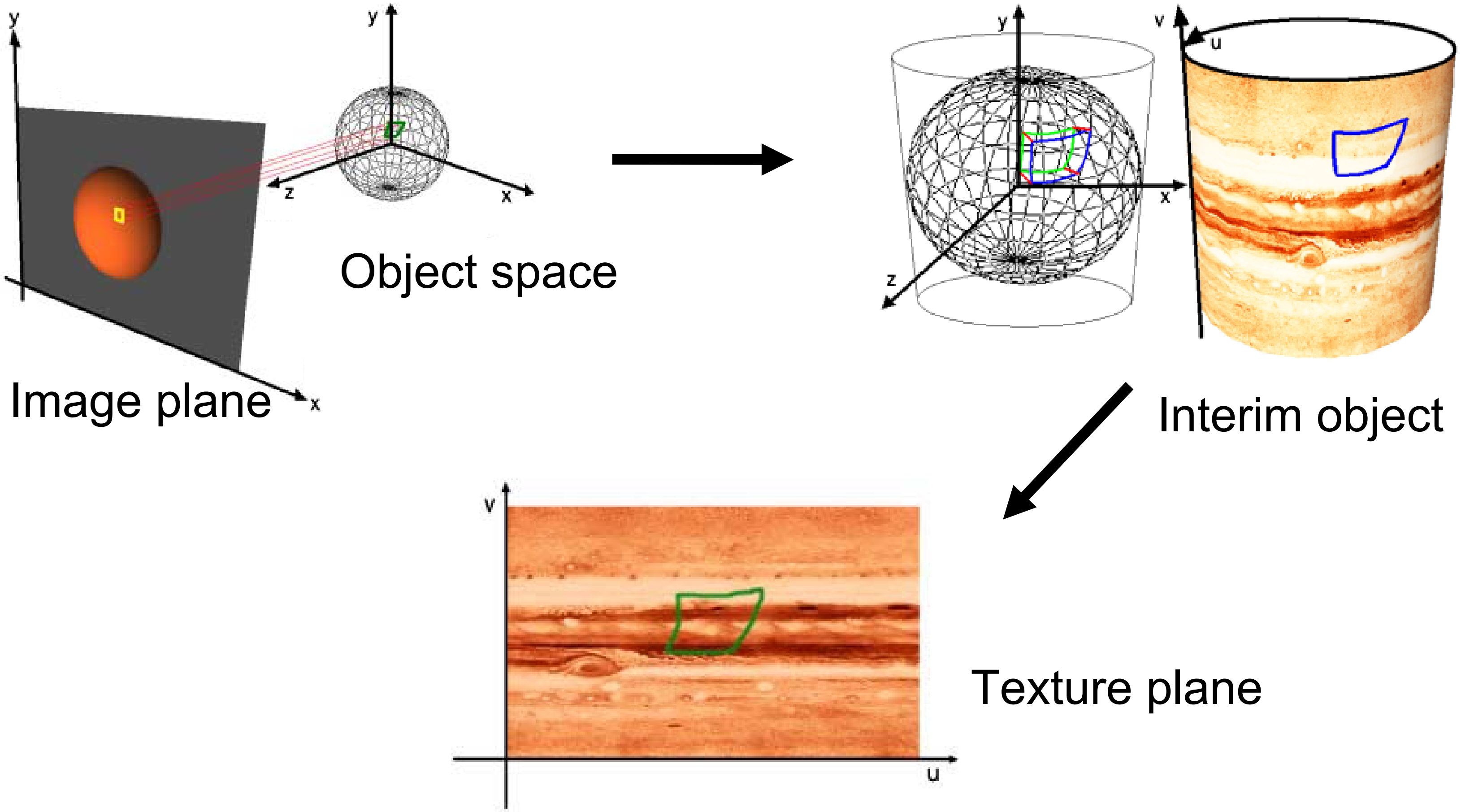


4. Normal of a temporary object



10.6 Texture mapping

Inverse mapping using interim objects



Object space

Image plane

Interim object

Texture plane

10.6 Texture mapping

When using free-form surfaces (Bézier splines, B-splines), we can use the parameterization of the surface instead of mapping to an interim object. The parameter of a point on the surface is then also a texture coordinate.

For triangulated surfaces, we usually define a texture coordinate for every vertex in addition to the normal vector (color information is not necessary in this case since it is overwritten by the texture). During rasterization of the triangles, a texture coordinate is computed for every pixel using linear interpolation (OpenGL does this automatically).

Modern graphics hardware often store textures in graphics memory for faster access.

10.6 Texture mapping

Aliasing

Texture mapping is very sensitive to aliasing artifacts:

- A pixel in image space can cover an area of several texels.
- On the other hand, a texel of the texture can cover more than one pixel in the resulting image.
- Textures are often patched together periodically, in order to cover a larger area. If the sampling rate is to low aliasing artifacts occur.

→ Oversampling, filtering, interpolation (instead of sampling), level-of-detail

10.6 Texture mapping

Bump mapping

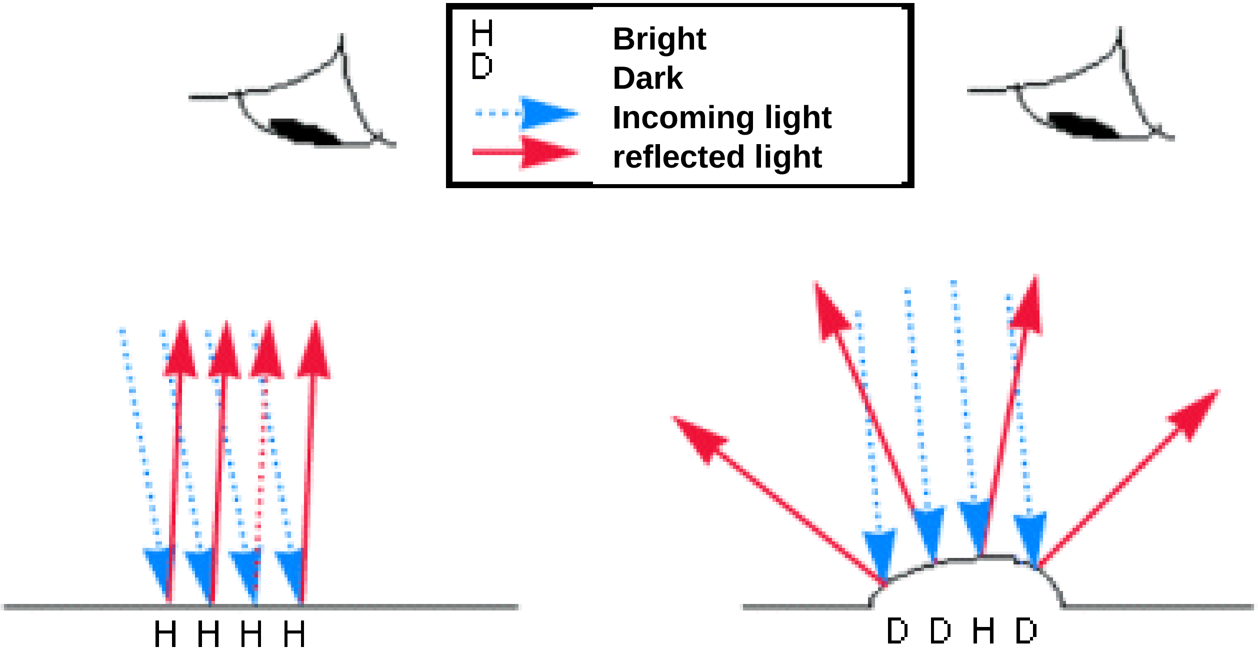
Texture mapping simulates a textured but planar/smooth surface.

In order to simulate a “rougher” surface and make it appear “more” three-dimensional, **bump mapping** does not change the surface geometry itself but changes the normal vectors that are used by the lighting model:

Simulation of surface bumps on top of planar surfaces by changing the normal vectors.

10.6 Texture mapping

Bump mapping



H Bright

D Dark

Incoming light

reflected light

a smooth planar surface appears evenly bright

an arched surface appears darker when facing away from the viewer

10.6 Texture mapping

Bump mapping

The change ΔN of the normal vector N is done in a procedural fashion or by using a texture map. This change can, for example, be described by a gray-level texture. The gradient of this texture (interpreted as a scalar field) then gives us the amount and direction of the change ΔN.

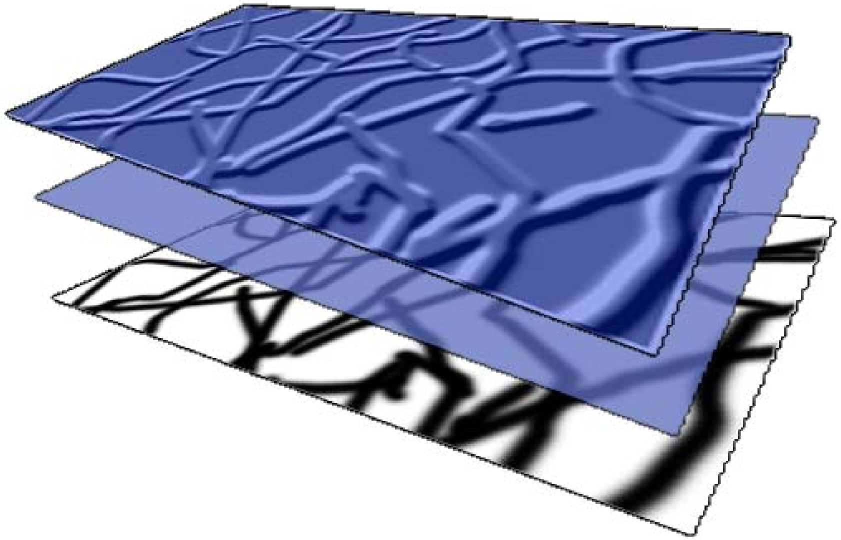
This way, regular structures (e.g. a golf ball) as well as irregular structures (e.g. bark) can be simulated.

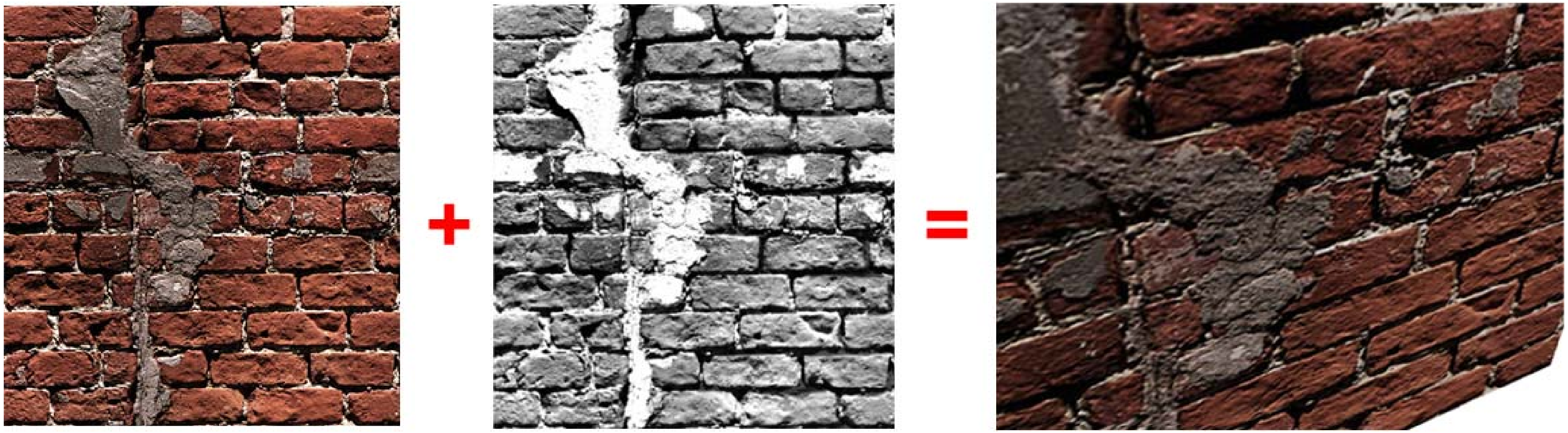
When looking at an object that uses bump mapping, it is, however, often noticeable that the surface itself is planar.

10.6 Texture mapping

Bump mapping

Examples:

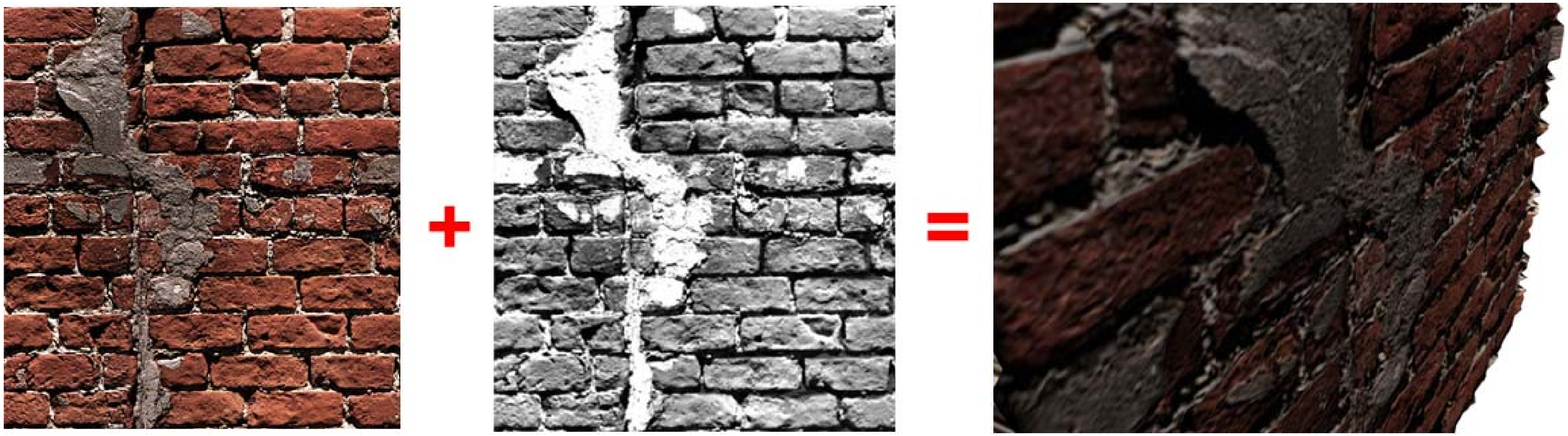




10.6 Texture mapping

Displacement mapping

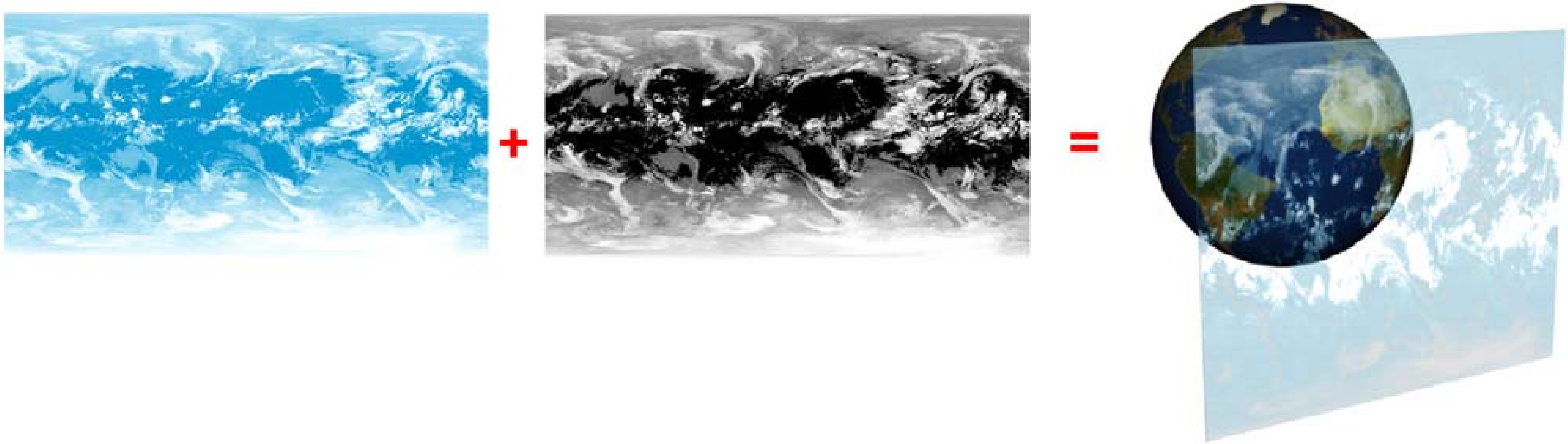
On top of the surface, a height-field is used, which moves the points of the surface in direction of the normal vector. This technique also changes the shape of the surface making it no longer planar.



10.6 Texture mapping

Opacity mapping / transparency mapping

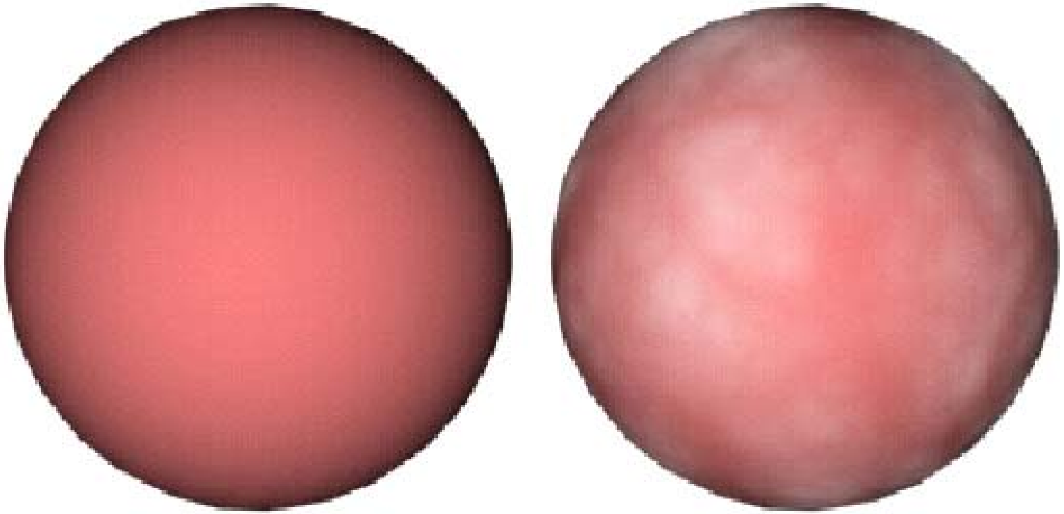
Using opacity mapping, the alpha-value of a transparent surface can be changed locally. The object, for which opacity mapping is used, can be changed according to the used texture in its entirety or only locally.



10.6 Texture mapping

Procedural mapping

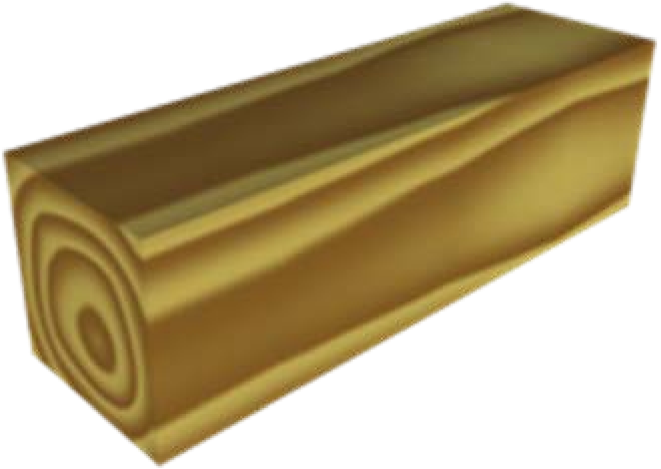
An algorithmic description, which simulates bumps or unevenness, is used to change the surface of an object. This is often used, for example, for 3-D textures.



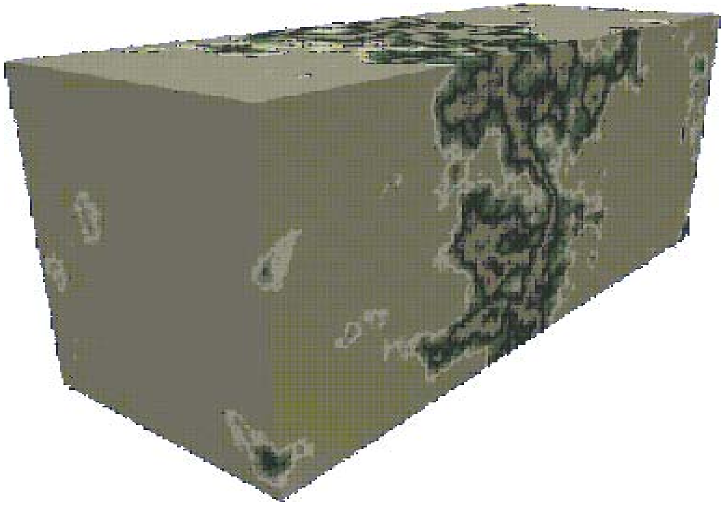
10.6 Texture mapping

3-D (texture) mapping

Instead of a 2-D image, a 3-D texture (volumetric image) is used and usually mapped onto a series of planes.



wood grain

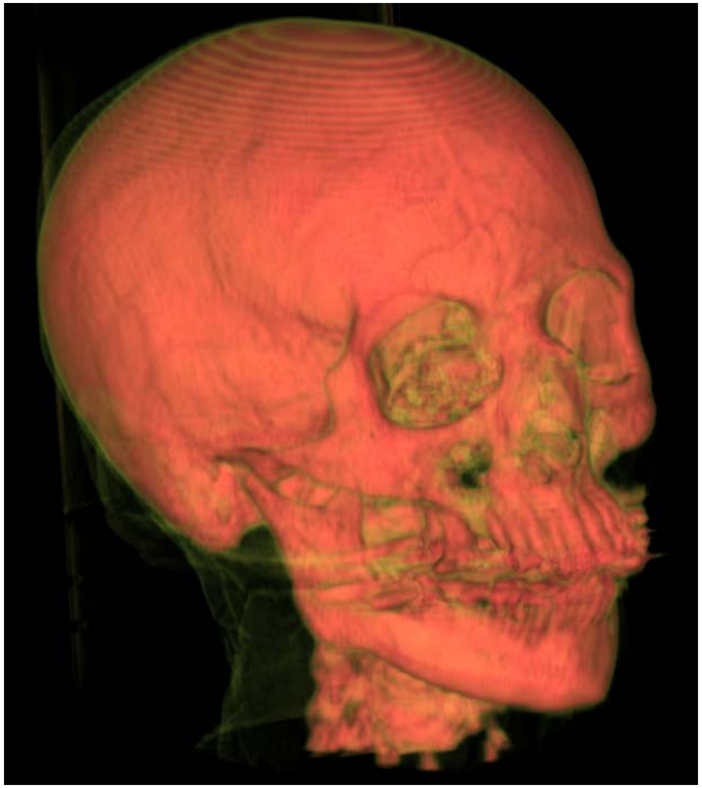


marble

10.6 Texture mapping

3-D (texture) mapping

3-D texture mapping can be used to visualize (e.g. medical) 3-D volumetric data sets:



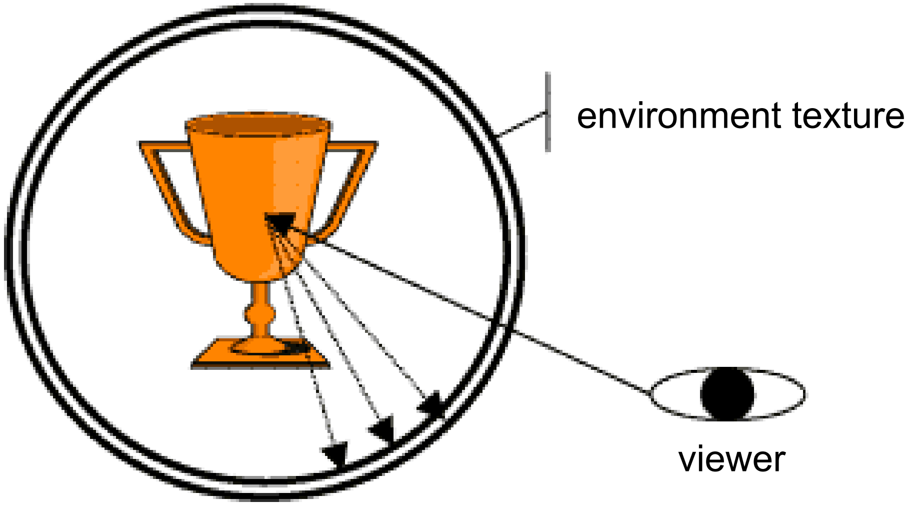
10.6 Texture mapping

Environment mapping

Environment mapping simulates realistic mirroring effects of the (virtual or physical) environment surrounding the object. This way, a complex surrounding can be integrated as a photo-realistic image, without explicitly modeling the surrounding.

An interim objects (sphere, cube) is used to project the environment on.

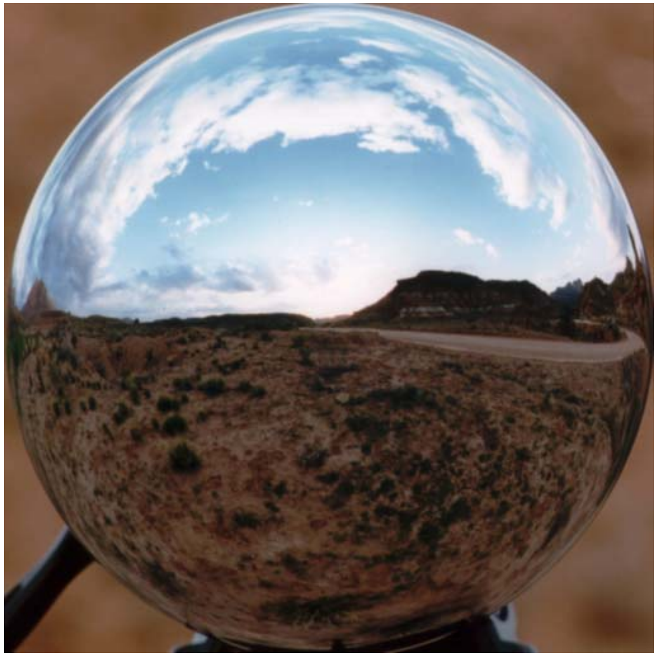
Nowadays, this is supported by current graphics hardware.

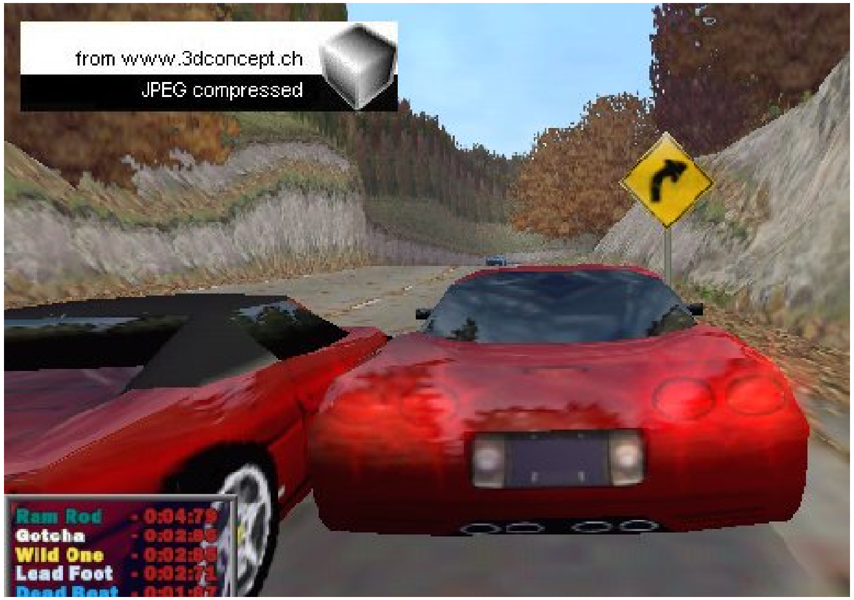


10.6 Texture mapping

Environment mapping

Examples:

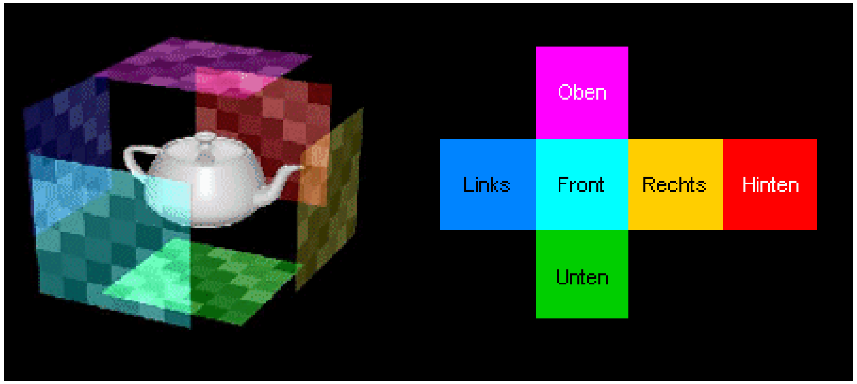


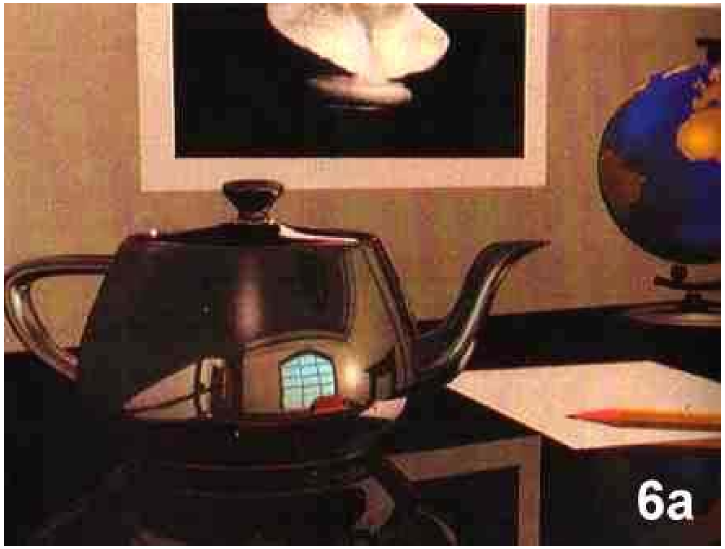


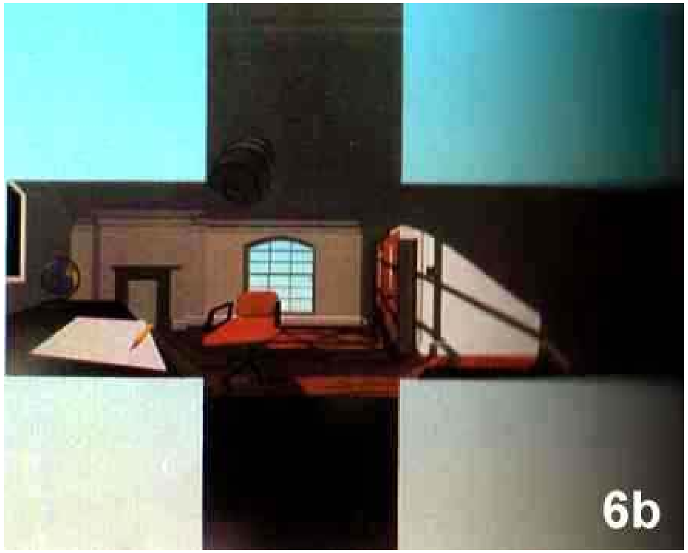
10.6 Texture mapping

Environment mapping

Examples:



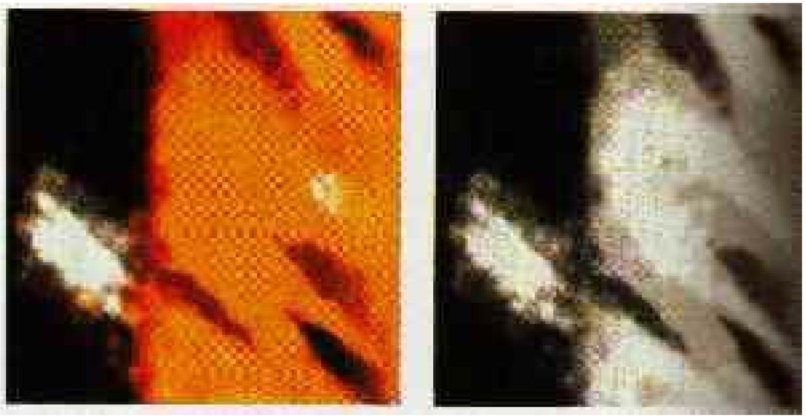




10.6 Texture mapping

Chrome / reflection mapping

An arbitrary two-dimensional pattern is mapped onto a reflecting surface. The texture itself stays fixed at a certain location in 3-D space. Often blurriness is used to achieve more realistic effects.



10.6 Texture mapping

Example: chrome / reflection mapping + ray-tracing

10.6 Texture mapping

Comments:

Different types of mapping techniques can be combined and applied to the same surface. This is supported in most commercial rendering and animation tools.

Most of these techniques can be implemented on the graphics hardware (after appropriate pre-processing) achieving rendering in real-time.

See, for example, NVIDIA’s FX Composer:

http://developer.nvidia.com/object/fx_composer_home.html

10.7 OpenGL illumination and surface rendering

Light sources

OpenGL supports up toe eight light source (`GL_LIGHT0` through `GL_LIGHT8`). To enable lighting you need to issue:

```
      glEnable (GL_LIGHTING)
```

Each light source can be enabled using, for example:

```
      glEnable (GL_LIGHT0)
```

Properties of light sources can be changed using the command:

```
      glLight* (lightName, lightProperty,
                propertyValue);
```

10.7 OpenGL illumination and surface rendering

Properties

Different properties are available:

Location:

```
GLfloat position [] = { 0.0, 0.0, 0.0 };
glLightfv (GL_LIGHT0, GL_POSITION, position);
```

Color:

```
GLfloat color [] = { 1.0, 1.0, 1.0 };
glLightfv (GL_LIGHT0, GL_AMBIENT, color);
glLightfv (GL_LIGHT0, GL_DIFFUSE, color);
glLightfv (GL_LIGHT0, GL_SPECULAR, color);
```

10.7 OpenGL illumination and surface rendering

Attenuation:

```
glLightf (GL_LIGHT0, GL_CONSTANT_ATTENUATION,
          1.5);
glLightf (GL_LIGHT0, GL_LINEAR_ATTENUATION,
          0.75);
glLightf (GL_LIGHT0, GL_QUADRATIC_ATTENUATION,
          0.4);
```

Spot lights:

```
GLfloat direction [] = { 1.0, 0.0, 0.0 };
glLightfv (GL_LIGHT0, GL_SPOT_DIRECTION,
           direction);
glLightf (GL_LIGHT0, GL_SPOT_CUTOFF, 30.0);
glLightf (GL_LIGHT0, GL_SPOT_EXPONENT, 2.5);
```

10.7 OpenGL illumination and surface rendering

Material properties

Different kind of materials can be generated with regard to, for example, their shininess using `glMaterial*`:

```
GLfloat diffuse []
    = { 0.2, 0.4, 0.9, 1.0 };
GLfloat specular []
    = { 1.0, 1.0, 1.0, 1.0 };
glMaterialfv (GL_FRONT_AND_BACK,
   GL_AMBIENT_AND_DIFFUSE, diffuse);
glMaterialfv (GL_FRONT_AND_BACK,
   GL_SPECULAR, specular);
glMaterialf (GL_FRONT_AND_BACK, GL_SHININESS,
   25.0);
```

10.7 OpenGL illumination and surface rendering

Normal vectors

Normal vectors can be provided by using the command `glNormal*`:

```
GLfloat normal [] = { 1.0, 1.0, 1.0 };
GLfloat vertex [] = { 2.0, 1.0, 3.0 };
glNormal3fv (normal);
glVertex3fv (vertex);
```

Make sure that the normal vector is provided **before** the vertex since OpenGL is a state machine!

If your normal vectors are not normalized OpenGL can do that for you if you issue:

```
glEnable (GL_NORMALIZE);
```

10.7 OpenGL illumination and surface rendering

Shading model

In OpenGL, two shading models are available: flat shading and Gouraud shading (default)

You can specify which of the shading models to use by using the following function

```
glShadeModel (mode);
```

Where mode can assume the values `GL_SMOOTH` and `GL_FLAT`.

10.8 OpenGL texture functions

Creating the texture and copy it to the graphics memory:

```
GLuint image [];
unsigned int width = 256, height = 256;
glTexImage2D (GL_TEXTURE_2D, 0, GL_RGBA,
              width, height, 0, GL_RGBA,
              GL_UNSIGNED_BYTE, image);
```

Enable textures:

```
glEnable (GL_TEXTURE_2D);
```

If you use OpenGL prior to version 2.0 width and height have to be powers of two!

10.8 OpenGL texture functions

Texture coordinates

Provide a texture coordinate for every vertex of your polygonal mesh:

```
GLfloat texcoord = { 1.0, 1.0 };
GLfloat vertex = { 2.0, 1.0, 3.0 };
glTexCoord2fv (texcoord);
glVertex3fv (vertex);
```

Again, provide the texture coordinate before the vertex!

When using vertex arrays, texture coordinates can also be provided as a single array:

```
GLfloat texcoordarray;
glEnableClientState (GL_TEXTURE_COORD_ARRAY);
glTexCoordPointer (nCoords, GLfloat, 0,
                         texcoordarray);
```

10.8 OpenGL texture functions

Naming textures

If you use more than one texture you need to provide names in order to be able to switch between the provided textures.

```
GLuint texname;
glGenTextures (1, texname);
```

Then, you can change between them using these names:

```
glBindTextures (GL_TEXTURE_2D, texname);
```

Remember, OpenGL is a state machine so it will use this texture from now on for every texture related commands!