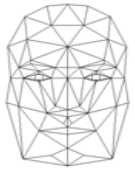


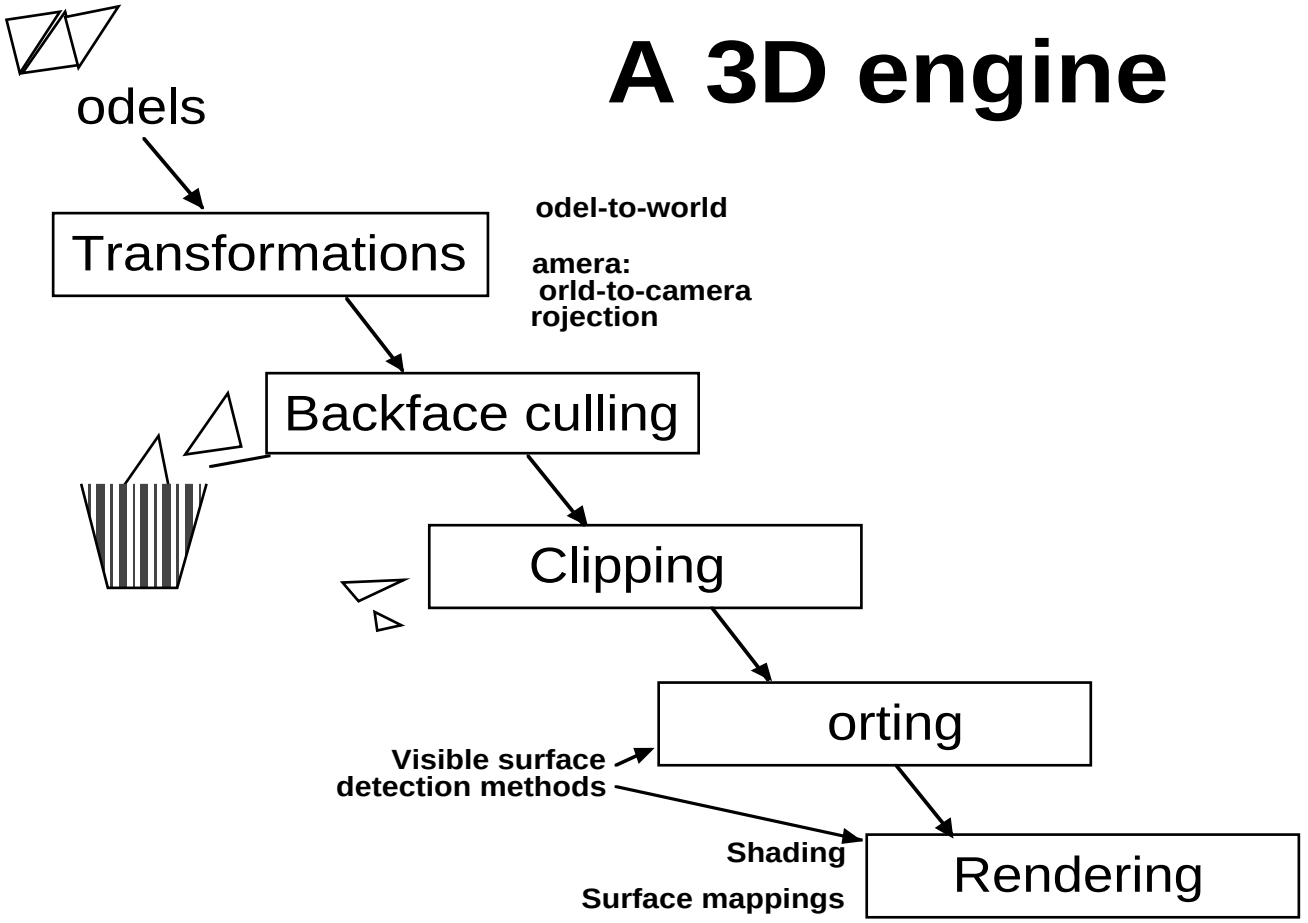
Lecture 4
3D graphics part 2

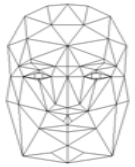
Today's topics:

Object representation (intro)

Visible surface detection (intro):
Z-buffer
Back-face culling

Illumination models, lighting

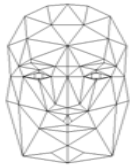

A 3D engine
odels
Transformations
odel-to-world
amera:
orld-to-camera
rojection
Backface culling
Clipping
orting
Visible surface
detection methods
Shading
Surface mappings
Rendering



3D object representation

Most common for real-time:
Polygon surfaces

- **Supported by graphics accelerators**
- **Easy to handle**
- **Not very space efficient**
- **Often triangle-based**



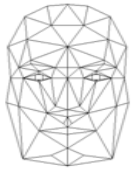
How do we store that?

First try: A simple format

Vertex = (x, y, z)

Triangle = array of Vertex

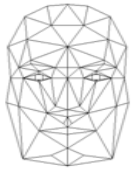
3DObject = array of Triangle

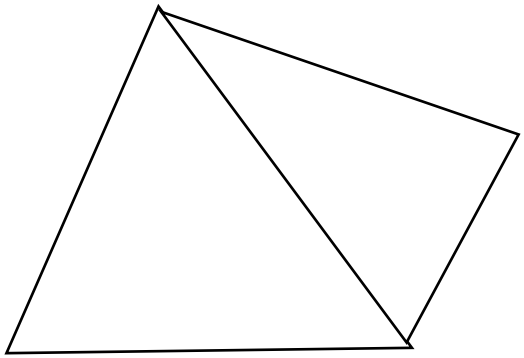


Example

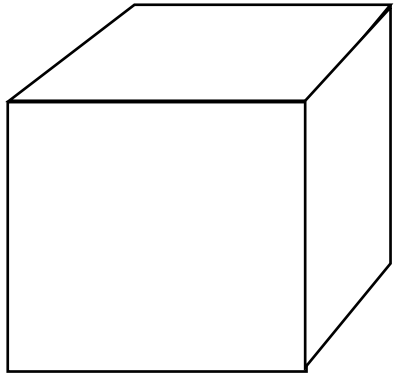
Tr iangle[1]: (10, 10, 10), (10, 20, 10) (10, 20, 20)
Triangle[2]: (10, 10, 10), 10, 20, 10), (10, 10, 20)
Triangle[3]: (. . .), (. . .), (. . .)
. . .

…but why is this not quite good?



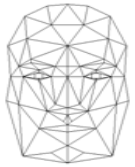


Pyramid:
4 triangles
4 corners
12 vertices!



Cube:
6 squares or 12 triangles
8 corners
24 or 36 vertices!

Several times too many vertices to process!



A better format

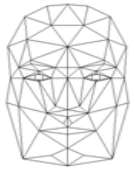
Vertex = (x, y, z)

Vertex table = array of Vertex

Triangle = array of integers

Triangle table = array of Triangles

3DObject = Vertex table + Triangle table



Example

Vertex table:
(10, 10, 10)
(10, 20, 10)
(10, 20, 20)
(10, 10, 20)
. . .

All vertices in the object in one list

Triangle table:
(1, 2, 3)
(1, 2, 4)
. . .

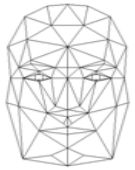
Indexes to the vertex table

Modelling in OpenGL

Lfloat vertices[] ={{-1,-1,-1}, {1,-1,-1}, {1,1,-1}, {-,1,-1},{-1,-1,1}, {1,-1,1}, {1,1,1}, {-1,1,1}};

Lubyte cubeIndices[24] ={0,3,2,1, 2,3,7,6, 0,4,7,3, ,2,6,5, 4,5,6,7, 0,1,5,4};

lDrawElements can draw the entire shape with ne call (almost)!



Models on disk

Wavefront .obj format. Simple, text-based mesh format. Example: A cube:

```
Exported from Wings 3D 0.98.26b
tllib cube.mtl
cube1
8 vertices, 6 faces
-1.00000000 -1.00000000 1.00000000
-1.00000000 1.00000000 1.00000000
1.00000000 1.00000000 1.00000000
1.00000000 -1.00000000 1.00000000
-1.00000000 -1.00000000 -1.00000000
-1.00000000 1.00000000 -1.00000000
1.00000000 1.00000000 -1.00000000
1.00000000 -1.00000000 -1.00000000
```

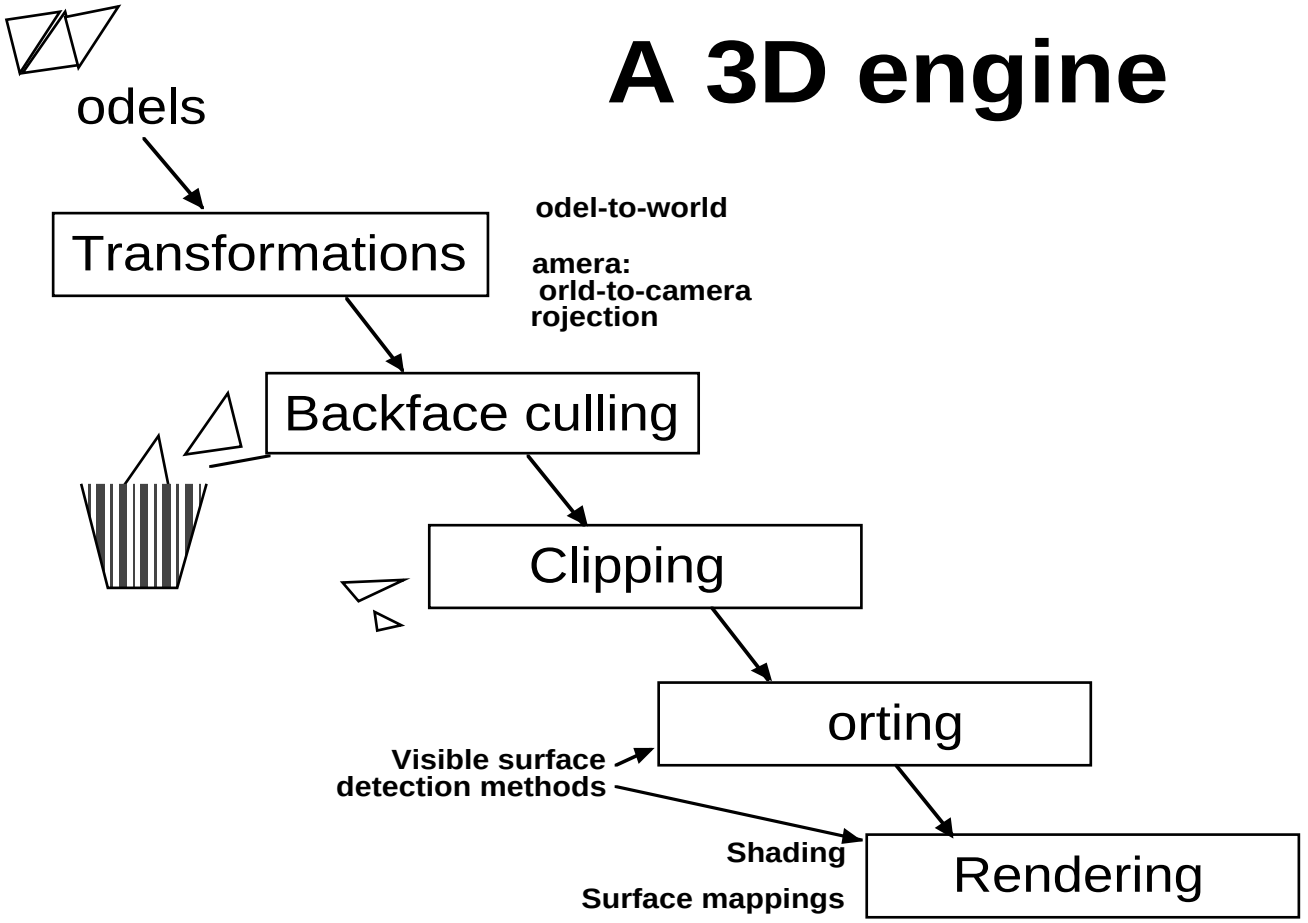
Vertex list

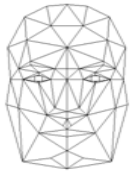
```
n -0.57735027 -0.57735027 0.57735027
n -0.57735027 0.57735027 0.57735027
n 0.57735027 0.57735027 0.57735027
n 0.57735027 -0.57735027 0.57735027
n -0.57735027 -0.57735027 -0.57735027
n -0.57735027 0.57735027 -0.57735027
n 0.57735027 0.57735027 -0.57735027
n 0.57735027 -0.57735027 -0.57735027
```

Normal vectors list

```
cube1_default
semtl default
3//3 2//2 1//1 4//4
5//5 1//1 2//2 6//6
6//6 2//2 3//3 7//7
7//7 3//3 4//4 8//8
8//8 4//4 1//1 5//5
8//8 5//5 6//6 7//7
```

Polygon list

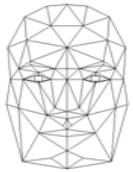

A 3D engine
odels
Transformations
odel-to-world
amera:
orld-to-camera
rojection
Backface culling
Clipping
orting
Visible surface
detection methods
Shading
Surface mappings
Rendering



Detection of visible surfaces

ackface culling
-buffer
ainter’s algorithm
SP trees
can-line method

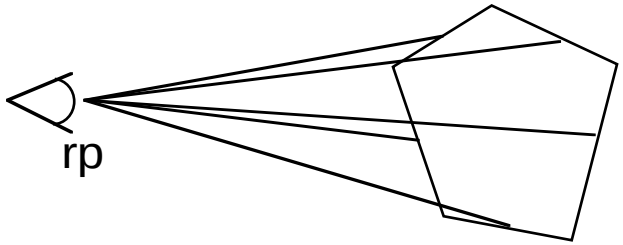
ay-casting
-buffer
rea subdivision
ctrees
ortals



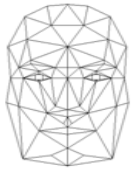
Backface culling

bject space method

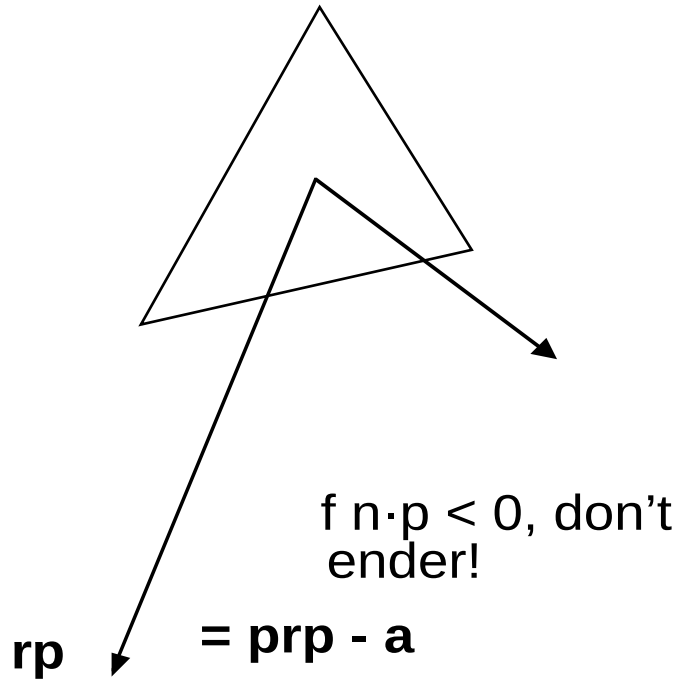
emove all polygons that are “looking away” rom the camera



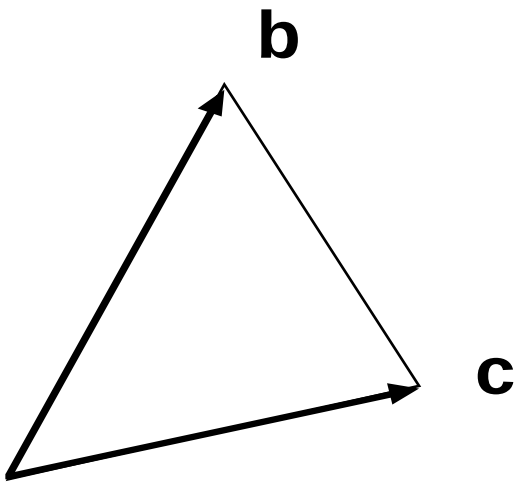
emoves ≈ 50% of all polygons that would therwise be in view.



Backface culling in camera (view) space

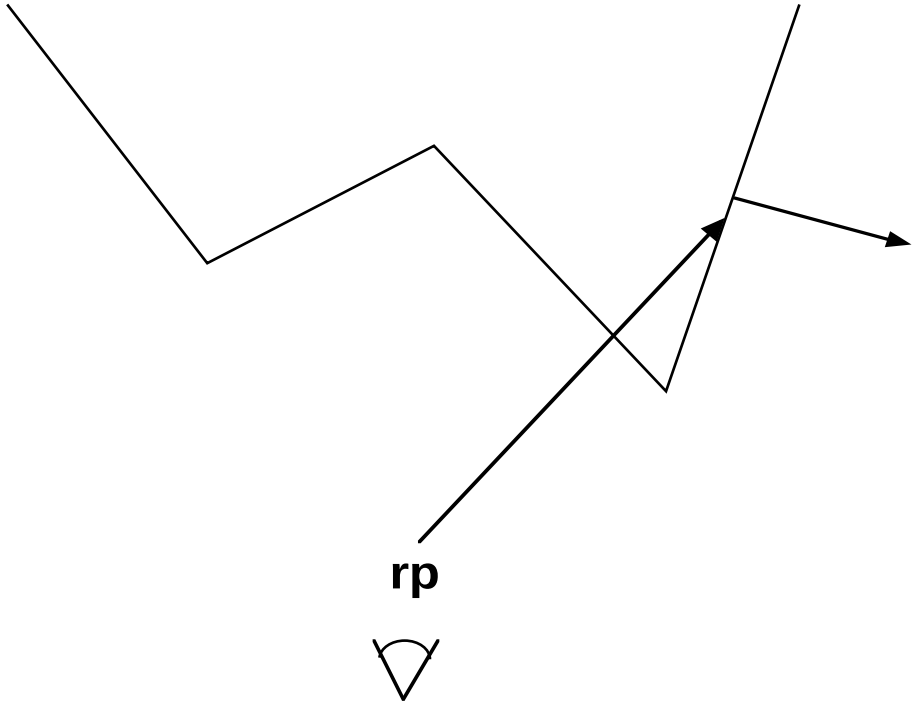


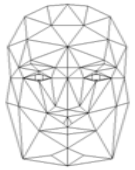
Backface culling in screen space



Cross product produces a z component
If z < 0, don’t render!

You can not take the z component in camera space (viewing coordinates)!





Z-buffer

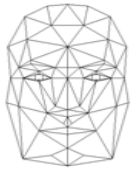
Depth-buffer method

mage-space method

Z” since we usually look along Z axis.

n extra “depth image” buffer is used, the Z buffer.

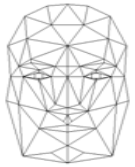
he Z buffer holds a Z value for every pixel in the mage. The Z value corresponds to the nearest object ritten so far, that has touched that pixel.



Z-buffer algorithm

nitialize Z-buffer to infinite distance and the image buffer to
ackground.

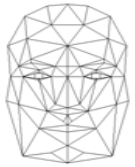
or each polygon
    for each pixel (x,y) in the polygon
        calculate z value
        if z closer than the current z-buffer value Z(x,y)
            write pixel to image (x,y)
            write z to z-buffer Z(x,y)



Calculation of Z

values must survive the projection - as pecified in previous lecture

values are calculated for each vertex and nterpolated over the surface



Culling and Z-buffer in OpenGL

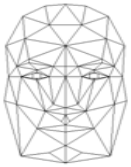
lEnable(GL_CULL_FACE);

lCullFace(GL_BACK);
lCullFace(GL_FRONT);

nitialize context with depth buffer, e.g. GLUT_DEPTH

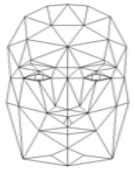
lEnable(GL_DEPTH_TEST);

lClear(GL_DEPTH_BUFFER_BIT);



When will culling + Z-buffering not be enough?

- Can not handle transparency
- High-level methods are needed to reduce the amount of data, to avoid passing unseen surfaces to the OpenGL pipeline



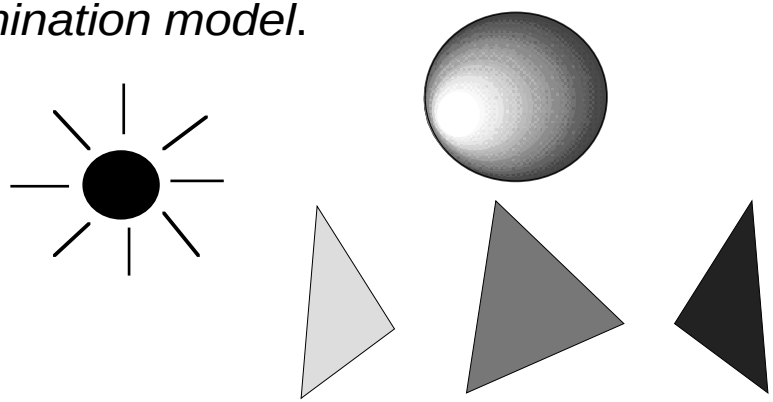
Illumination

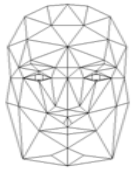
e know *where* to put a polygon. Now, *what* should we fill it with? What ixel value should we choose?

everal factors to take into account. The most important ones:

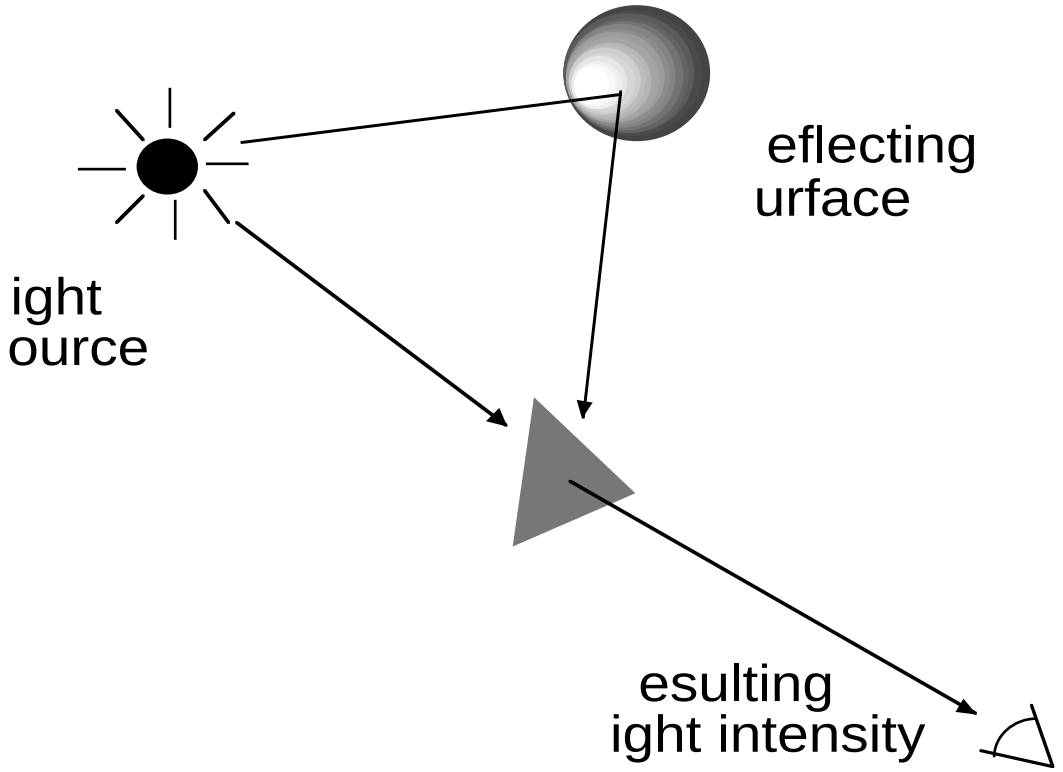
Shading, illumination.
Texture mapping.

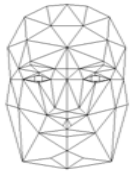
hading is determined according to an *illumination model*.



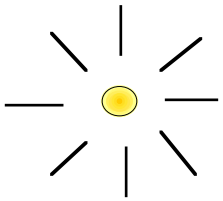


Light sources





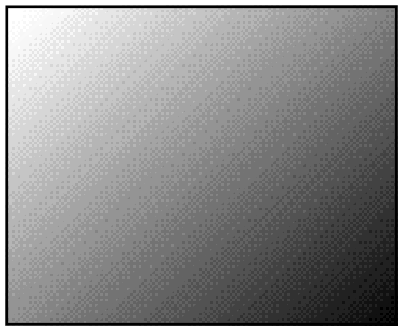
Light sources



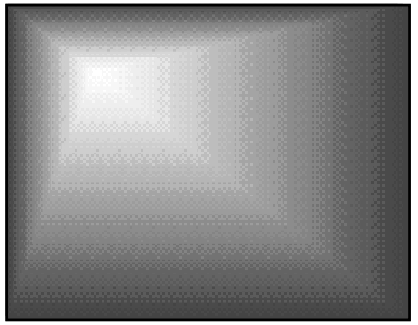
mall sourcescan
e modelled
s**point light**
ource

thers can be modelled
s**distributed light**
ources

Reflections

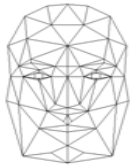


iffuse reflection



pecular reflection

aper lastic etal irror



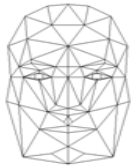
3-component illumination model

A common simple illumination model is built from three components:

Ambient light

Diffuse reflections

Specular reflections



Ambient light

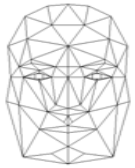
Same everywhere!

$$I_{amb} = k_d * I_a$$

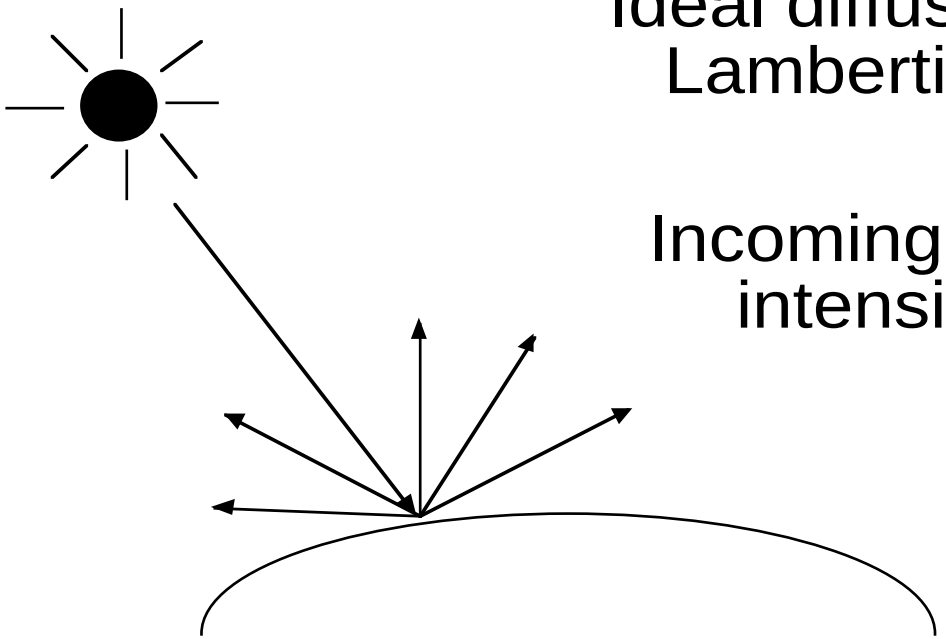
ight emitted
rom surface

iffuse
eflectivity

mbient light
evel of scene



Diffuse reflections

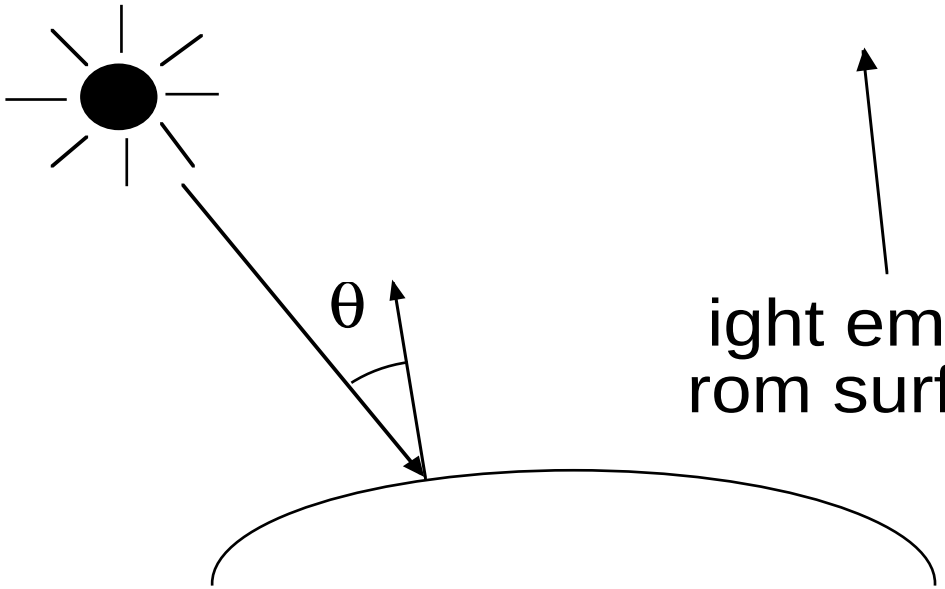


Ideal diffuse reflector =
Lambertian surface

Incoming light produces same intensity in all directions!

Lambert's cosine law

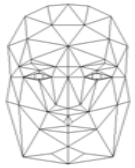
$$I_{diff} = k_d * I_l * \cos\theta$$



ight emitted
rom surface

iffuse
eflectivity

ight level
f light
ource

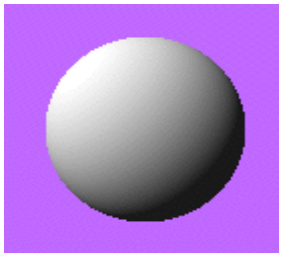


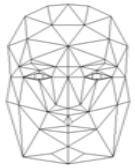
Example: Diffuse sphere

$k_d = 1$

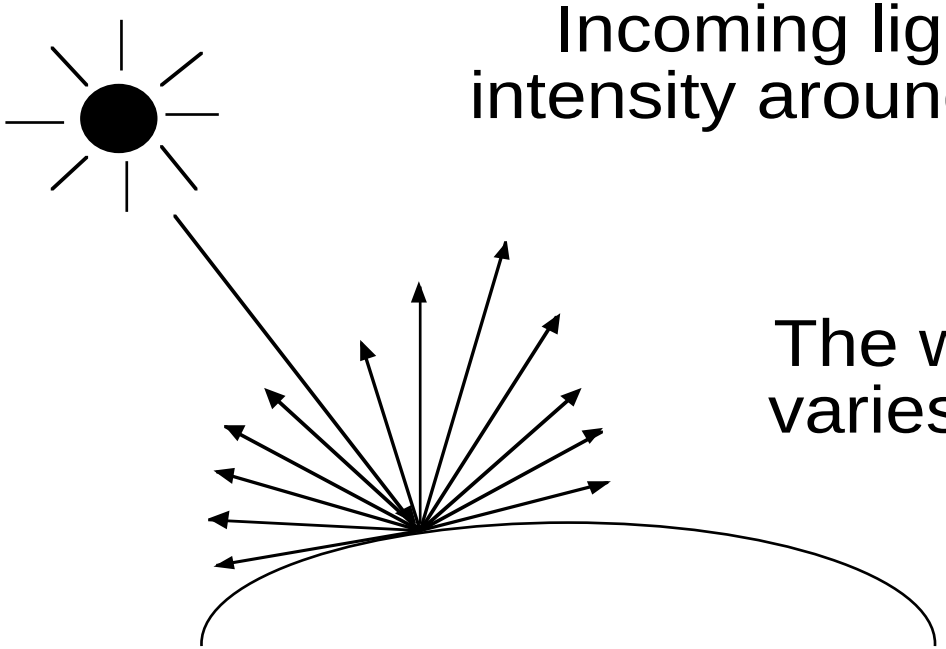
$I_l = 0.9$

$I_a = 0.1$



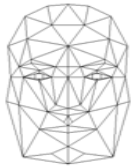


Specular reflections

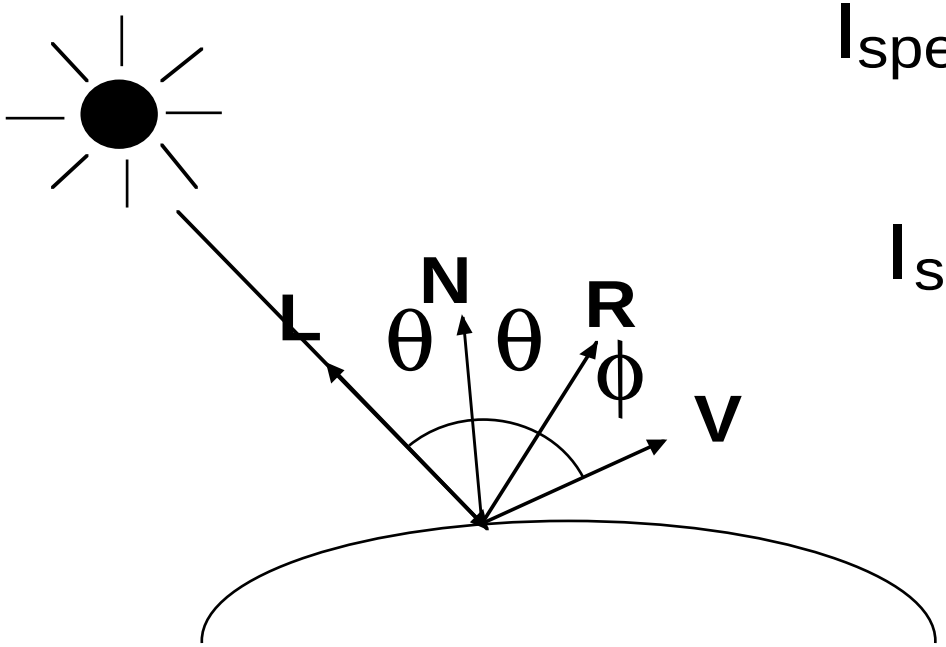


Incoming light produces higher intensity around the mirroring angle!

The width of the highlight varies with surface types!



The Phong model

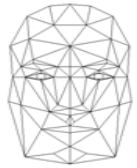


$$I_{spec} = W(\theta) * I_l * \cos^n \phi$$

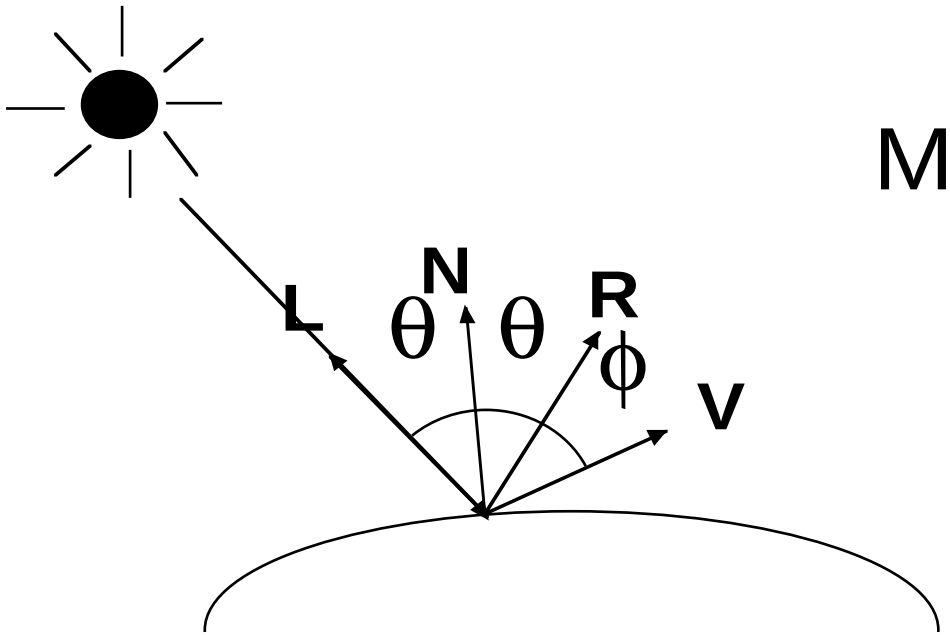
or

$$I_{spec} = k_s * I_l * \cos^n \phi$$

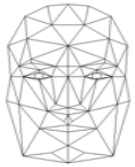
$$\cos \phi = R \cdot V$$



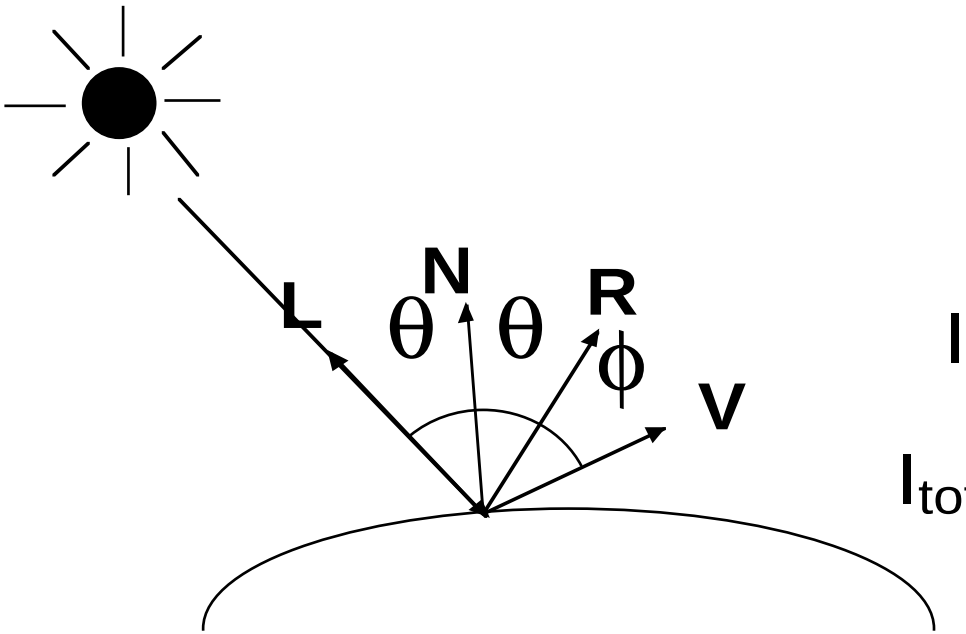
Calculation of R



Mirror L by N!



Total contribution from one surface

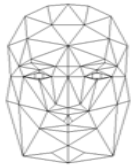


$$I_{amb} = k_d * I_a$$

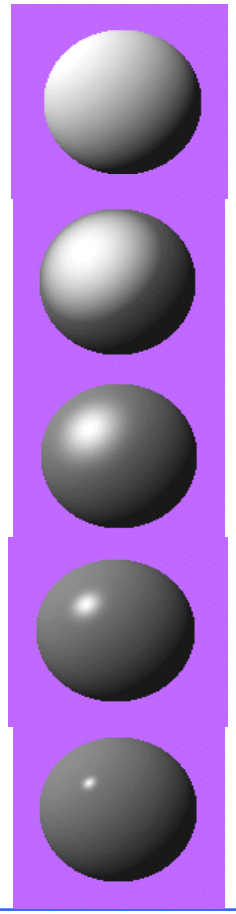
$$I_{diff} = k_d * I_l * \mathbf{L} \cdot \mathbf{N}$$

$$I_{spec} = k_s * I_l * (\mathbf{R} \cdot \mathbf{V})^n$$

$$I_{total} = I_{amb} + I_{diff} + I_{spec}$$



Examples



iffuse surface, $k_d = 0.9$

pecular surface, n = 1

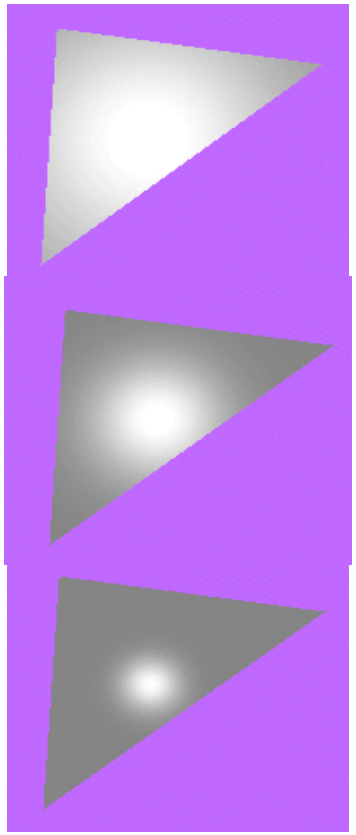
pecular surface, n = 5

pecular surface, n = 25

pecular surface, n = 125

$_d = 0.45$
$_s = 0.5$
$_l = 1.0$
$_a = 0.1$

Examples

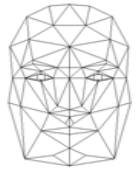


pecular surface, n = 5

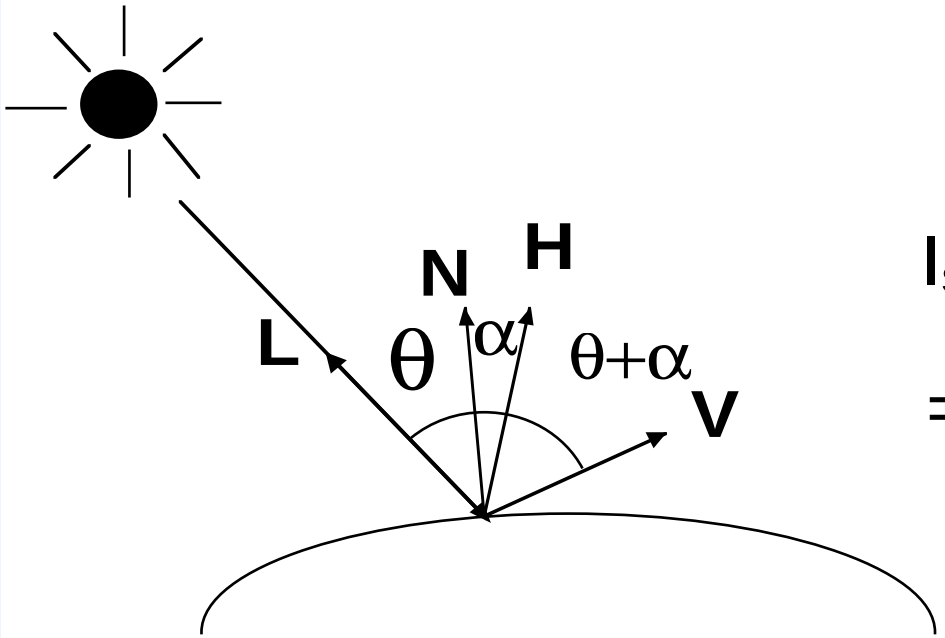
pecular surface, n = 25

pecular surface, n = 125

$_d = 0.45$
$_s = 0.5$
$_l = 1.0$
$_a = 0.1$



Alternative formulation



Halfway vector

$$\mathbf{H} = (\mathbf{L}+\mathbf{V}) / |\mathbf{L} + \mathbf{V}|$$

$$I_{spec} = k_s * I_l * \cos^n \alpha =$$

$$= I_{spec} = k_s * I_l * (\mathbf{N} \cdot \mathbf{H})$$