Computer Graphics:

Visualisation – Lecture 3

Taku Komura
tkomura@inf.ed.ac.uk

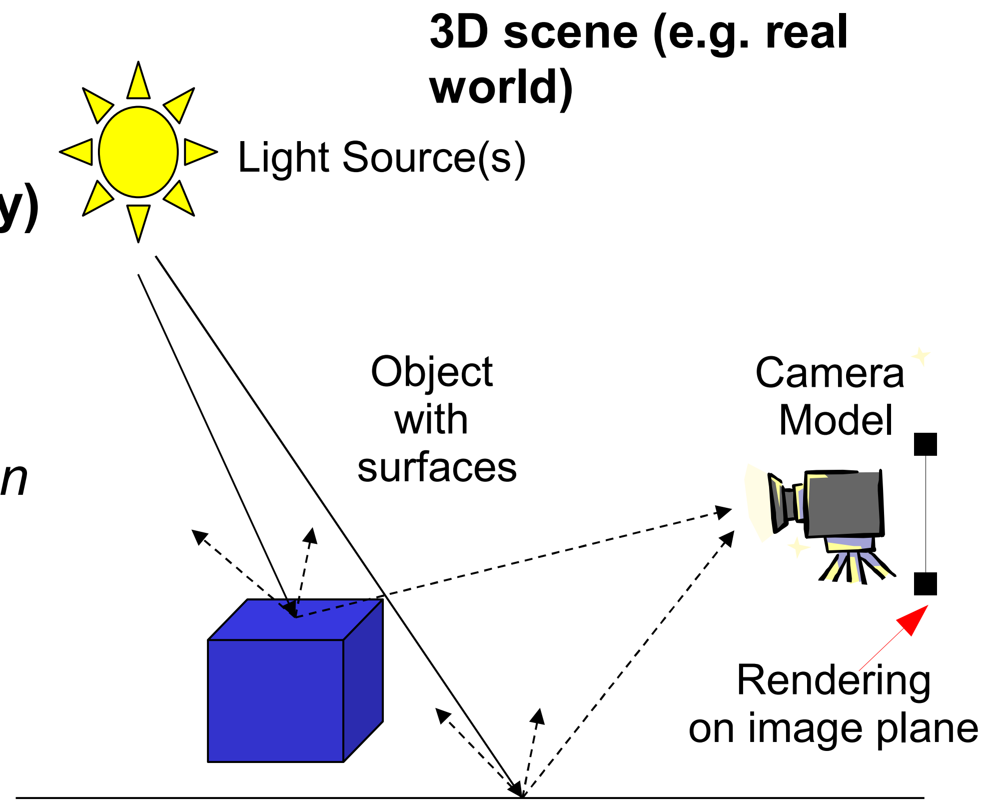
Institute for Perception, Action & Behaviour

Last lecture .....

- **Visualisation** can be greatly enhanced through the use of **3D computer graphics**
  - computer graphics are our **tool in visualisation**
- In order to do effective visualisation we need:
  - to know some computer graphics *(this lecture)*

Computer Graphics : simulation of light behaviour in 3D

- Effective simulation requires to model:
  - object representation **(geometry)**
  - object illumination **(lighting)**
  - camera model **(vision)**
    - *world to image plane projection*
    - ***rendering:** converting graphical data into an image*

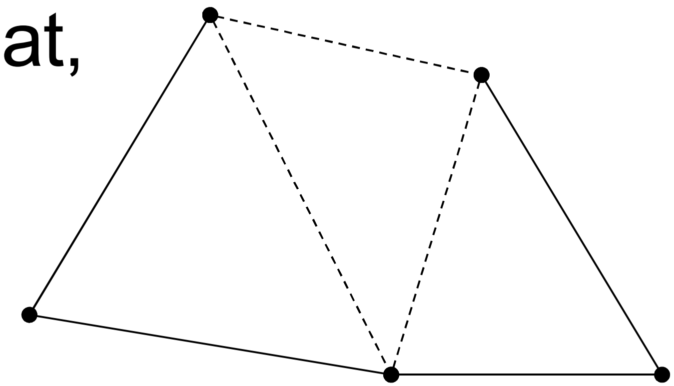


Overview of Computer Graphics

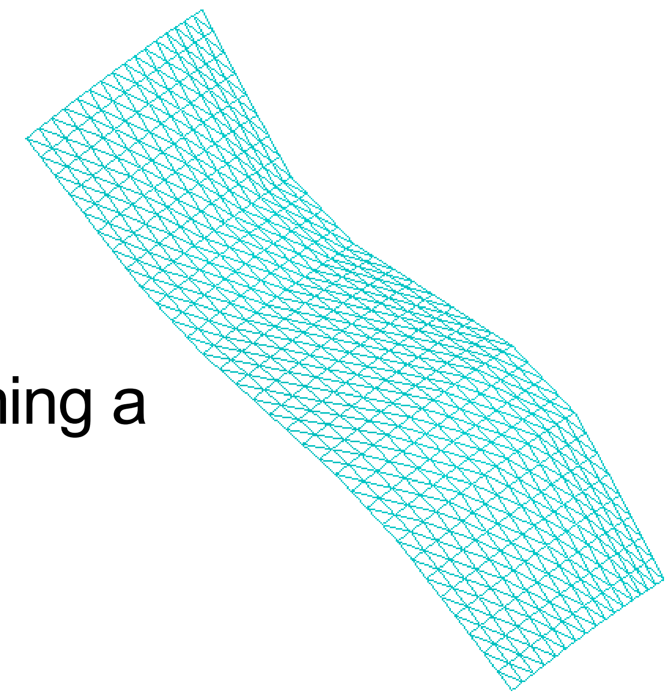
- Data representation
- Lighting
  - illumination
    - Ambient
    - Diffuse
    - Specular
  - Shading
  - Lighting and surface shape perception
- Camera model

Data Representation : 3D shape

- Approximate smooth surfaces with flat, planar polygons
  - polygons formed of edges & vertices

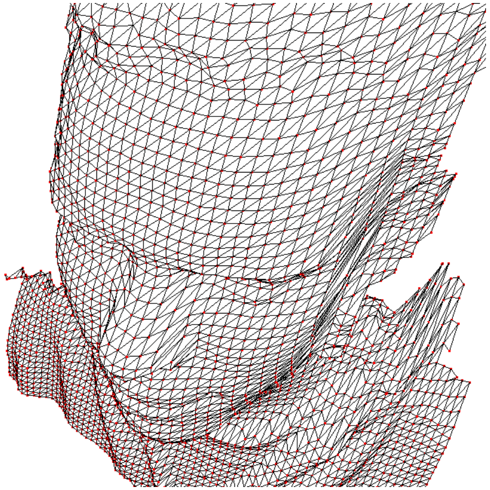


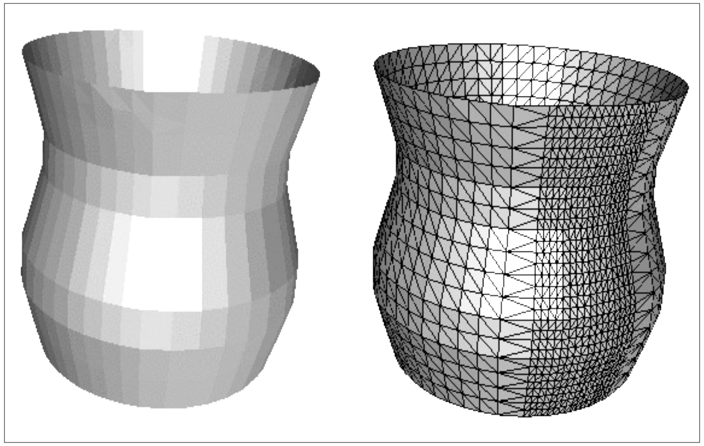
  - **vertex:** positional point (2D or 3D)
  - **edge:** joins 2 vertices
  - **polygon:** enclosed within N edges
    - polygons share common edges
  - **mesh:** set of connected polygons forming a surface (or object)

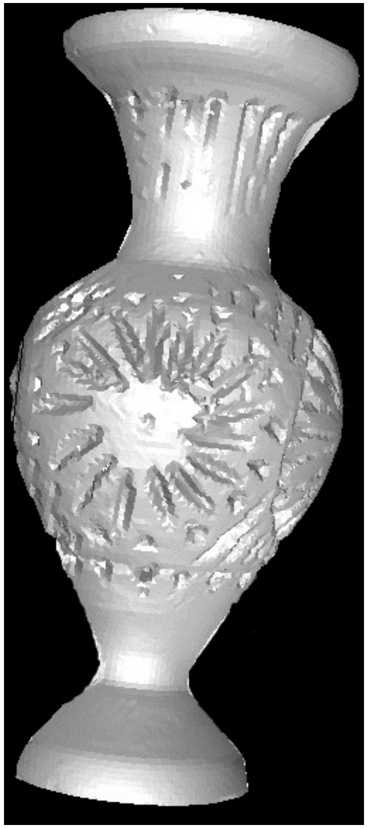


Hierarchy of Surface Representation

Surface mesh : examples



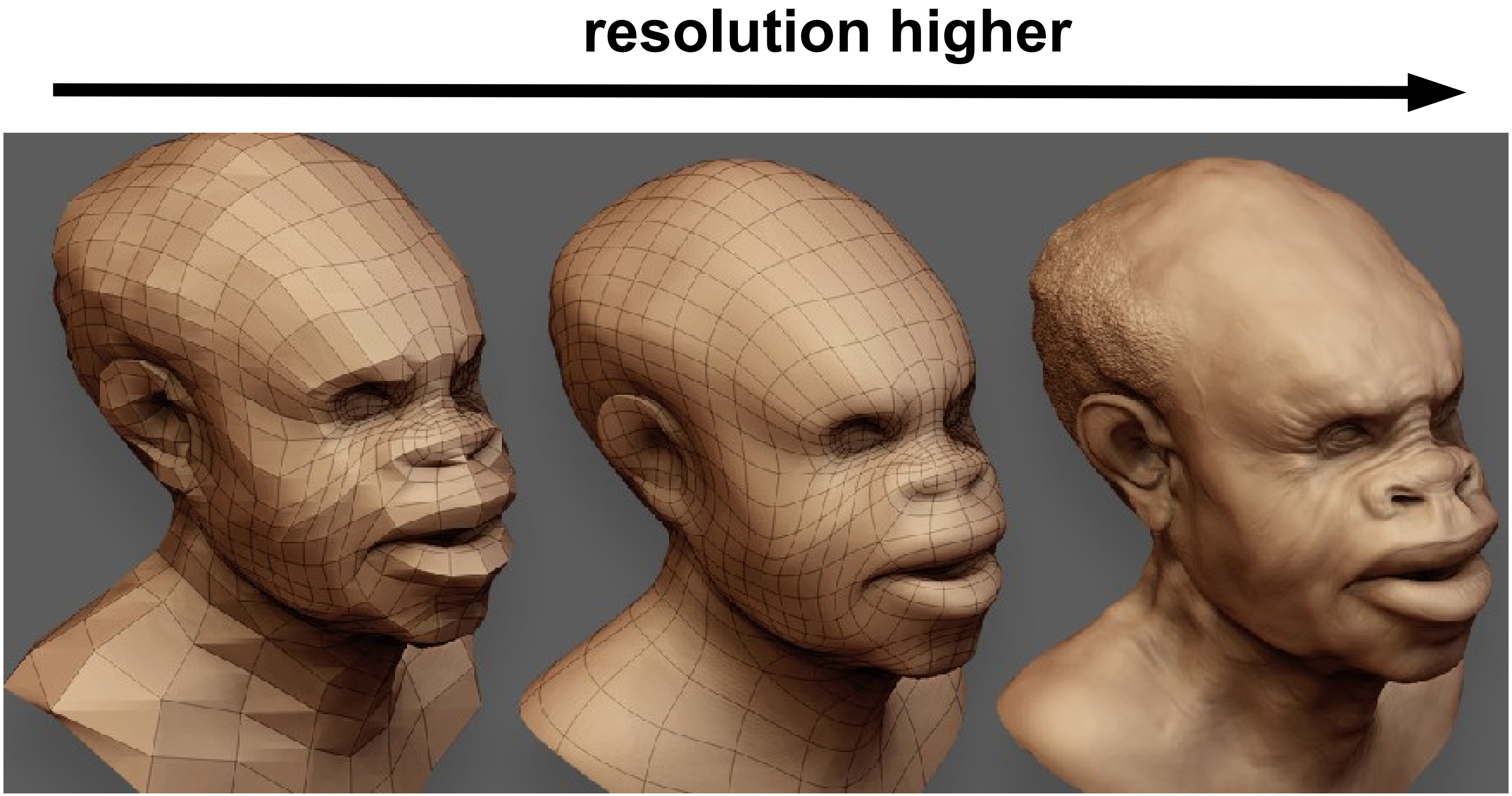




(close up)

- Several **primitives** utilised, triangles generally fastest to draw
  - modern graphics cards : 20-225 million + triangles per second

Different resolutions alter perception of surface smoothness

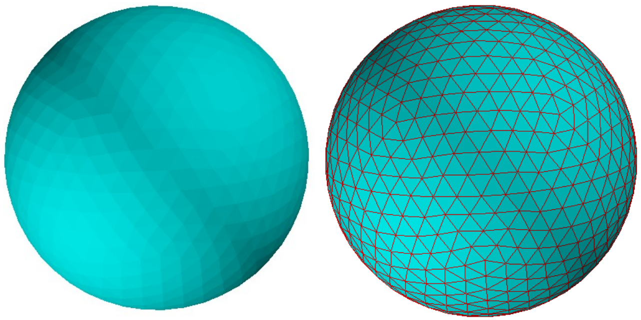


Mesh Based Representation

- 3D file formats:
  - set of vertices in $\mathbb{R}^3$
  - polygons reference into vertex set
    - implicitly define edges
  - e.g.

```
vertex 0 0 0
vertex 0 1 0
....
polyon 3 2 1 3
polyon 3 5 6 8
...
```

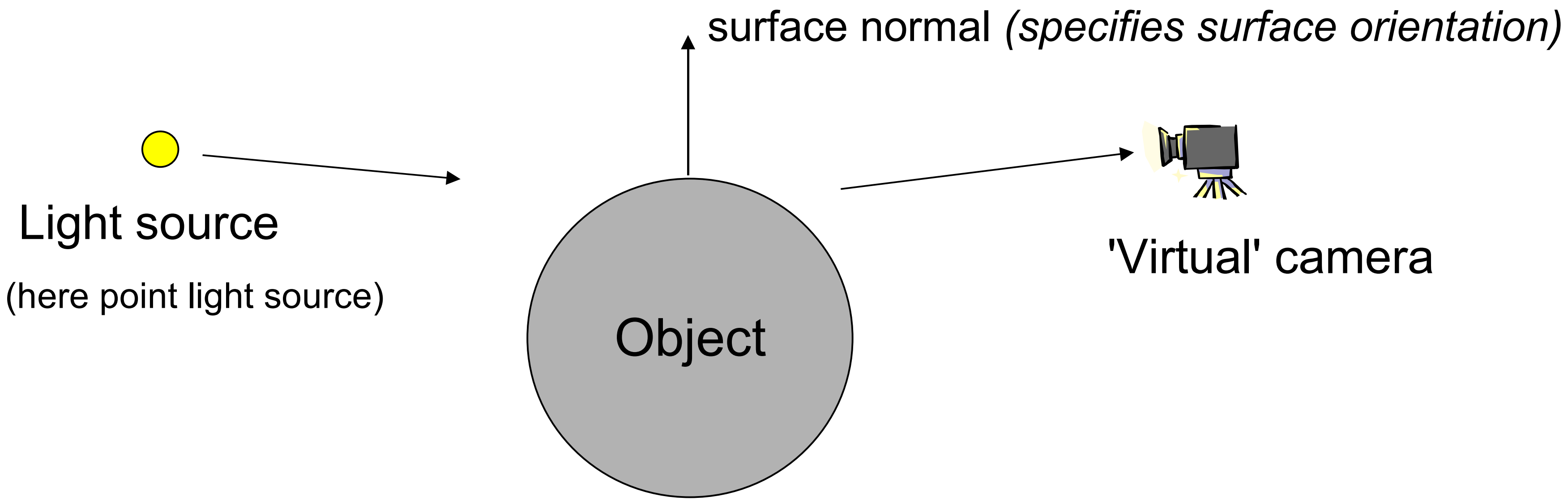
- perform transformations only on vertices



```
#VRML V1.0 ascii
#
Separator {
        Material {
                ambientColor 0.2 0.2 0.2
                diffuseColor 1.0 1.0 1.0
        }
        Coordinate3 {
                point [
                        4.455030 -1.193380 1.930940,
                        4.581220 -1.506290 1.320410,
                        4.219560 -1.875190 1.918070,
                        3.535530 1.858740 -3.007500,
                        3.793260 1.185430 -3.034130,
                        4.045080 1.545080 -2.500000,
                        3.510230 3.468900 0.803110,
                        3.556410 3.514540 0.000000,
                        3.919220 3.078210 0.405431,
                        ....
                        ....
        IndexedFaceSet {
                coordIndex [
                        0, 1, 2, -1,
                        3, 4, 5, -1,
                        6, 7, 8, -1,
                        9, 10, 11, -1,
                        12, 13, 14, -1,
                        15, 16, 17, -1,
                        18, 19, 20, -1,
                        21, 22, 23, -1,
                        ....
                        ....
```

Light interaction with surfaces

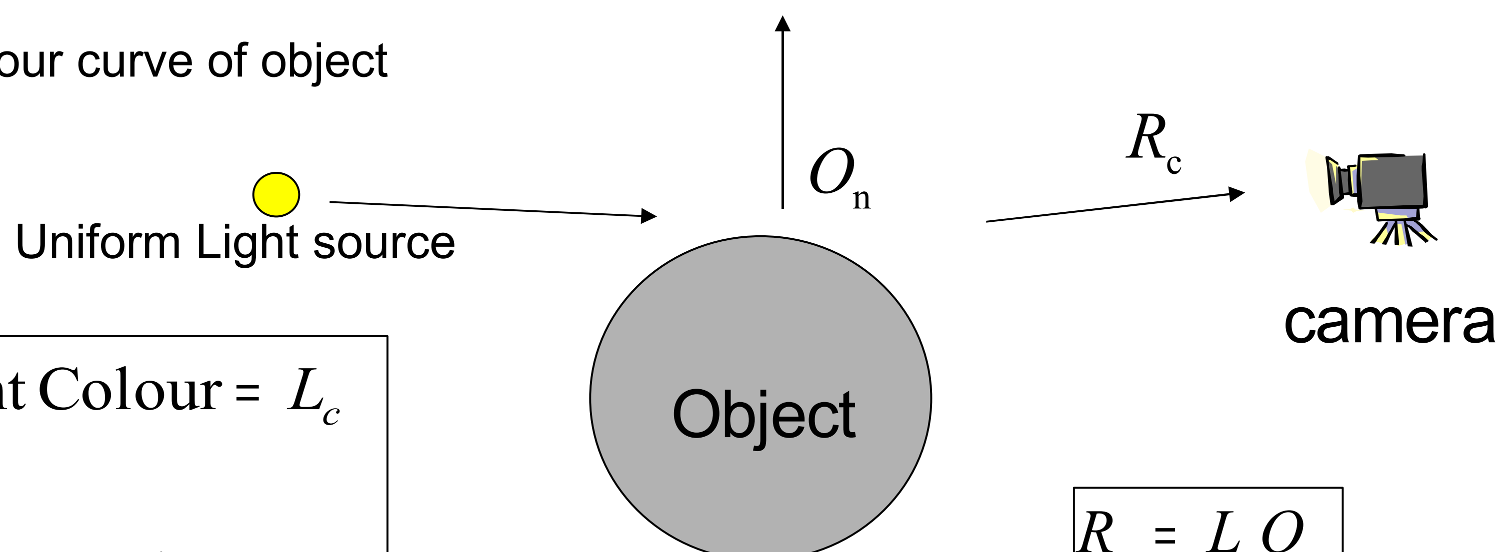
- Simple 3 parameter model
  - The sum of 3 illumination terms:
    - **Ambient :** 'background' illumination
    - **Specular :** bright, shiny reflections
    - **Diffuse :** non-shiny illumination and shadows



Ambient Lighting

- Light from the environment
- light reflected or scattered from other objects
- simple approximation to complex 'real-world' process
- Result: **globally uniform colour for object**
  - R_c = resulting intensity curve
  - L_c = light intensity curve
  - O_c = colour curve of object

Example: sphere

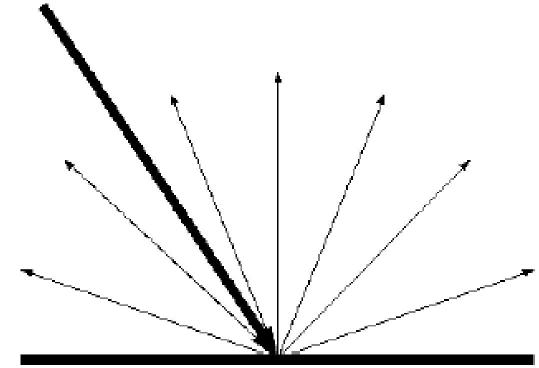


$$\text{Light Colour} = L_c$$

$$\text{Object Colour} = O_c$$

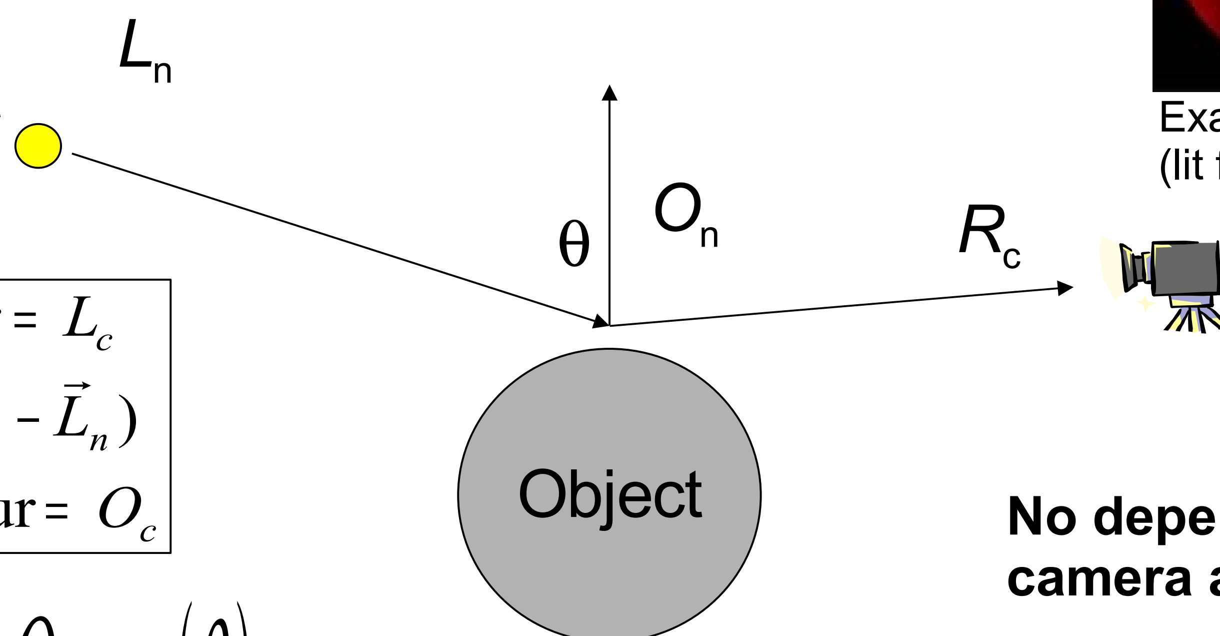
$$R_c = L_c O_c$$

Diffuse Lighting



- Also known as Lambertian reflection
  - considers the angle of incidence of light on surface
    (angle between light and surface normal)
  - Result: **lighting varies over surface with orientation to light**

Example: sphere
(lit from left)



$$\text{Light Colour} = L_c$$

$$\cos\theta = (\vec{O}_n \bullet -\vec{L}_n)$$

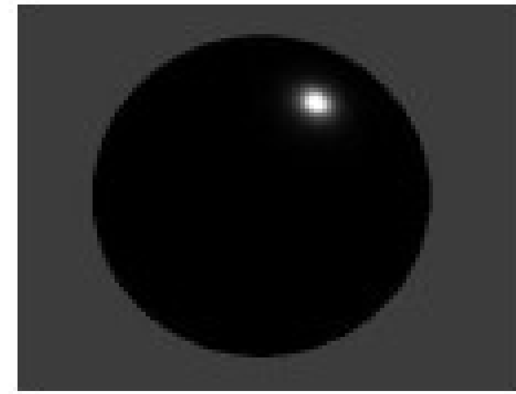
$$\text{Object Colour} = O_c$$

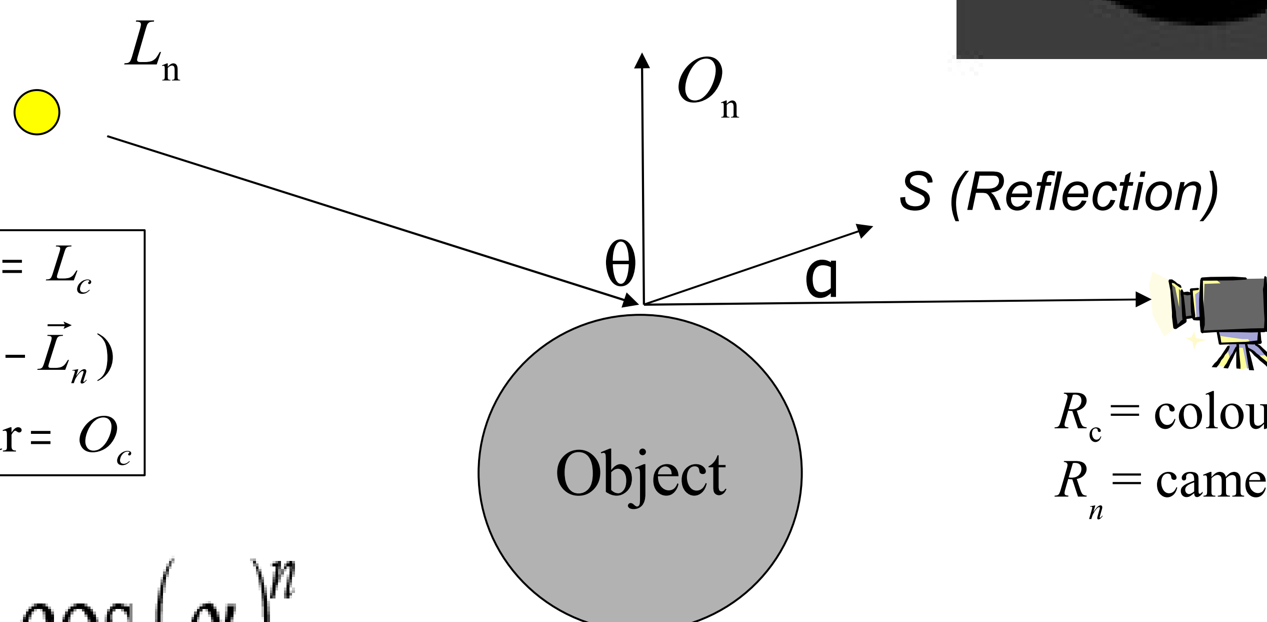
$$R_c = L_c O_c \cos(\theta)$$

No dependence on camera angle!

Specular Lighting

- Direct reflections of light source off shiny object
  - specular intensity *n* = shiny reflectance of object
  - Result: **specular highlight on object**





Light Colour = L_c

$\cos\theta = (\vec{O}_n \bullet -\vec{L}_n)$

Object Colour = O_c

R_c = colour curve

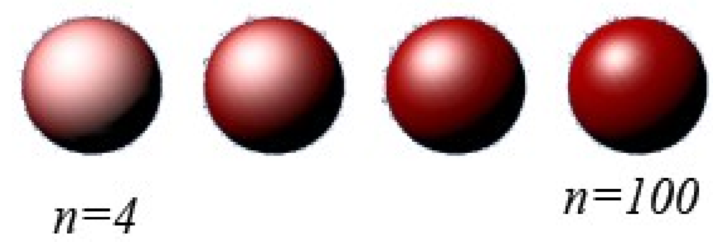
R_n = camera position

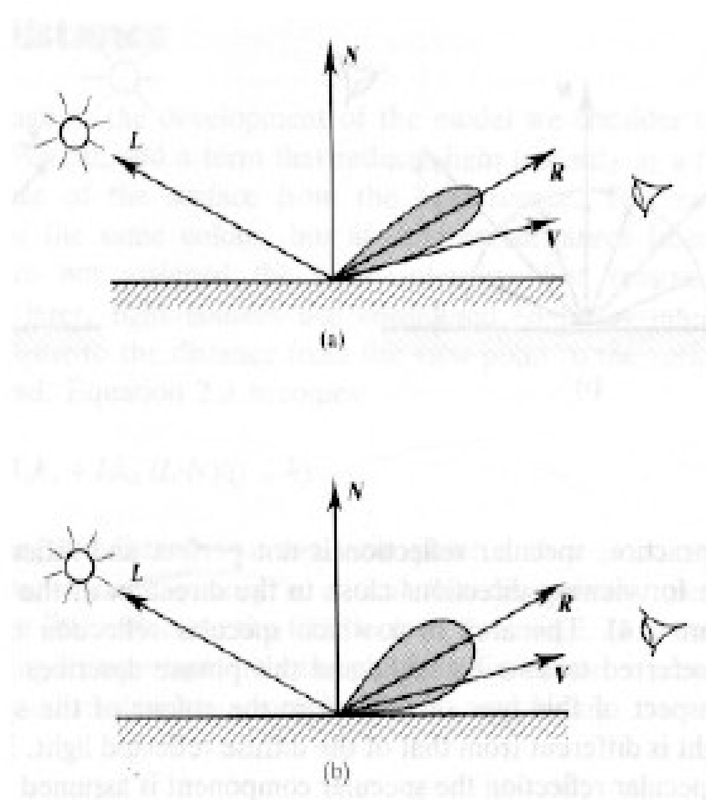
$$R_c = L_c k_s \cos(\alpha)^n$$

No dependence on object colour.

Specular Light

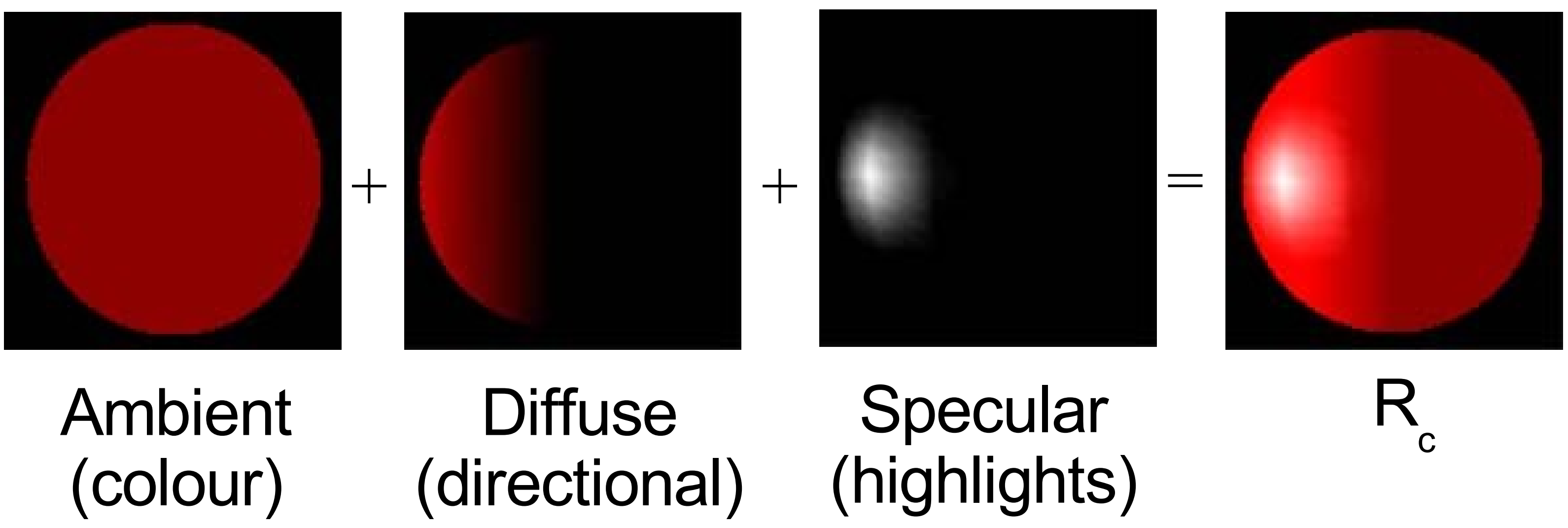
- Specular light with different *n* values





Combined Lighting Models

- $R_c = w_a(\text{ambient}) + w_d(\text{diffuse}) + w_s(\text{specular})$
  - for relative weights w_a, w_d, w_s
  - also specular power *n*



Are we supposed to do the computation of lighting at all the points over the surface?

- Depends on the shading model
  - Flat Shading (once per polygon)
  - Gouraud shading (for all the vertex of the polygon)
  - Phong Shading (all the points)

Local Shading Models

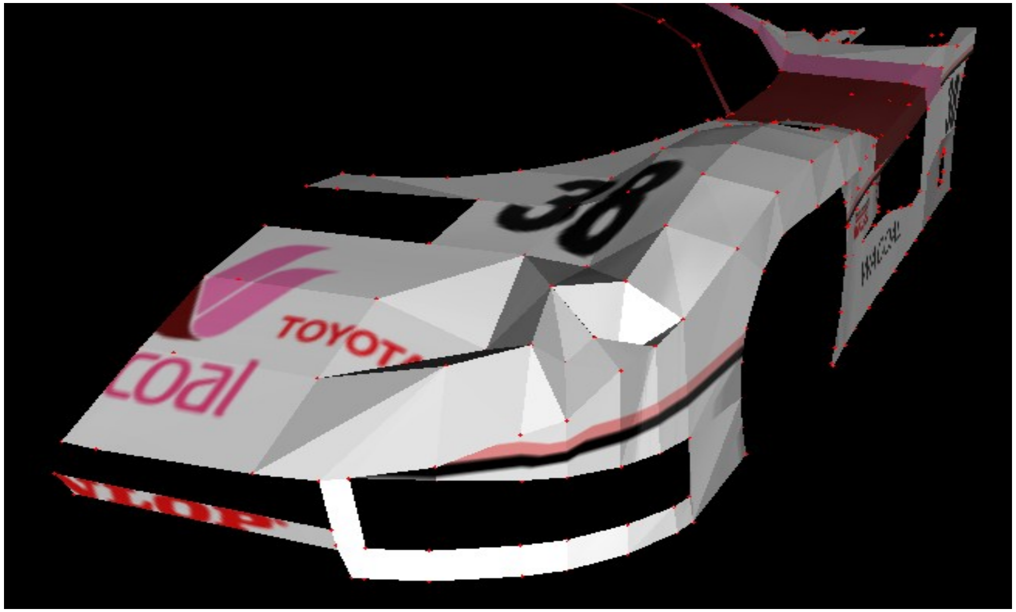
- Flat Shading (less computation needed)
- Gouraud shading
- Phong Shading (heavy computation needed)

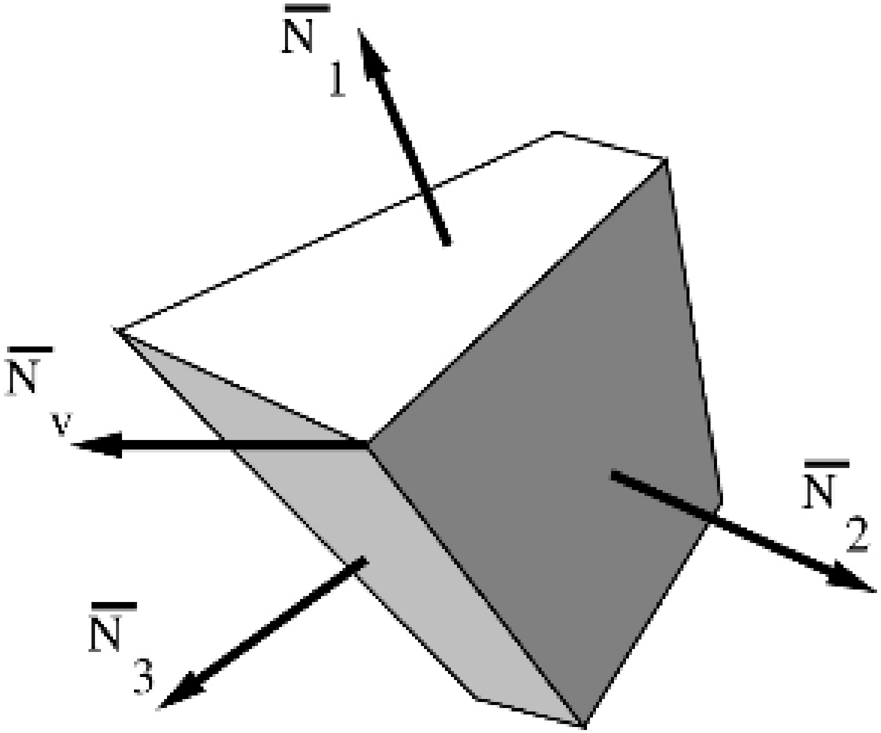
How to compute the normal vectors?

- The mesh is a set of polygons
- We can compute the normal vectors for each polygon (triangle)
- We can color the whole polygon using the same normal vector (flat shading)

Flat Shading

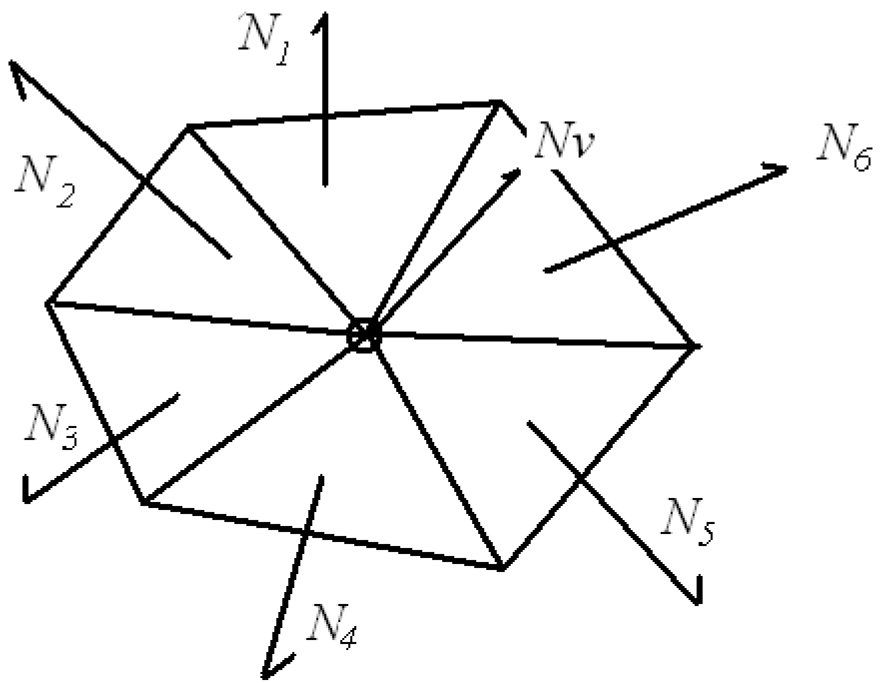
- Compute the color at the middle of the polygon
- All points in the same polygon are colored by the same color





Computing the normal vectors of the vertices

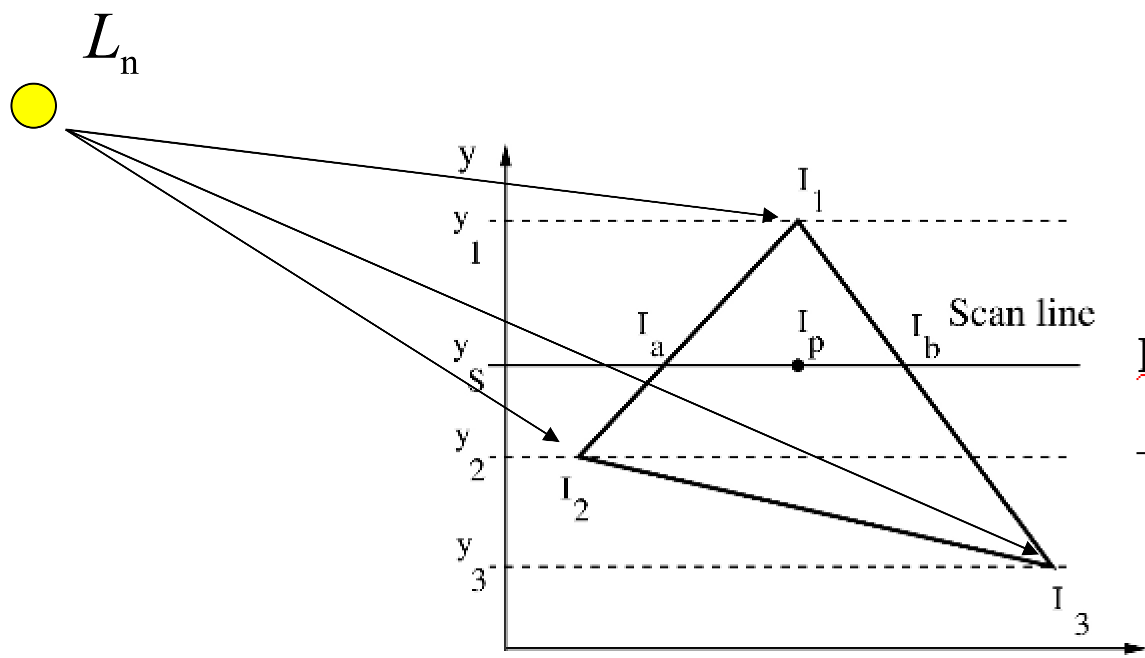
- We can compute the normals of the vertices by averaging the normal vectors of the surrounding triangles



$$Nv = (N_1 + N_2 + \dots + N_n)/n$$

Gouraud Shading (Smooth Shading)

- Compute the color at each vertex first
- Compute the color inside the polygon by interpolating the colors of the vertices composing the polygon

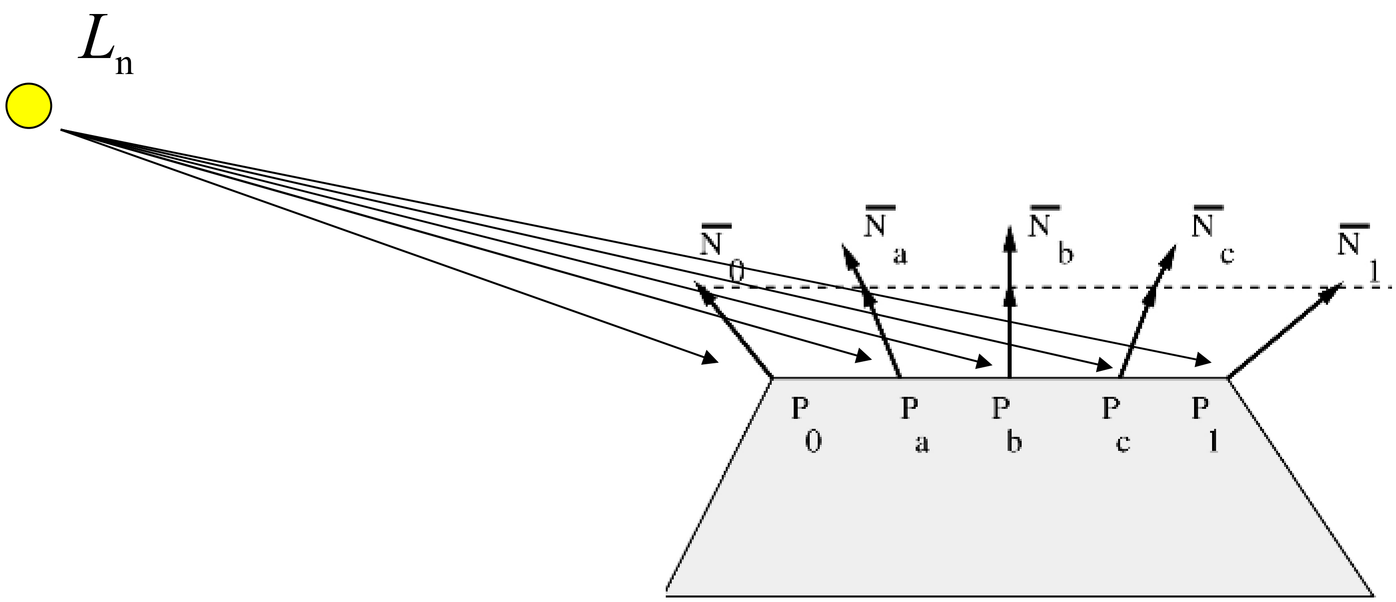


$$I_p = I_b - (I_b - I_a)\frac{x_b - x_p}{x_b - x_a}$$

Gouraud shading model

Phong Shading

- interpolating the normal vectors at the vertices
- Do the computation of illumination at each point in the polygon

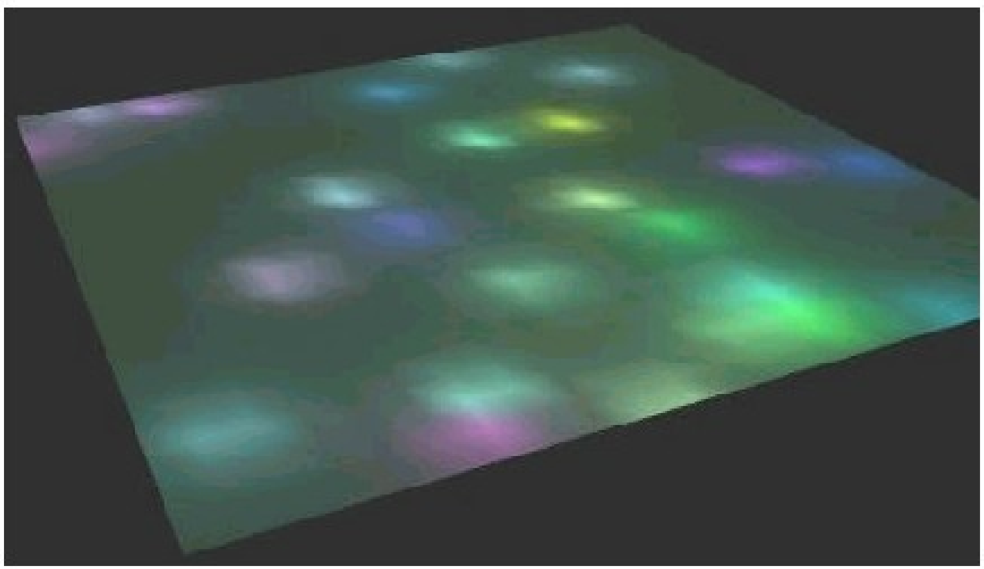


Phong shading model

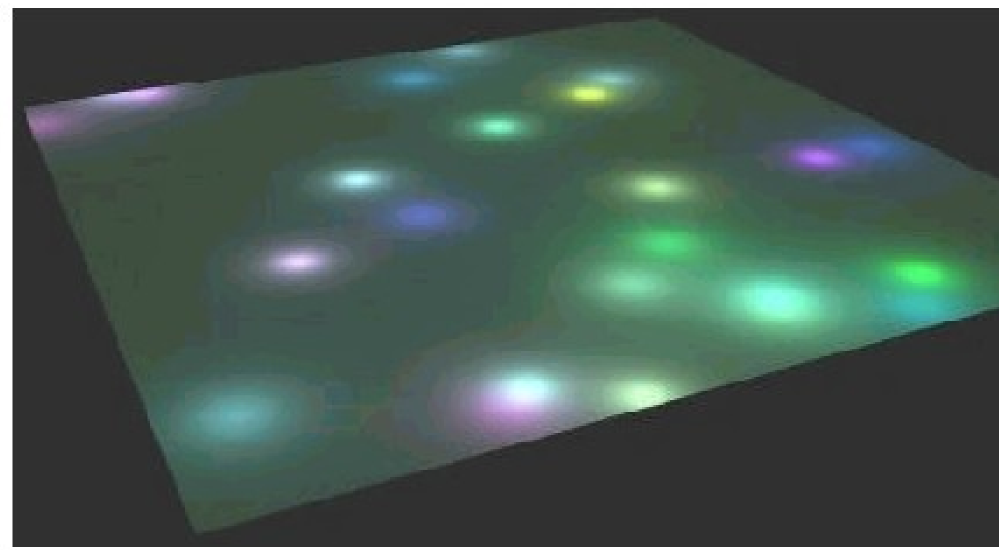
Flat shaded
Utah Teapot

Phong shaded
Utah Teapot

Flat shaded spot lights

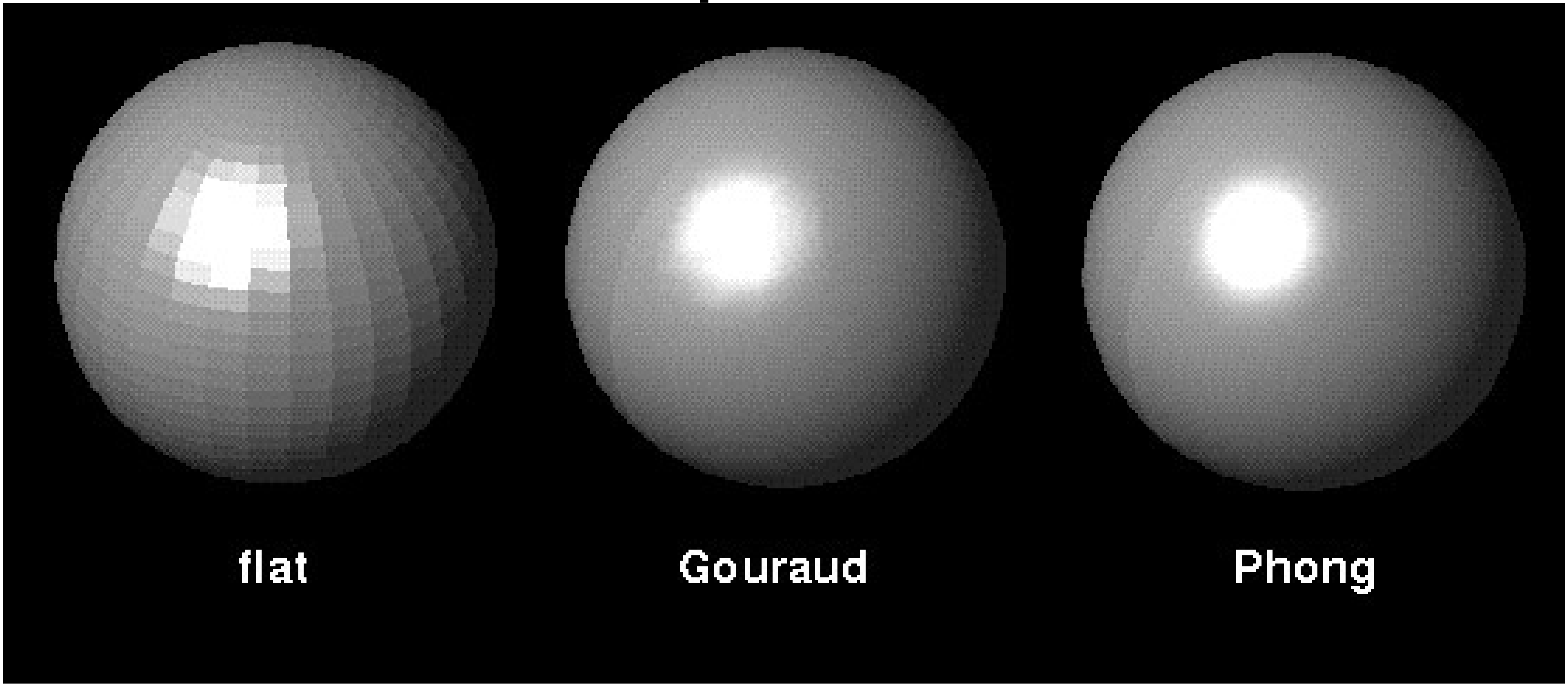


Phong shaded spot lights



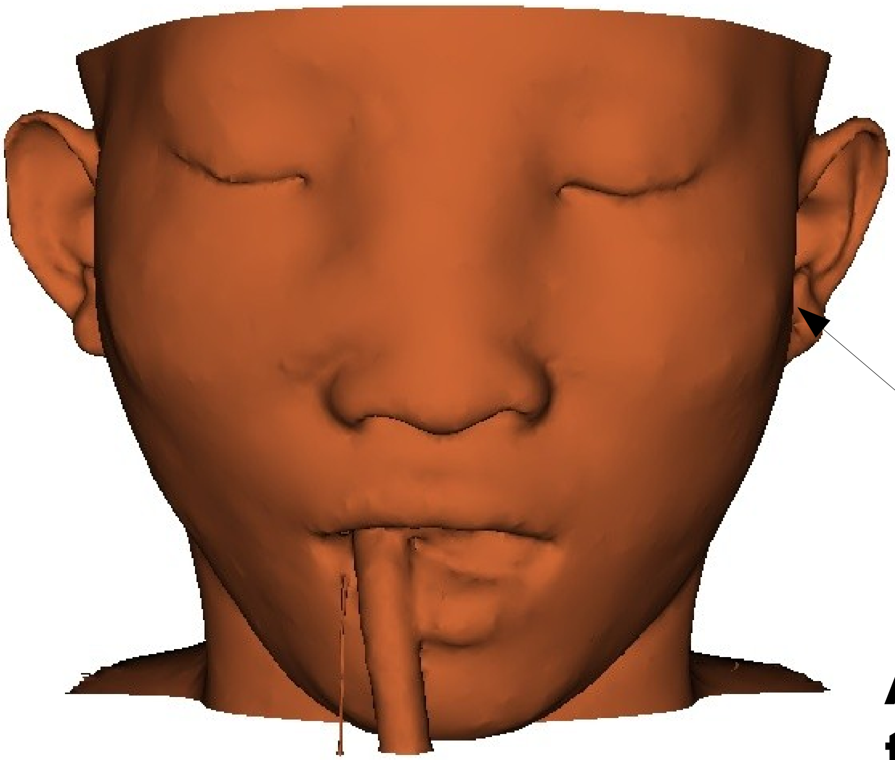
- Gouraud shading is not good when the polygon count is low

Comparison



- Summary
  - Phong shading is good but computationally costly
  - Flat shading is easy but results are too bad
  - Gouraud shading is usually used for simple applications

Surface Shape Perception - 1



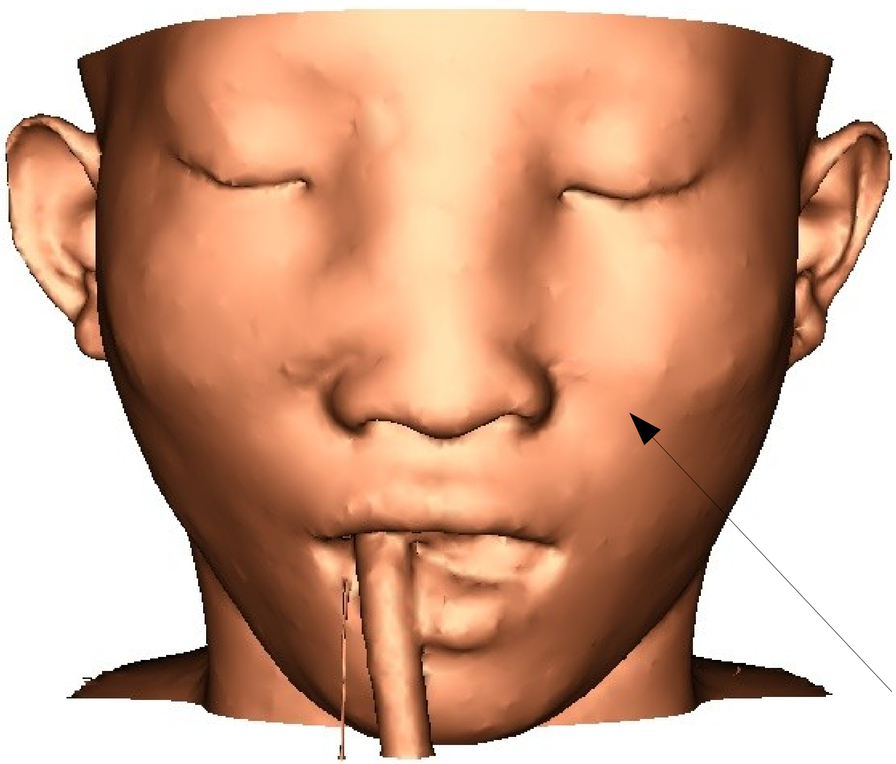
3D surface of the skin from a medical scanner.

Diffuse lighting only. Light is coming from the top front

Perpendicular to light

Area perpendicular to the light can be recognized well

Surface Shape Perception - 2



3D surface of the skin from a medical scanner.

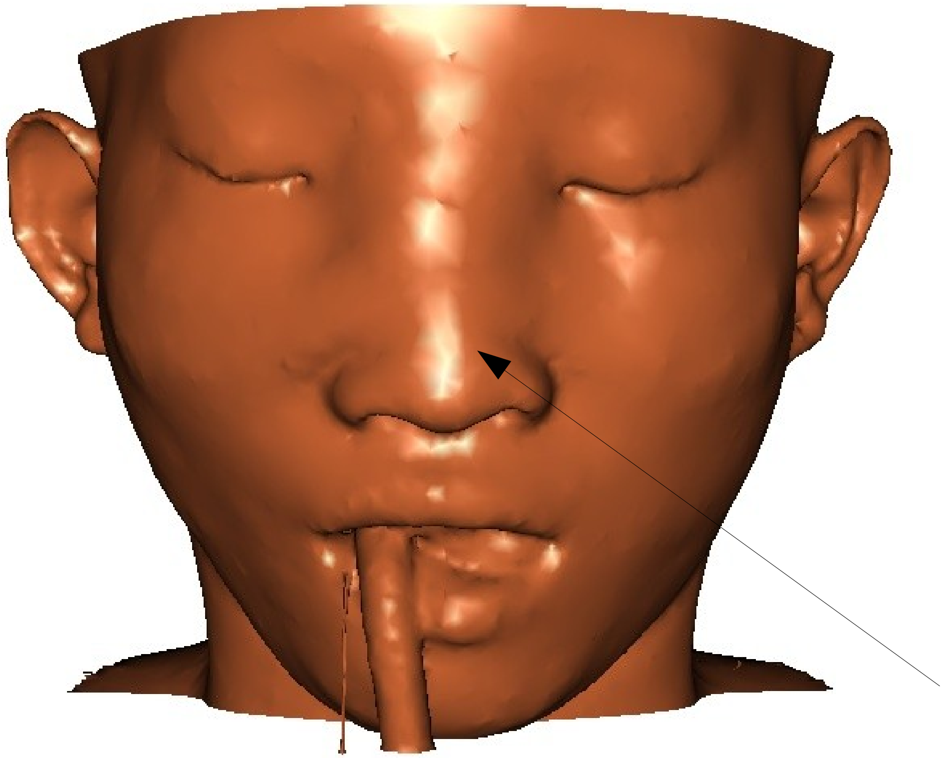
Diffuse + specular lighting.

Specular Power = 4.0

Edge of highlight

Area at the Edge of the highlight can be recognized well

Surface Shape Perception - 3



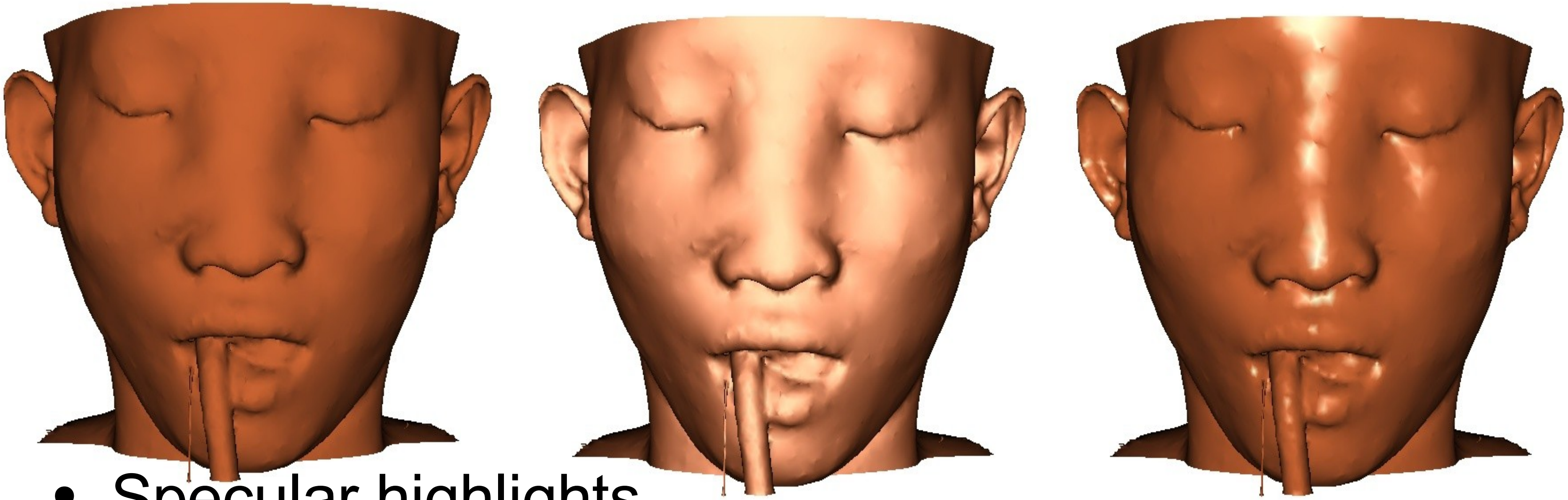
3D surface of the skin from a medical scanner.

Diffuse + specular lighting.

Specular Power = 200.0

Edge of highlight

Perception of Shape



- Specular highlights
  - improve perception of surface shape features (e.g. nose)
  - ... but only where the highlight occurs

Enhancing the Perception of Shape

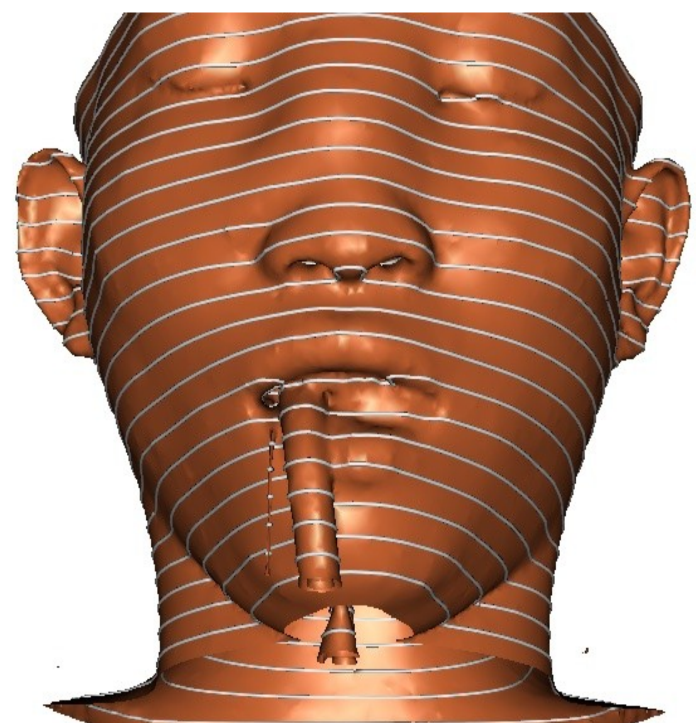
- Specular highlights
  - We can
    - dynamically change the specular power,
    - Rotate the light
    - Rotate the viewpoint

    to enhance the perception of the shape

    >> changing the edge of the highlight

Other cues to shape

- **Texture**
  - The motion/direction of lines or patterns on the surface of the shape
- **Stereo**
  - Viewing depth with 2 eyes
  - Stereo displays frequently used for visualisation

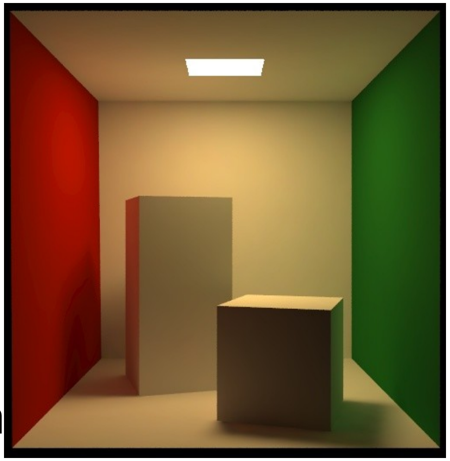


Scene Coordinate Systems

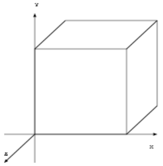
Object Coordinates

Object A's Transform

World Coordinates
-Light Position
-Camera Position
-Object Position(s)
3D (x,y,z)



Object A's
Coordinate System
3D (x,y,z)

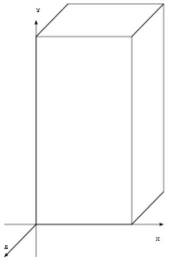


Camera transform
- 4x4 Matrix (3D→2D)

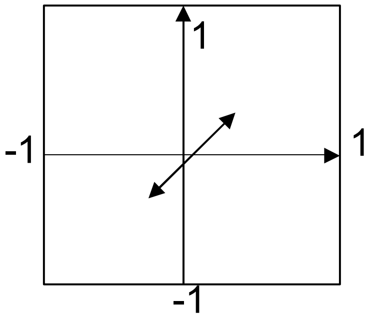
Display Coordinates

Objects B's Transform

Object B's
Coordinate System
3D (x,y,z)



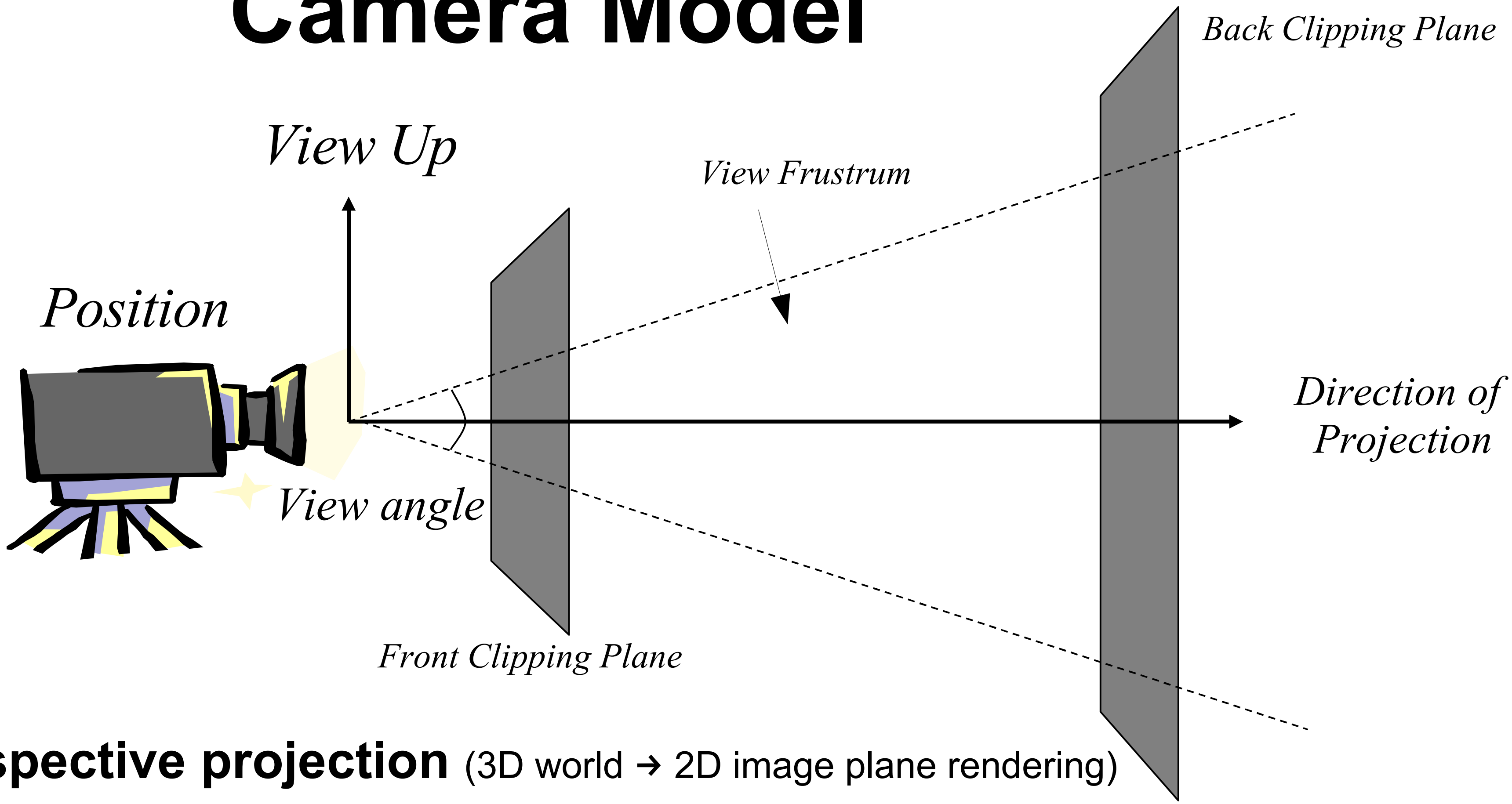
View Coordinates



Display transform
-Window size
-Position
2D (x,y)

- Object / World co-ordinates: **transformations and rotations in $\mathbb{R}^3$**
- Camera transform: 3D→2D matrix transformation (perspective projection).

Camera Model



- **Perspective projection** (3D world → 2D image plane rendering)
  - all rays into camera go through a common point
- **View Frustrum** – 3D space viewable to camera
  - bound by clipping planes (front, back); by camera view angle (top, bottom)
  - clipping planes eliminate data that is too near or too distant from camera

Summary

- Computer Graphics (basics)
  - representing object geometry as **polygon meshes**
  - **illumination models** (ambient, diffuse, specular)
  - Shading models (flat, Gouraud, Phong)
  - **camera model & projection**
- Next lecture: systems architectures for visualisation