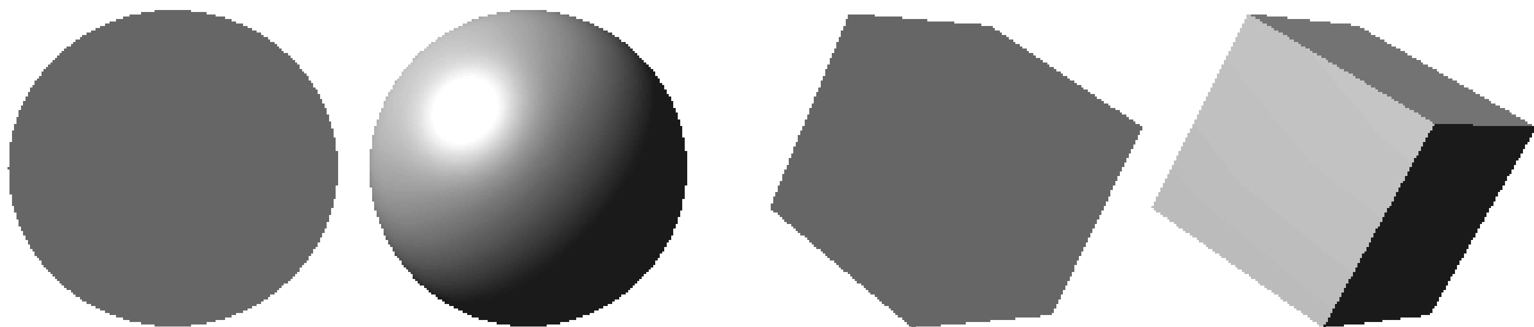
Illumination and shading



Objects with and without shading effects

Shading: Colour or intensity modifications of a surface by illumination effects.

© Springer

Illumination models

More realistic images require detailed computation of illumination effects (reflection, shading, shadows,...).

From a theoretical point of view, the computations for shading described in the following sections would have to be carried out for each wavelength of the light individually.

From a practical point of view, the computations will always be restricted to the three primary colours red, green and blue in order to determine the RGB-values for the representation.

© Springer

Light sources

- **Ambient light** does not come from a specific light source and has no direction. It represents the light that is more or less everywhere in the scene, originating from multiple reflections of light at various surfaces. Ambient has only a colour.

- **Directional light sources** is used to model light coming from a source in almost infinite distance, for instance sunlight. The light rays are all parallel. In addition to the colour, the light rays have a direction.

© Springer

Light sources

- A lamp is modelled as a **point light source.**
  - Point light source have a colour and a position.
  - The intensity of the light decreases with increasing distance (**attenuation**).

    The intensity decreases quadratically with the distance.

© Springer

Light sources

Theoretical approach: Multiply the intensity by the factor $1/d^2$ when the light hits the surface of an object at distance d.

Drastic effect: $d \to 0 \;\Rightarrow$ intensity $\to \infty$

Modified model:

$$f_{\text{att}} \;=\; \min\left\{\frac{1}{c_1 + c_2 d + c_3 d^2},\; 1\right\}$$

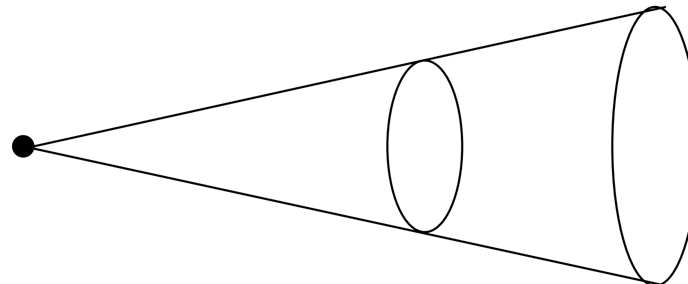
The linear term can model **atmospheric attenuation.**

© Springer

Light sources

- **Spotlight**: A point light source whose light is only emitted into one direction in the form of a cone.

  Also quadratic decrease of the intensity with increasing distance.



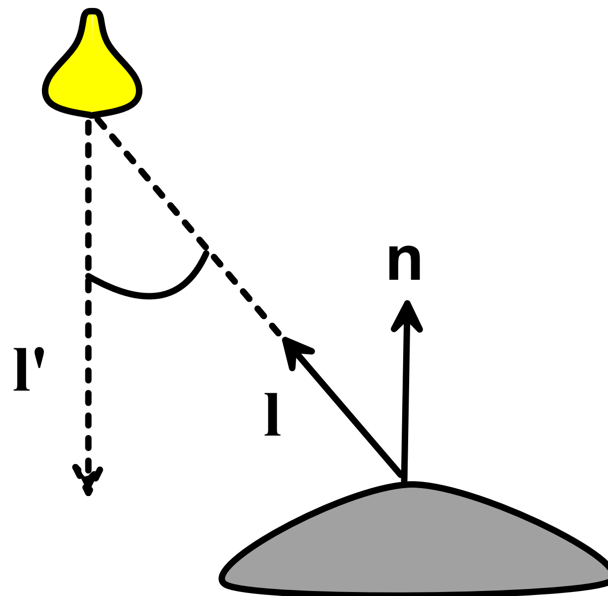
© Springer

Light sources

Intensity of the light is smaller close to the boundary of the cone of light than at the centre.

Intensity of light at the point on the surface coming from the spotlight (Warn model):

$$I \;=\; I_S \cdot f_{\text{att}} \cdot (\cos\gamma)^p \;=\; I_S \cdot f_{\text{att}} \cdot \left(-\mathbf{l}'^{\top} \cdot \mathbf{l}\right)^p$$



© Springer

Java 3D: Light sources

Ambient light:

```
AmbientLight ambLight =
             new AmbientLight(ambColour);
```

`ambLight.setInfluencingBounds(bounds);`
generated ambient light with the colour (`Color3f`) `ambColour`.

As already in the case of `Interpolator`s for animation, a light source must also of a region where it “shines”.

The method `setInfluencingBounds(bounds)` defines this region. `bounds` is a `BoundingSphere`.

© Springer

Java 3D: Light sources

Directional light:
```
DirectionalLight dirLight =
    new DirectionalLight(lightColour,
                         lightDir);
```

generates directional light of the colour `lightColour` with light rays parallel to the direction given by `Vector3f lightDir`.

© Springer

Java 3D: Light sources

Point light source:

```
PointLight pLight =
        new PointLight(lightColour,
                       location,
                       attenuation);
```

defines a point light source in the point (`Point3f`) `location` with colour `lightColour`.

`attenuation` is an instance of the class `Point3f` whose three components specify the coefficients of the polynomial in the denominator of the attenuation formula.

(c) Springer

Java 3D: Light sources

Spot light: A point light source emits light in all directions.

The class `SpotLight` extends the class `PointLight`.

A `SpotLight` needs a direction in which the light cone is oriented.

```
SpotLight spLight =
        new SpotLight(lightColour,
                      location,
                      attenuation,
                      direction,
                      angle,
                      concentration);
```

© Springer

Java 3D: Light sources

The first three parameters have the same meaning as for `PointLight`.

The `Vector3f direction` specifies the direction in which the spotlight shines.

The `float`-value `angle` defines the angular limit corresponding to half of the opening angle of the cone of light.

The `float`-value `concentration` between 0 and 120 determines how much the spotlight is focussed to the centre axis.

© Springer

Java 3D: Light sources

- All light sources should be assigned to the `BranchGroup bgLight` by `addChild(...)`.
- `bgLight` must be assigned to the `SimpleUniverse` by the method `addBranchGraph(...)`.

Java 3D: Light sources

Moving light sources:

- Do not assign the light source directly to the `BranchGroup bgLight`.
- Generate a transformation group for the movement.
- Assign the light source to this transformation group.
- Assign this transformation group to `bgLight`.

(see `MovingLight.java`)

© Springer

Java 3D: Light sources

- Light sources themselves are invisible.
- In order to model a lamp, a corresponding object must be created and assigned to the scene.
- The corresponding light source can be assigned to the transformation group of the object instead of `bgLight`.

© Springer

Light emitting objects

Objects emitting light:

Equation for the intensity (for shading): $I = k_i$

In detail:

$$I^{\text{(red)}} = k_i^{\text{(red)}} \qquad I^{\text{(green)}} = k_i^{\text{(green)}} \qquad I^{\text{(blue)}} = k_i^{\text{(blue)}}$$

An object emitting light is not considered as a light source in the scene and will not illuminate other objects.

© Springer

Reflection

Intensity at a pixel illuminated by a light source:

$$I = I_{\text{light source}} \cdot f_{\text{pixel}}.$$

- $I_{\text{light source}}$ is the intensity of the light coming from the light source.
- f_{pixel} is a factor that depends on various parameters. The colour of the surface, its shininess, the distance to the light source in case attenuation must be taken into account, and the angle at which the light hits the surface in the considered pixel.

© Springer

Reflection

Illumination equation for ambient light:

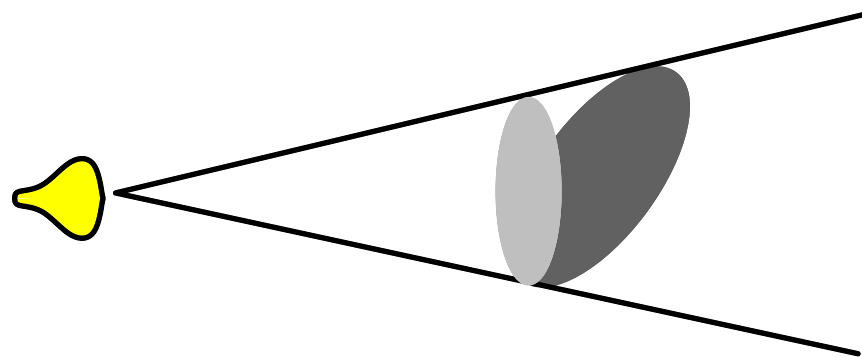
$$I = k_a \cdot I_S$$

k_a: reflection coefficient of the surface (for ambient light).

© Springer

Diffuse reflection

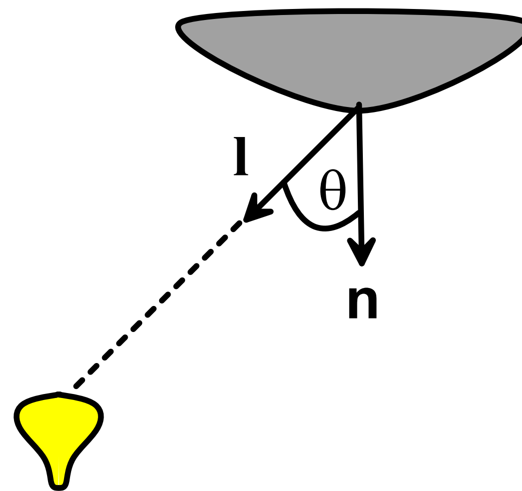
Light intensity depending on the angle in which the light hits the surface.



Lambert's cosine law:

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

Diffuse reflection

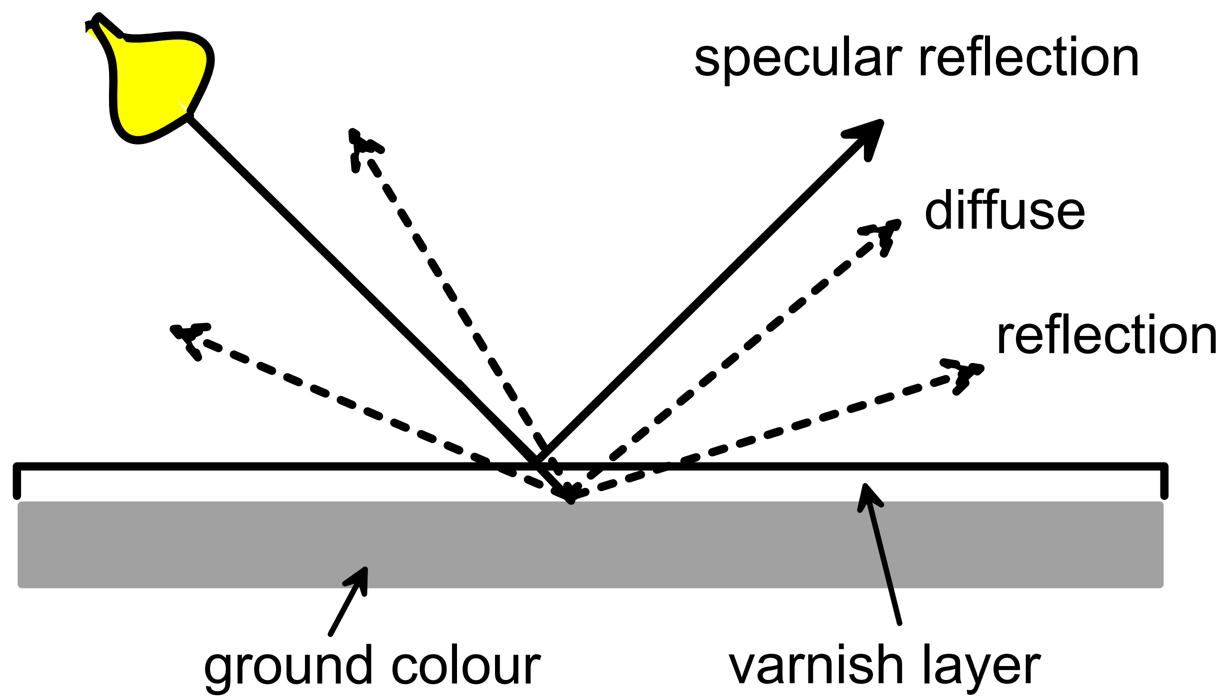


For normalised vectors:

$$I = I_L \cdot k_d \cdot (\mathbf{n}^\top \cdot \mathbf{l})$$

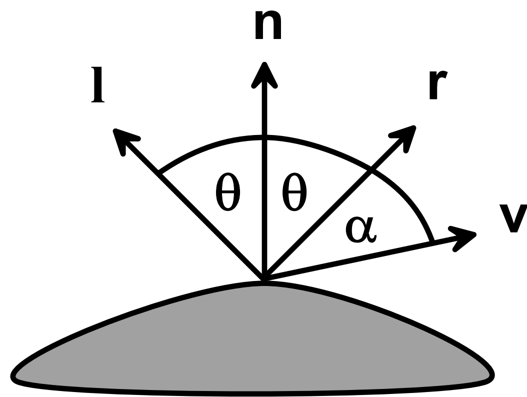
© Springer

Specular reflection



© Springer

Specular reflection



$$\mathbf{r} = \mathbf{n} \cdot \cos(\theta) + \mathbf{s} \qquad \mathbf{s} = \mathbf{n} \cdot \cos(\theta) - \mathbf{l}$$

$$\mathbf{r} = 2 \cdot \mathbf{n} \cdot \cos(\theta) - \mathbf{l} \qquad \mathbf{r} = 2 \cdot \mathbf{n}(\mathbf{n}^\top \cdot \mathbf{l}) - \mathbf{l}$$

Precondition: $0^\circ \leq \theta < 90^\circ$, i.e. $\mathbf{n}^\top \cdot \mathbf{l} > 0$

Specular reflection

Ideal mirror: Specular reflection only in the direction of the vector $\mathbf{r}$.

Shiny surface, but not perfect mirror: Specular reflection around $\mathbf{r}$.

Phong illumination model:

$$I = I_L \cdot W(\theta) \cdot (\cos(\alpha))^n$$

$0 \leq W(\theta) \leq 1$: Fraction of the light to which specular reflection applies for light under the angle θ.

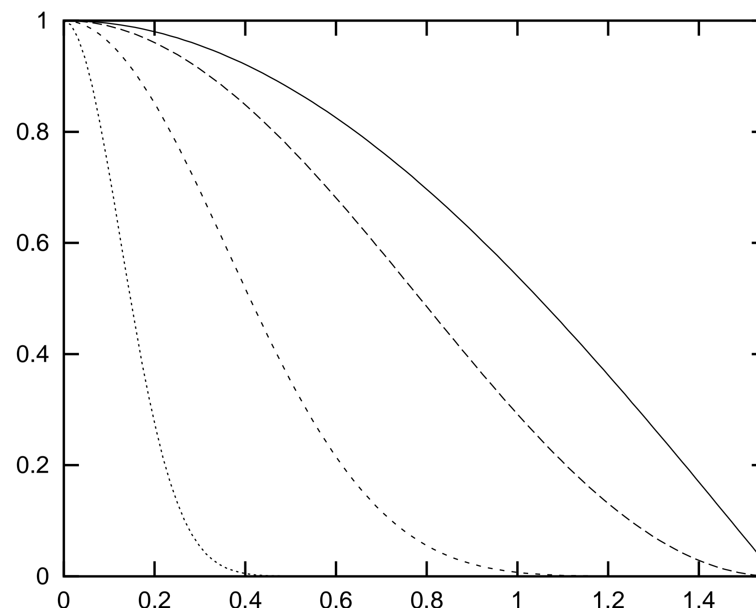
n: **Specular reflection coefficient,**

$n \to \infty$: Ideal mirror.

© Springer

Phong illumination model

$$I = I_L \cdot W(\theta) \cdot (\mathbf{r}^\top \cdot \mathbf{v})^n$$

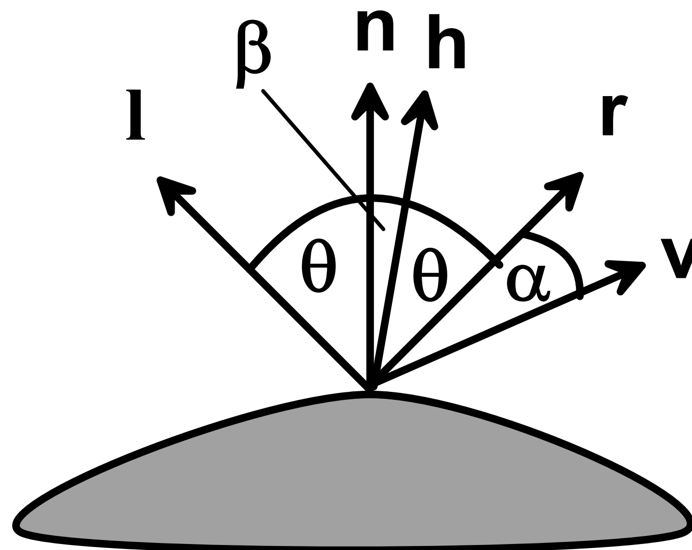


$$(\cos\alpha)^{64}, (\cos\alpha)^{8}, (\cos\alpha)^{2}, \cos\alpha$$

© Springer

Mod. Phong illumination model

Alternative measure for the strength of specular reflection: Deviation of the normal vector $\mathbf{n}$ from halfway vector $\mathbf{h}$ between $\mathbf{l}$ and $\mathbf{v}$.



© Springer

Mod. Phong illumination model

Modified Phong illumination model: Use $\cos\alpha$ instead of $\cos\beta$:

$$\cos\beta \;=\; \mathbf{n}^{\top} \cdot \mathbf{h}$$

where

$$\mathbf{h} \;=\; \frac{\mathbf{l} + \mathbf{v}}{\| \mathbf{l} + \mathbf{v} \|}$$

Advantage: In case of directional light and parallel projection, the halfway vector $\mathbf{h}$ does not change.

© Springer

Multiple light sources

$$
\begin{aligned}
I &= I_{\text{self_emission}} + I_{\text{ambient_light}} \cdot k_a \\
&+ \sum_j I_j \cdot f_{\text{att}} \cdot g_{\text{cone}} \cdot \left(k_d \cdot (\mathbf{n}^\top \cdot \mathbf{l}_j) + k_{\text{sr}} \cdot (\mathbf{r}_j^\top \cdot \mathbf{v})^n \right) .
\end{aligned}
$$

© Springer

Deferred Shading

High computational costs for the z-buffer algorithm when more than one or two light sources are present.

Complex computations for shading, even for objects which will be overwritten in the z-buffer by other objects.

Deferred Shading:

- Run the z-buffer algorithm once and fill only the z-buffer.
- In the second run (where also the frame buffer is filled), only those objects are shaded whose z-coordinate is entered in the z-buffer and that need to be shaded.

© Springer

Java 3D: Surface properties

An `Appearance` is assigned to each object providing information about the appearance of the object's surface.

An important attribute of `Appearance` is the `Material`.

`Material` defines colour and reflection properties of the surface.

© Springer

Java 3D: Surface properties

Constructor for `Material`:

```
Material ma =
             new Material(ambientColour,
                          emissiveColour,
                          diffuseColour,
                          specularColour,
                          shininessValue);
```

- `ambientColour` is a colour telling how much ambient light is reflected by the surface.

Java 3D: Surface properties

- `emissiveColour` is the intensity (colour) of the self-emitting light of the surface.

  The object will occur in this colour, when there is no other light. The object will illuminate other objects.

- `diffuseColour` is the colour for diffuse reflection.

- `specularColour` is the colour for specular reflection.

(c) Springer

Java 3D: Surface properties

- The value `shininessValue` is the specular reflection exponent in the Phong illumination model.

```
Appearance app = new Appearance();
```

generates a new `Appearance`.

© Springer

Java 3D: Surface properties

`app.setMaterial(ma);`
assigns surface properties to the `Appearance` that are defined in the `Material ma`.

An `Appearance app` can be assigned to any elementary geometric object (`Box, Sphere, Cylinder, Cone`) or `Shape3D` within the constructor or by the method `setAppearance(app)`.

(see `LightingExample.java` and `LightingExample2.java`)

© Springer

Shading

Bent surfaces are approximated by polygons.

A (planar) polygon has the same normal vector in each point.

Constant or **flat shading:**

For a polygon, the colour is determined only for a single pixel based on one normal vector. All other pixels resulting from the projection of the polygon obtain the same colour, leading to a homogeneous colour for the projection of the polygon.

© Springer

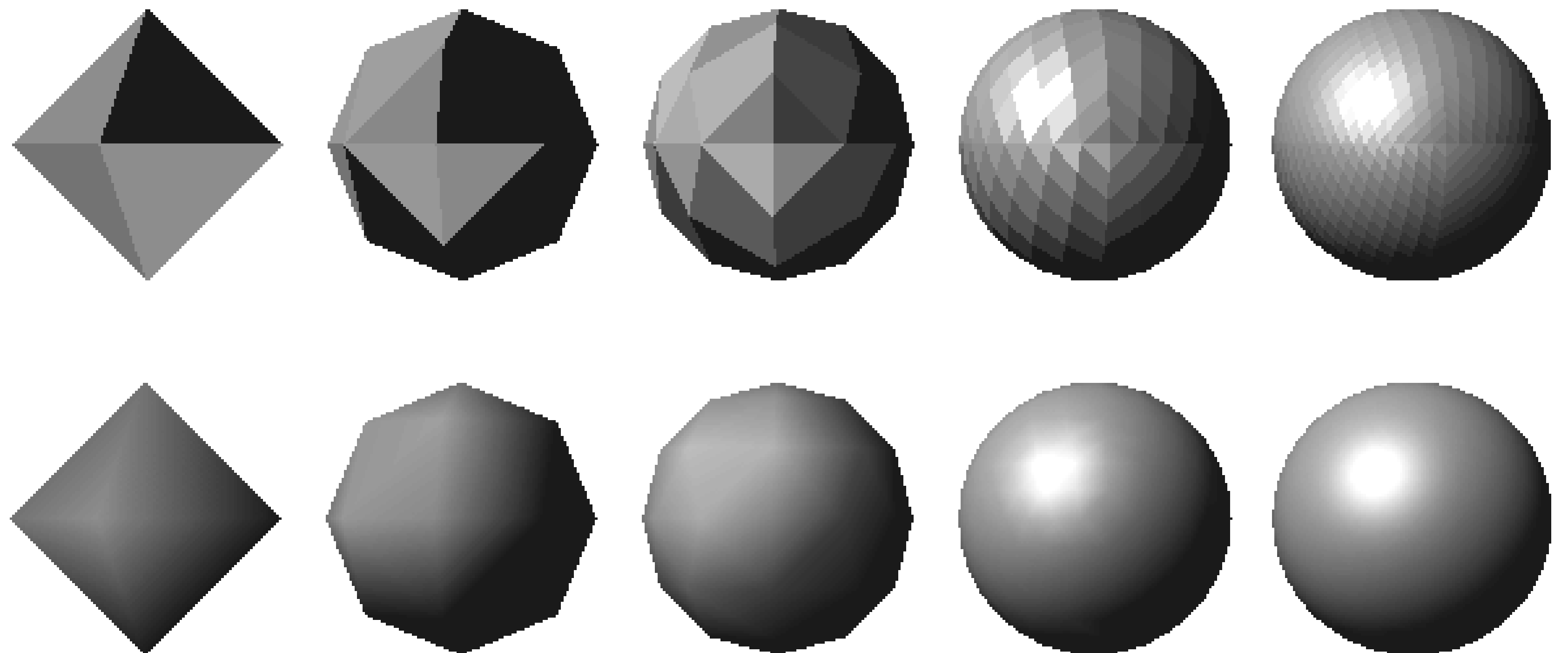
Constant shading

Implicit assumptions:

- The light source is in infinite distance so that $\mathbf{n}^\top \cdot \mathbf{l}$ is constant. This applies only to directional light sources.
- The viewer is in infinite distance so that $\mathbf{n}^\top \cdot \mathbf{v}$ is constant. This is true for parallel projections.
- The polygon represents the real surface of the object and is not just an approximation of a curved surface.
- No specular reflection occurs.

Constant vs. *Gouraud*

Constant shading leads to facets on on bent surfaces.



© Springer

Interpolated shading

Interpolated shading requires the definition of individual normal vectors in the vertices of a polygon or triangle.

The normal vectors can be derived from the original surface which is approximated by the triangles.

If the triangles are not derived from a freeform surface, but where specified manually, different normal vectors in the vertices of the triangle can still be computed. In each vertex, the standard normal vectors to the triangles that share the vertex are interpolated.

© Springer

Gouraud shading

Gouraud shading computes the colour in each of the three vertices of a triangle based on the corresponding normal vectors.

The shading of the other points in the triangle is based on colour interpolation derived from the three vertices.

This leads to a linear colour gradient over the triangle.

© Springer

Gouraud shading

Given the three vertices (x_i, y_i, z_i) and their associated intensities I_i $(i = 1, 2, 3)$, any point in the triangle can be represented in the form

$$t_1 \cdot (x_1, y_1, z_1) + t_2 \cdot (x_2, y_2, z_2) + t_3 \cdot (x_3, y_3, z_3)$$

where $t_1 + t_2 + t_3 = 1$, $t_1, t_2, t_3 \geq 0$.

Then

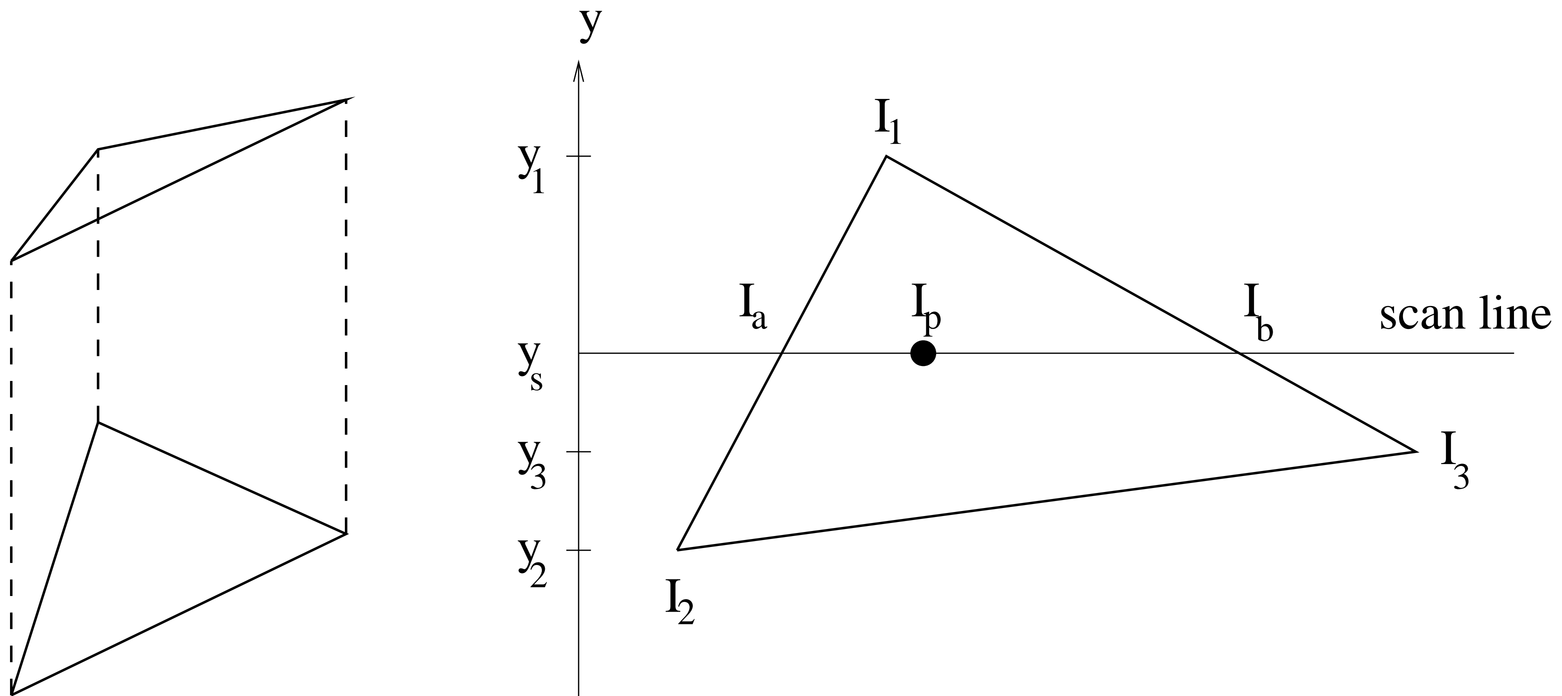
$$t_1 \cdot I_1 + t_2 \cdot I_2 + t_3 \cdot I_3$$

is the intensity assigned to the considered point.

Interpolated intensity is only an approximation of the true intensity.

© Springer

Gouraud shading



© Springer

Gouraud shading

$$I_a = I_1 - (I_1 - I_2)\frac{y_1 - y_s}{y_1 - y_2}$$

$$I_b = I_1 - (I_1 - I_3)\frac{y_1 - y_s}{y_1 - y_3}$$

$$I_p = I_b - (I_b - I_a)\frac{x_b - x_p}{x_b - x_a}$$

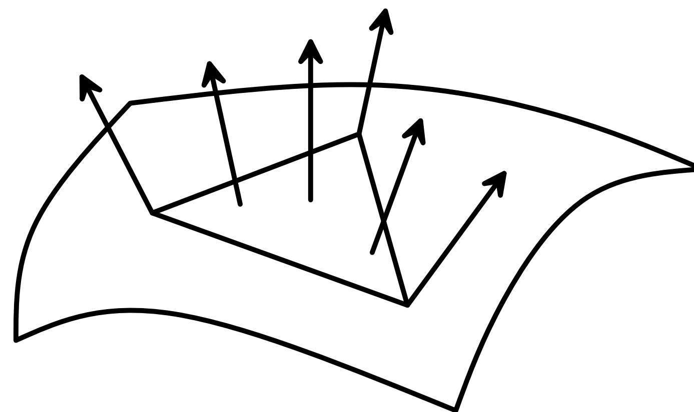
Because of the linear interpolation scheme for Gouraud shading, the minimum and maximum intensity on a triangle will always be in one of the vertices.

© Springer

Phong shading

Phong shading interpolates the normal vectors in the vertices instead of the colour intensities.

This means that the illumination equation must be evaluated for each point (pixel) in the triangle, not only for the vertices.



© Springer

Java 3D: Shading

Java 3D applies Gouraud shading by default.

Java 3D offers the choice between `SHADE_GOURAUD` and `SHADE_FLAT`.

`SHADE_FLAT` refers to constant shading.

The shading algorithm can be chosen in the `Appearance`.

© Springer

Java 3D: Shading

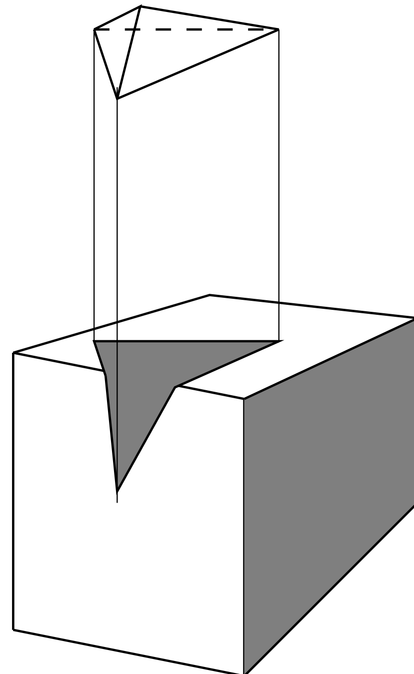
```
Appearance app = new Appearance();

ColoringAttributes ca = new
       ColoringAttributes(
         new Color3f(1.0f,1.0f,1.0f),
         ColoringAttributes.SHADE_FLAT);

app.setColoringAttributes(ca);
```

(see `ShadingExample.java`)

© Springer

Shadows

“Casting a shadow” is not an active matter, but simply the lack of light from a light source that does not reach the object’s surface with the shadow on it.



© Springer

Shadows

Illumination equation including shadows:

$$
\begin{aligned}
I &= I_{\text{self_emission}} + I_{\text{ambient_light}} \cdot k_a \\
&+ \sum_j S_j \cdot I_j \cdot f_{\text{att}} \cdot g_{\text{cone}} \cdot \left(k_d \cdot (\mathbf{n}^\top \cdot \mathbf{l}_j) + k_{\text{sr}} \cdot (\mathbf{r}_j^\top \cdot \mathbf{v})^n \right)
\end{aligned}
$$

where

$$
S_j = \begin{cases} 1 & \text{if the light from light source } j \text{ reaches the surface} \\ 0 & \text{otherwise (shadow).} \end{cases}
$$

Two-pass z-buffer algorithm

1. Apply the standard z-buffer algorithm for each light source. Consider the light source as the viewer. Use only the z-buffer, ignore the frame buffer.

2. Apply the z-buffer algorithm to the viewer with the following modification:

But before a projection is entered into the frame buffer F_V for the viewer, an illumination test is carried out to check whether the surface is illuminated by the considered light source.

© Springer

Two-pass z-buffer algorithm

Let T_V and T_L be the transformations turning the perspective projection with the viewer and the light source as the centre of projection into a parallel projection to the x/y-plane.

Coordinates of a point on the surface to be projected: (x_V, y_V, z_V).

$$\begin{pmatrix} x_L \\ y_L \\ z_L \end{pmatrix} = T_L \cdot T_V^{-1} \cdot \begin{pmatrix} x_V \\ y_V \\ z_V \end{pmatrix}$$

coordinates of the same point from the viewpoint of the light source.

© Springer

Two-pass z-buffer algorithm

If a smaller value than z_L is entered in the z-buffer Z_L at (x_L, y_L), then there must be an object between the light source and the considered surface so that this surface does not receive any light from this light source (shadow).

Otherwise, the considered point on the surface is illuminated by the light source.

© Springer

Transparency

Two characteristics of transparent surfaces:

- How much light (of which colour) can pass through the transparent surface?
- How strong is the refraction? (Refraction will not be discussed here.)

Contours of objects behind transparent surfaces are visible.

Translucent surfaces like milk glass let light pass through without showing the contours of objects behind.

© Springer

Interpolated transparency

Consider a surface F_2 positioned behind a transparent surface F_1.

Interpolated/filtered transparency: Intensity at pixel P:

$$I_P = (1 - k_{\text{transp}}) \cdot I_1 + k_{\text{transp}} \cdot I_2$$

- I_1: Intensity of the point on if F_1 would be treated like a normal nontransparent surface.
- I_2: Intensity of the point (on F_2) when F_1 would be completely invisible.
- $k_{\text{transp}} \in [0,1]$: **Transmission coefficient.**
  $k_{\text{transp}} = 1$: F_1 is completely transparent (invisible).
  $k_{\text{transp}} = 0$: F_1 is not transparent at all.

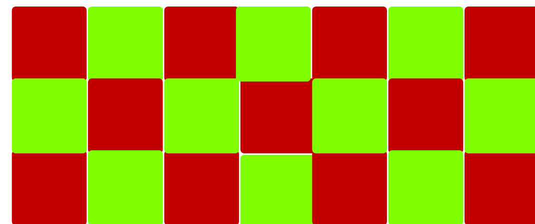
© Springer

Screen-door transparency

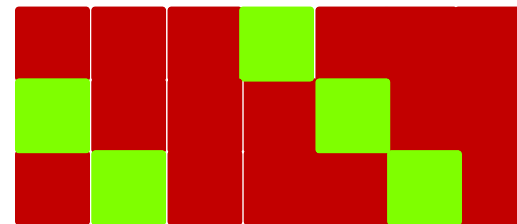
Screen-door transparency: Pixels obtain their colour alternatingly from the transparent surface and the surface behind.

Depending on the transmission coefficient, more or less pixels are assigned to the transparent surface.

transparent/background surface:



50% transparency



25% transparency

© Springer

Java 3D: Transparency

```
TransparencyAttributes ta =
    new TransparencyAttributes();

ta.setTransparencyMode(
   TransparencyAttributes.BLENDED);

ta.setTransparency(transpValue);

trApp.setTransparencyAttributes(ta);
```

`transpValue`: `float`-value between 0 and 1 defining the transmission coefficient.

© Springer

Java 3D: Transparency

Available `TransparencyMode`s:

- `BLENDED`: interpolated (filtered) transparency.
- `SCREEN_DOOR`: Screen-door transparency.
- Also: `NICEST, FASTEST, NONE`

(see `TransparencyExample.java`)

Textures

Texture: Image mapped to a surface for modelling surface details.

Images on surfaces: Examples:

- Painted wall
- Background landscape or sky with no explicit 3D modelling
- Wolrd map projected onto a globe

Patterns/structures: Examples:

- Wood grain
- Roughcast
- Ingrain wallpaper

Textures

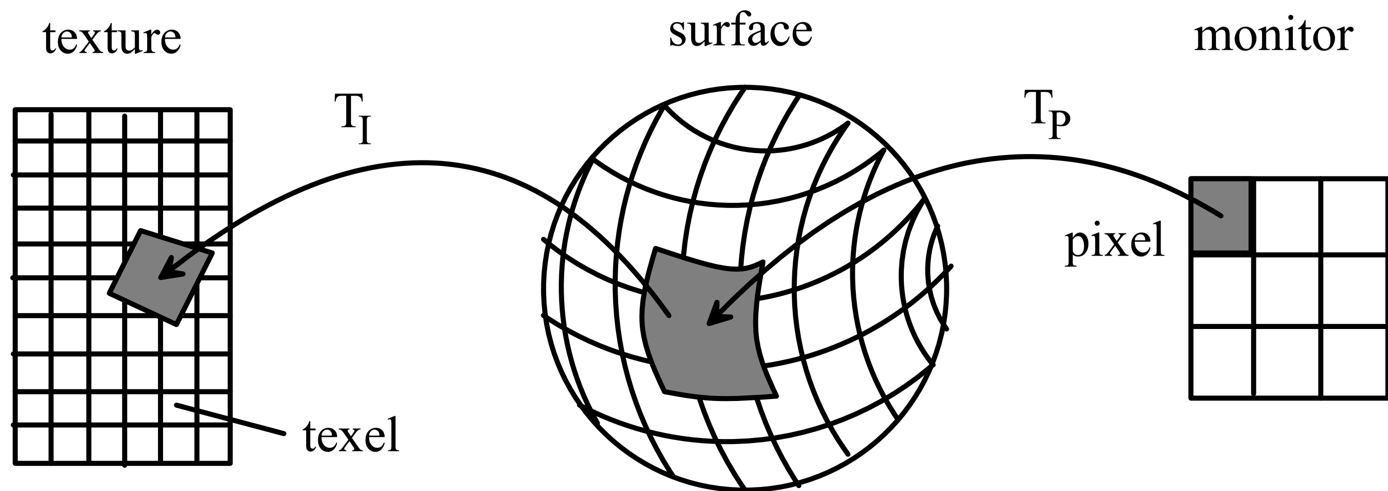
Textures are sometimes used for modelling small (wood grain) or distant (background landscape) three-dimensional structures.

Using explicit an 3D model based on extremely small polygons for such effects would be too inefficient.

Patterns or structures are usually mapped repeatedly onto a surface.

Concrete images or text is usually only mapped once to a surface.

Textures



Using a texture

© Springer

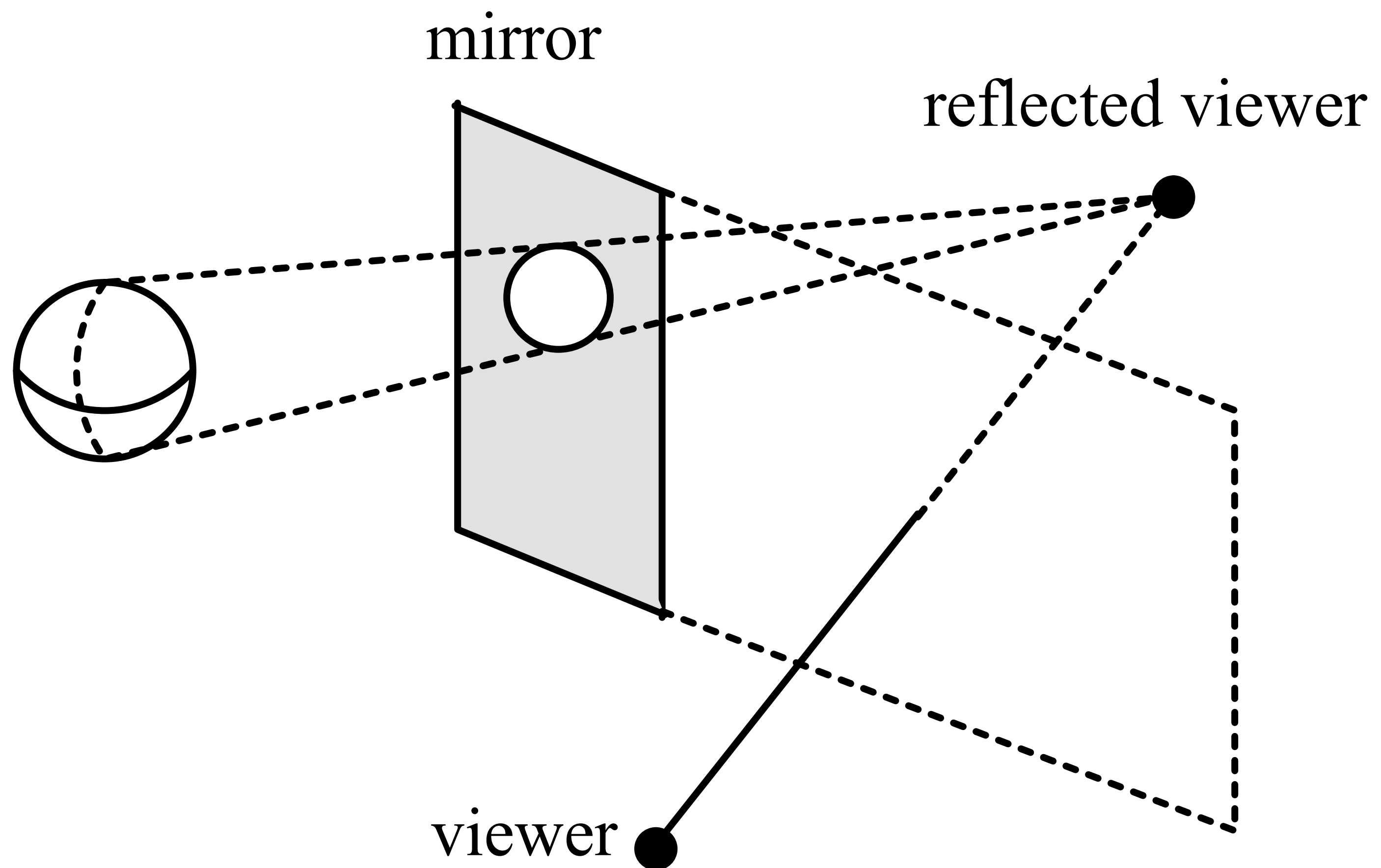
Textures

Environment or **reflection mapping**: Technique to model mirrors or reflecting surfaces like the surface of calm water.

- Reflect the viewer at the corresponding surface.
- Compute the image which the reflected viewer would see.
- Use this image as a texture for the reflecting surface.

© Springer

Environment mapping



Bump mapping

A surface remains flat when a texture is attached to it.

Bump mapping leaves the surface flat, but modifies the normal vectors.

A bump map assigns to each texture point a perturbation value $B(i,j)$ specifying how much the point on the surfaces should be moved along the normal vector for the relief.

If the surface is given in parametric form and the point to be modified is $P = P(x(s,t),\ y(s,t),\ z(s,t))$, then the nonnormalised modified normal vector at P is obtained from the cross product of the partial derivatives with respect to s and t.

© Springer

Bump mapping

$$\mathbf{n} \;=\; \frac{\partial P}{\partial s} \times \frac{\partial P}{\partial t}$$

If $B(T(P)) = B(i,j)$ is the corresponding bump value, one obtains

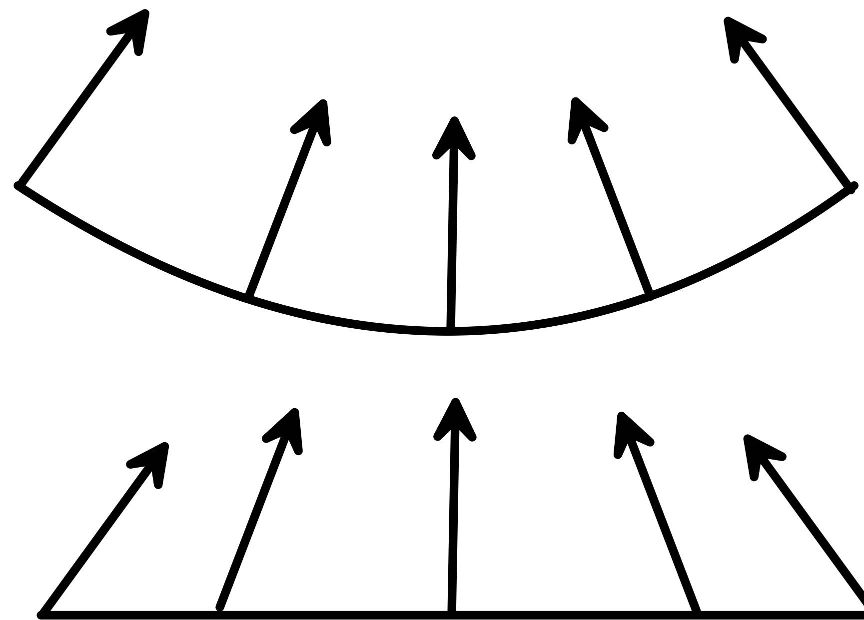
$$P' \;=\; P + B(T(P)) \cdot \frac{\mathbf{n}}{\|\,\mathbf{n}\,\|}.$$

Good approximation for the normal vector:

$$\mathbf{n}' \;=\; \frac{\mathbf{n}+\mathbf{d}}{\|\,\mathbf{n}+\mathbf{d}\,\|} \quad \text{where} \quad \mathbf{d} \;=\; \frac{\frac{\partial B}{\partial u}\cdot\left(\mathbf{n}\times\frac{\partial P}{\partial t}\right) - \frac{\partial B}{\partial v}\cdot\left(\mathbf{n}\times\frac{\partial P}{\partial s}\right)}{\|\,\mathbf{n}\,\|}$$

© Springer

Bump mapping



Generating a 3D impression using a bump mapping

© Springer

Java 3D: Textures

Load an image to an `ImageComponent2D`.

The height and the width of the `ImageComponent2D` must be powers of two.

```
TextureLoader textureLoad =
   new TextureLoader("image.jpg",null);

ImageComponent2D textureIm =
   textureLoad.getScaledImage(128,128);
```

© Springer

Java 3D: Textures

Generate a `Textur2D`, which can be assigned to an `Appearance`.

```
Texture2D myTexture =
    new Texture2D(
      Texture2D.BASE_LEVEL,Texture2D.RGB,
      textureIm.getWidth(),
      textureIm.getHeight());

myTexture.setImage(0,textureIm);

Appearance textureApp =
                new Appearance();
```

Java 3D: Textures

```
textureApp.setTexture(myTexture);

TextureAttributes textureAttr =
         new TextureAttributes();

textureAttr.setTextureMode(
      TextureAttributes.REPLACE);

textureApp.setTextureAttributes(
                    textureAttr);

textureApp.setMaterial(
                 new Material());
```

TextureExample.java

In principle, it can be specified in detail how to map the texture to the surface.

`TexCoordGeneration` offer a simple to map textures automaticaclly to surfaces.

```
TexCoordGeneration tcg =
new TexCoordGeneration(
TexCoordGeneration.OBJECT_LINEAR,
TexCoordGeneration.TEXTURE_COORDINATE_2);

textureApp.setTexCoordGeneration(tcg);
```

© Springer

Java 3D: Textures

Textures as background in Java 3D:

```
TextureLoader textureLoad =
 new TextureLoader("image.jpg",null);

Background bg =
 new Background(textureLoad.getImage());

bg.setApplicationBounds(bounds);
```

© Springer

Java 3D: Textures

`bounds` is, for instance, a `BoundingSphere` specifying the region where the texture should be used as background.

Add the background to the scene:

```
theScene.addChild(bg);
```

Instead of a background image, a colour can be assigned to the background:

```
Background bg = new Background(colour);
```

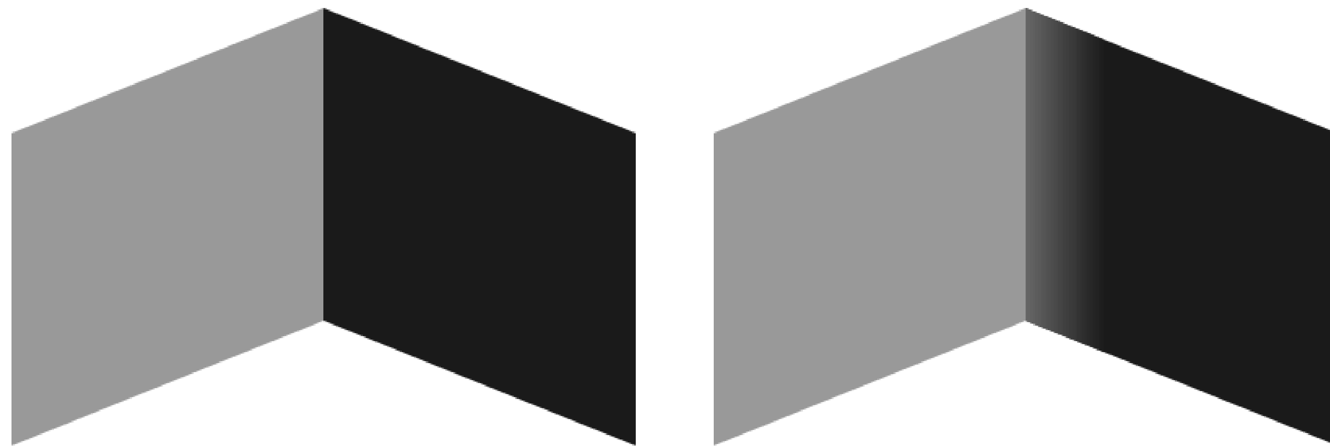
(see `BackgroundExample.java`)

© Springer

Radiosity model

So far: Illumination of objects only by explicit light sources and general ambient light, no reflection of light between objects.

Leads to sharp edges.



© Springer

Radiosity model

Radiosity: Rate of energy emitted by a surface O_i in the form of light.

Radiosity is the superposition of the self-emitted, reflected and, in case of transparency, light from the back.

Radiosity model: Each surface is considered as light source.

Interaction of the light between surfaces is independent of the position of the viewer when only diffuse reflection is considered.

© Springer

Radiosity equation

$$B_i \;=\; E_i + \varrho_i \cdot \sum_{j=1}^{n} B_j \cdot F_{ji} \cdot \frac{A_j}{A_i}$$

- B_i: Rate of energy coming from O_j measured in (energy/time)/area
- E_i: Rate at which light is emitted from surface O_i as an active light source (self-emitting).
- ϱ_i: Reflection coefficient of the surface O_i

© Springer

Radiosity equation

- F_{ji}: Dimensionless form and configuration factor taking the shape, the size and the relative orientation of the surfaces O_i and O_j into account.
- A_i: Area of the surface i

Transparency is not considered here.

© Springer

Radiosity equation

When only diffuse reflection is taken into account:

$$A_i F_{ij} \ = \ A_j F_{ji}$$

Therefore,

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

i.e.

$$B_i - \varrho_i \cdot \sum_{j=1}^{n} B_j \cdot F_{ij} \ = \ E_i.$$

© Springer

Radiosity equation

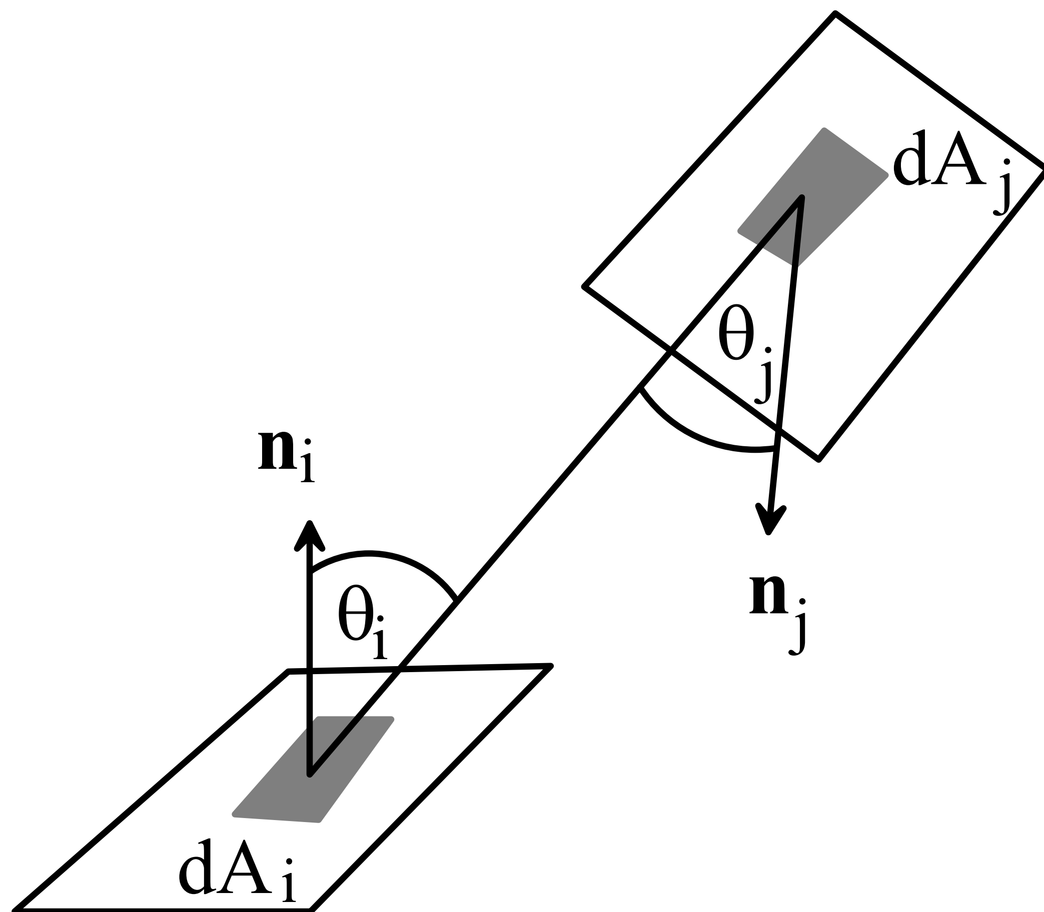
$$\begin{pmatrix} 1-\varrho_1 F_{1,1} & -\varrho_1 F_{1,2} & \dots & -\varrho_1 F_{1,n} \\ -\varrho_2 F_{2,1} & 1-\varrho_2 F_{2,2} & \dots & -\varrho_2 F_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ -\varrho_n F_{n,1} & -\varrho_n F_{n,2} & \dots & 1-\varrho_n F_{n,n} \end{pmatrix} \cdot \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{pmatrix}$$

This system of linear equations must be solved for the primary colours red, green and blue.

(ϱ_i and E_i depend on the colour.)

© Springer

Radiosity model



© Springer

Radiosity model

Form faktor from the patch (differential area) dA_i to the patch dA_j:

$$dF_{d_i,d_j} = \frac{\cos(\theta_i) \cdot \cos(\theta_j)}{\pi \cdot r^2} \cdot H_{ij} \cdot dA_j$$

$$H_{ij} = \begin{cases} 1 & \text{if } dA_j \text{ is visible from } dA_i \\ 0 & \text{otherwise} \end{cases}$$

© Springer

Radiosity model

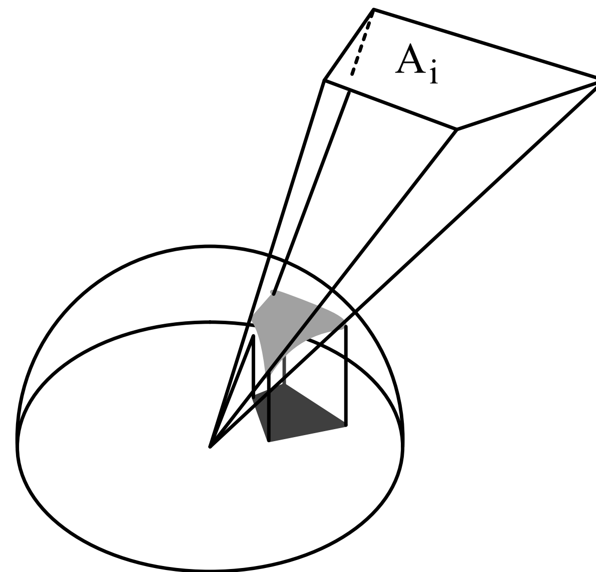
Form factor from the patch (differential area) dA_i to A_j:

$$dF_{d_i,j} = \int_{A_j} \frac{\cos(\theta_i) \cdot \cos(\theta_j)}{\pi \cdot r^2} \cdot H_{ij}\, dA_j$$

Form factor from the surface A_i to the surface A_j:

$$F_{i,j} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos(\theta_i) \cdot \cos(\theta_j)}{\pi \cdot r^2} \cdot H_{ij}\, dA_j\, dA_i$$

Radiosity model



Approximation of the form factor: Proportion of the circle that is covered by this projection.

© Springer

Radiosity model

Approximate solution of the system of linear equations of the radiosity equation: Stepwise refinement.

Iterative computation of the B_i.

1. Set each $B_i = E_i$ and $\Delta B_i = E_i$.
2. Choose O_{i_0} with the largest ΔB_{i_0}.
3. Update each B_i and ΔB_{i_0}

$$B_i^{\text{(new)}} = B_i^{\text{(old)}} + \varrho_i \cdot F_{i_0 i} \cdot \Delta B_{i_0}$$

$$\Delta B_i^{\text{(new)}} = \begin{cases} \Delta B_i^{\text{(old)}} + \varrho_i \cdot F_{i_0 i} \cdot \Delta B_{i_0} & \text{if } i \neq i_0 \\ 0 & \text{if } i = i_0 \end{cases}$$

© Springer

Radiosity model

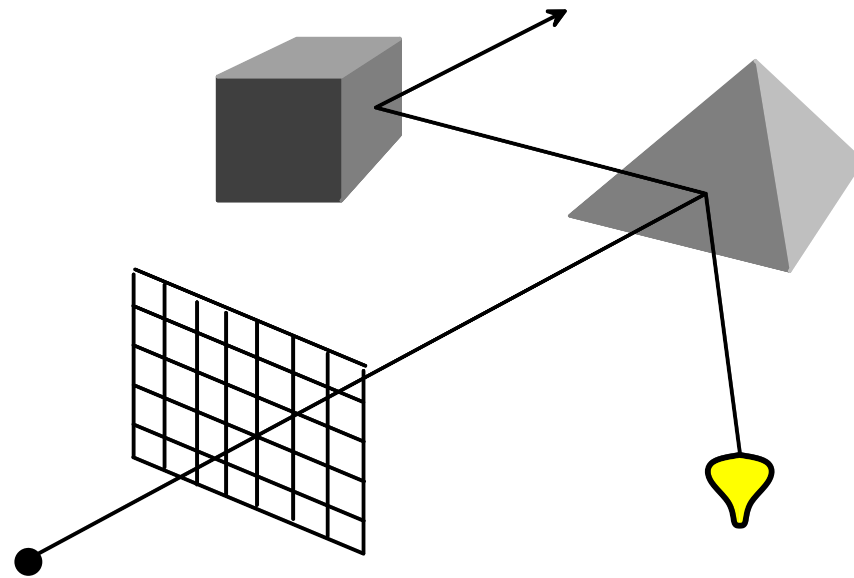
Repeat steps 2 and 3 until the changes are neglectable or the maximum number of iteration steps is reached.

Advantage of the stepwise refinement: Can be terminated any time and improves shading stepwise.

Specular reflection has to be added afterwards.

© Springer

Ray tracing



Recursive ray tracing for realistic specular reflection.

© Springer